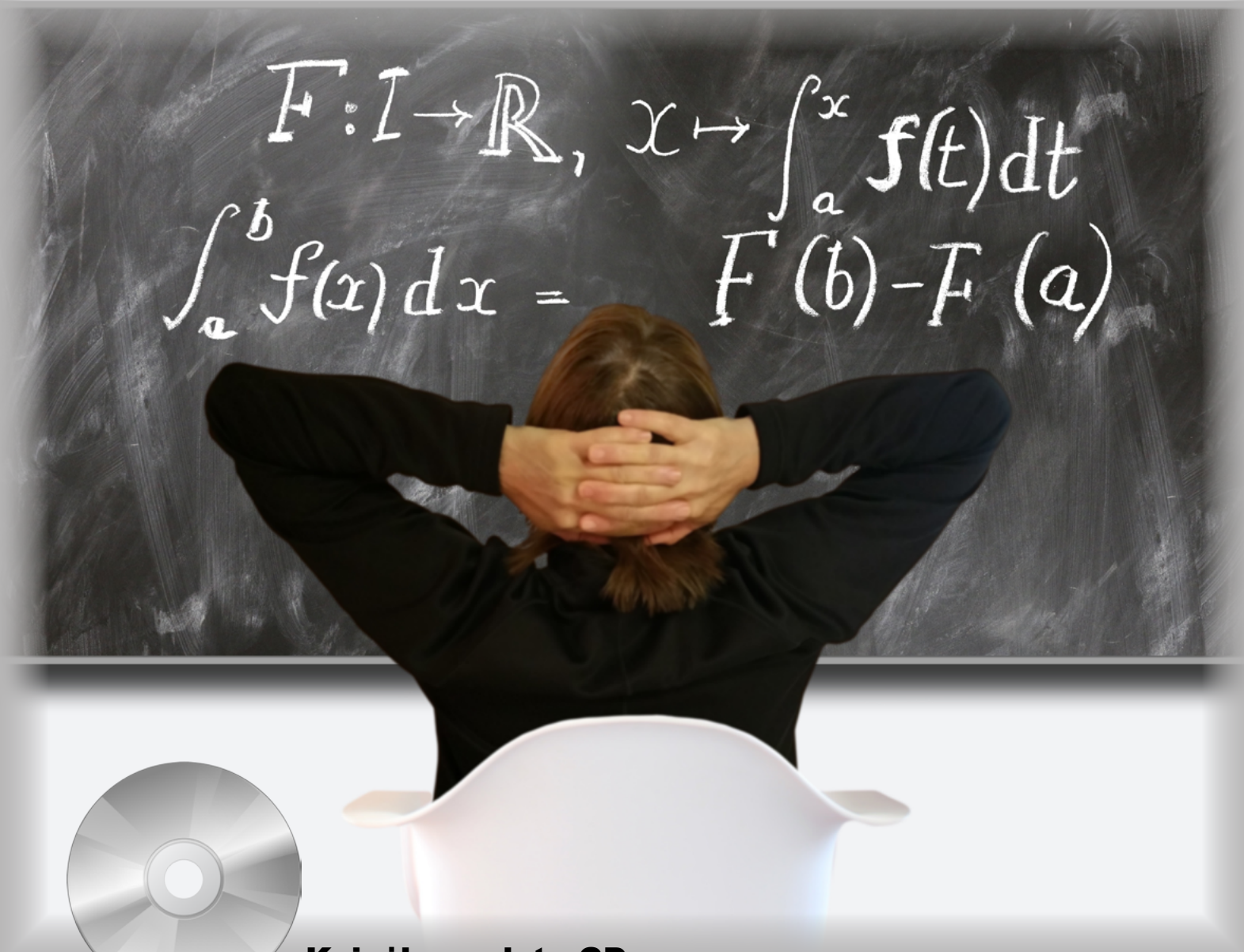
Program edukacyjny

do nauki znajdowania elementów rachunku różniczkowego – pochodne funkcji

Sławomir Guzewski



Książka z płytą CD

Exante

Recenzent
prof. zw. dr hab. Józef Banaś

PROGRAM EDUKACYJNY DO NAUKI ZNAJDOWANIA ELEMENTÓW RACHUNKU RÓŻNICZKOWEGO – POCHODNE FUNKCJI

exante.com.pl, wydawnictwoexante.pl, Wrocław 2018

Wersja elektroniczna publikacji jest wersją pierwotną

Książka z płytą CD

Program można również pobrać z Internetu, pod adresem:
https://drive.google.com/file/d/1BTsx1PvOoOqxTHQbsmJIlG60S3YEf_Bw/view?usp=sharing

Książka powstała na podstawie pracy inżynierskiej Autora obronionej w 2004 r.
na Wydziale Inżynierii Mechanicznej i Informatyki Politechniki Częstochowskiej.

Wydawnictwo Exante

Exante Wydawnictwo Naukowe
dr Klaudia Pujer
ul. Buforowa 24 lok. 10, 52-131 WROCŁAW

tel. + 48 606 168 165
wydawnictwo@exante.com.pl
www.exante.com.pl

Ark. wyd. 27,7
ISBN 978-83-66187-15-3 (PDF)
ISBN 978-83-66187-14-6 (oprawa miękka)
ISBN 978-83-66187-16-0 (CD)

Program edukacyjny

do nauki znajdowania elementów rachunku różniczkowego – pochodne funkcji

Sławomir Guzewski

Exante
Wrocław 2018

Spis treści

I. Wstęp

Komputerowe programy edukacyjne są z założenia programami wspomagającymi nauczanie oraz utrwalanie zdobytej wiedzy. Wyposażone są najczęściej w elementy multimedialne, wzbogacające pracę programu kolorowymi animacjami czy komentarzem lektorskim. Program, będący tematem niniejszej książki, ma na celu nauczanie znajdowania pochodnych funkcji i utrwalania tej umiejętności przez prezentacje gotowych lub wpisanie własnych przykładów funkcji do rozwiązania i pomoc w znalezieniu pochodnej poprzez ukazanie odpowiednio dobranych podpowiedzi. Powinien więc zainteresować zarówno młodzież szkół średnich, jak i studentów uczących się w wyższych szkołach technicznych czy wyższych szkołach inżynierskich. Program wyposażony został w edytor równań, który automatycznie układa funkcję na ekranie w sposób matematyczny po każdym wprowadzonym znaku z klawiatury.

Celem tej książki jest nie tylko prezentacja programu jako gotowego produktu, ale także pokazanie jego budowy w postaci dokładnego opisu zastosowanych instrukcji, uzupełnionych kodem źródłowym wyposażonym w liczne komentarze. Książka podzielona jest na 15 rozdziałów, z których drugi jest prezentacją programu, natomiast pozostałe rozdziały omawiają jego budowę. Każdy kolejny rozdział jest opisem jednego z dziewięciu modułów, z których składa się program. Najtrudniejszą jego częścią jest edytor równań, któremu poświęcono dwa ostatnie rozdziały. Na końcu książki znajduje się bibliografia, z której korzystano podczas pisania niniejszej książki. Mając tak dokładną dokumentację, Czytelnik będzie mógł sam spróbować napisać własną wersję programu, rozbudować istniejący program o nowe możliwości lub opisywany program przetworzyć na inny język programowania. Oczywiście książka może też mieć charakter dydaktyczny, pozwalający poznać tajniki programowania, szczególnie sposób przydzielania i zwalniania pamięci dla obiektów wykorzystywanych w programie czy operowania dynamicznymi strukturami danych. Zawarte w książce wskazówki i ostrzeżenia pozwolą uniknąć błędów, które często popełniają początkujący programiści. Do wydania papierowego książki dołączono płytę CD zawierającą skompilowany program, gotowy do uruchomienia w środowisku Windows. Program można również pobrać z Internetu, pod adresem:
https://drive.google.com/file/d/1BTsx1PvOoOqxTHQbsmJIlG60S3YEf_Bw/view?usp=sharing

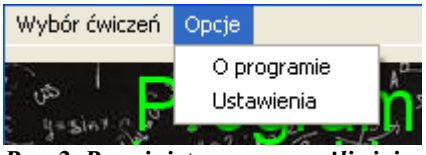
II. Działanie programu

Po uruchomieniu programu powinna ukazać się na ekranie forma powitalna (rysunek 1):

Rys. 1. Forma powitalna

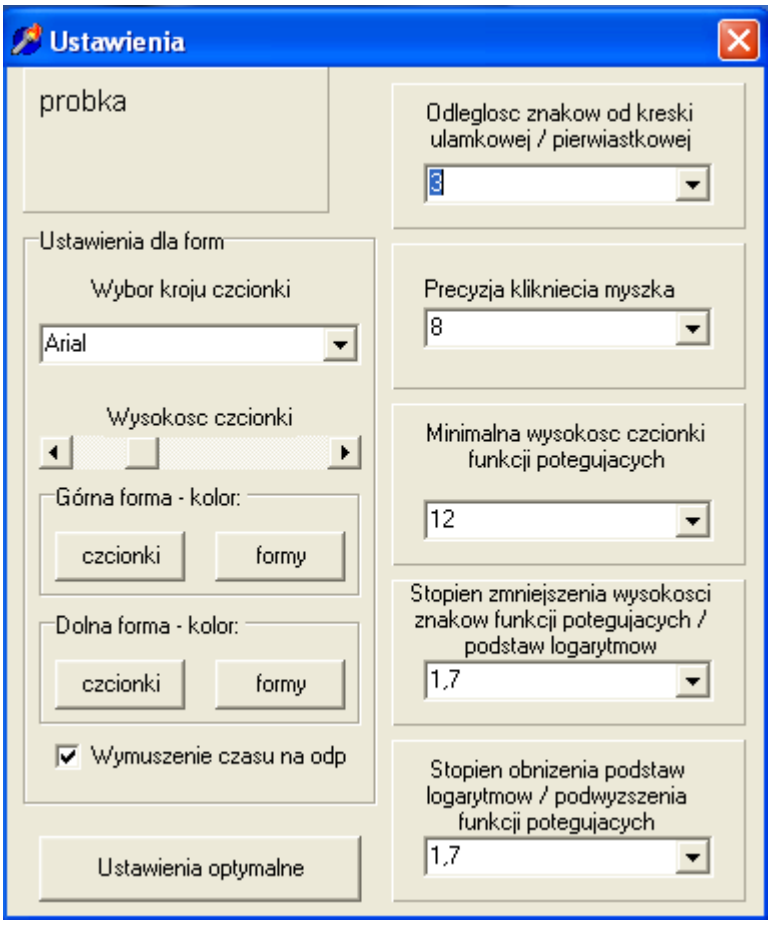
Widoczny w prawym, górnym rogu krzyżyk umożliwia zamknięcie programu. Brak pozostałych ikon świadczy o braku możliwości zminimalizowania czy zmiany rozmiaru formy, można ją jedynie przesuwać myszką po ekranie.

Widoczne menu pozwala na wybór ćwiczeń lub opcji programu (rysunek 2).



Rys. 2. Rozwinięte menu umożliwiające wybór ustawień programu lub wyświetlenie pliku pomocy

Wybór „`Opcje`” z dostępnego menu powoduje rozwinięcie podmenu, z którego można wybrać pozycję „`O programie`”, w wyniku której powinien otworzyć się plik pomocy. Po wyborze pozycji „`Ustawienia`” powinna otworzyć się forma ustawień z licznymi elementami regulacyjnymi (rysunek 3).



Rys. 3. Forma ustawień

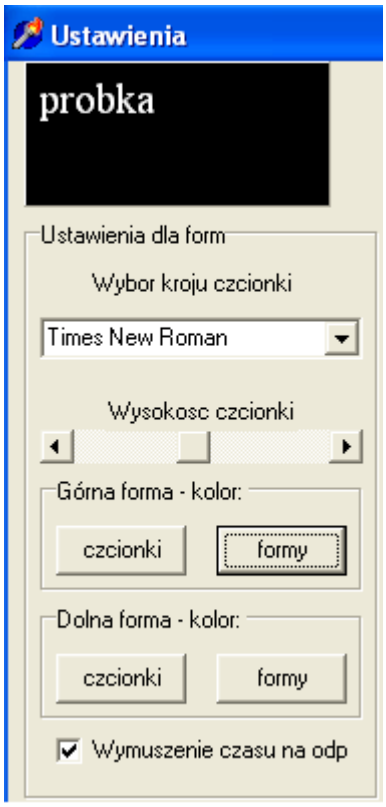
Elementy regulacyjne są pogrupowane na następujące kategorie:

- dotyczące wyłącznie form roboczych;
- dotyczące regulacji wielkości, z których korzystają edytory równań.

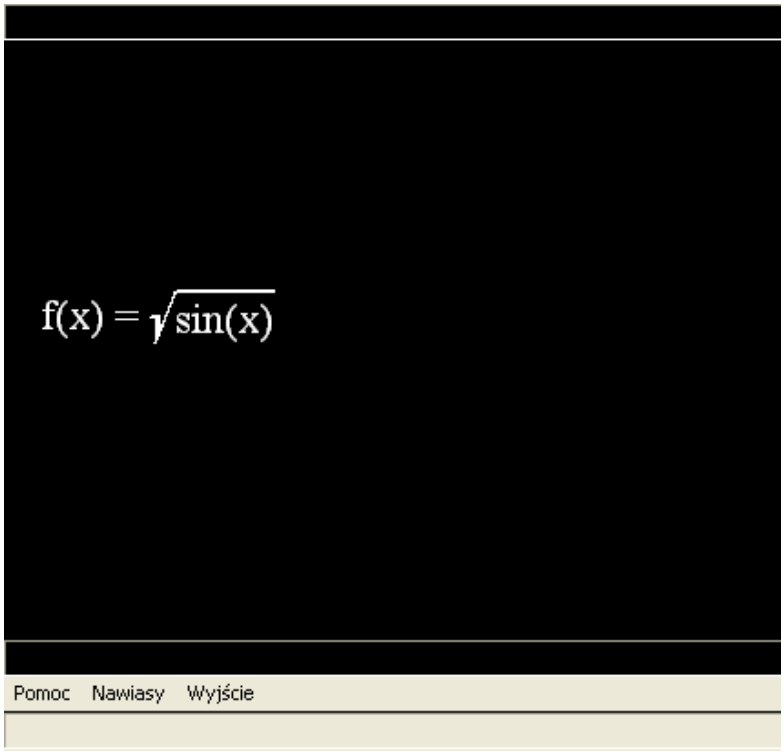
Ustawienia dotyczące form roboczych umożliwiają:

- wyboru kroju czcionki;
- wysokości czcionki;
- koloru formy i czcionki oddzielnie dla górnej i dolnej formy.

Za pomocą dodatkowego panelu, znajdującemu się w lewym, górnym rogu, możemy zaobserwować i wstępnie ocenić efekt swoich ustawień (rysunki 4a i 4b).



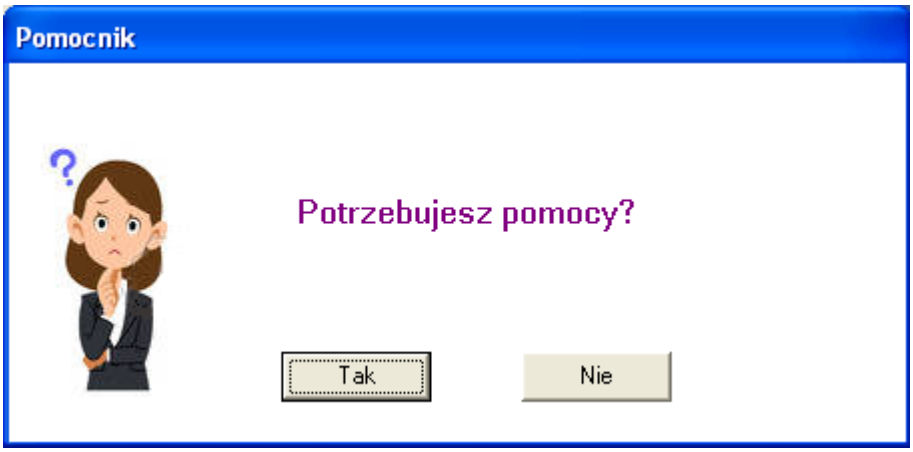
Rys. 4a. Zmiany kroju czcionki, rozmiaru czcionki oraz koloru czcionki formy górnej



Rys. 4b. Efekt ustawień widoczny na górnej formie roboczej (dolna forma posiada kolory standardowe)

Ustawienia kolorów: czcionki i tła formy dolnej przebiegają tak jak dla formy górnej.

Kontrolka „`Wymuszenie czasu na odp`” służy do zablokowania automatycznego pojawiania się monitu zachęcającego do skorzystania z pomocy. Automatyczne pojawianie się monitu ma na celu ponaglanie użytkownika, by ten w określonym czasie udzielił odpowiedzi – poprawnej lub błędnej (rysunek 5). Czas jaki daje użytkownikowi program na wpisanie pochodnej zależny jest od liczby znaków, z których składa się pochodna znaleziona przez program. Zablokowanie automatycznego pojawiania się monitu polega na odznaczeniu tej kontrolki (domyślnie jest ona zaznaczona).



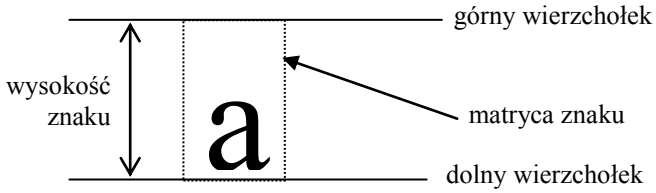
Rys. 5. Zachęcenie przez program do skorzystania z pomocy

Powyższy monit można także wywołać ręcznie, klikając dwukrotnie w dolną formę w czasie ćwiczeń.

Ustawień dotyczących wielkości, z których korzystają edytory równań, nie można zaobserwować w panelu kontrolnym – ich efekty można dostrzec jedynie po wyświetleniu form roboczych, czyli po wyborze jednej z czterech możliwości pracy programu. Ustawienia dotyczą następujących wielkości:

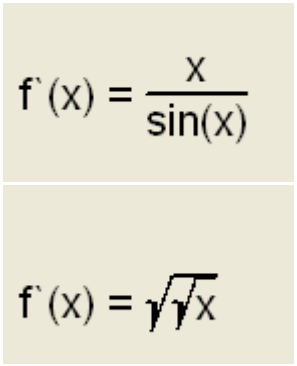
1. Odległość znaków od kreski ułamkowej/pierwiastkowej

W programie, wysokość znaku jest zawarta między dolnym i górnym wierzchołkiem matrycy, w której osadzony jest znak. Oznacza to, że rzeczywista wysokość znaków, szczególnie małych liter, jest zazwyczaj większa niż ich widok na ekranie – wyjaśnia to rysunek 6:

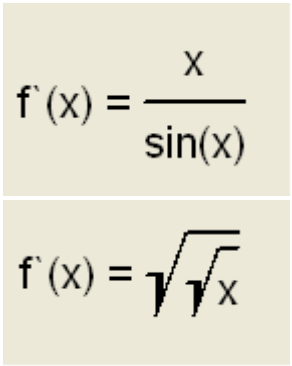


Rys. 6. Zasada określania wysokości znaku

W związku z tym, każda regulacja tego typu musi uwzględniać rozmiar matrycy w której znak jest umieszczony. Efekty regulacji dla dwóch skrajnych wartości przedstawiają rysunki 7a i 7b:



Rys. 7a. Wygląd ułamka i pierwiastka przy minimalnych ustawieniach odległości znaku od kreski



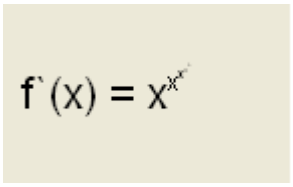
Rys. 7b. Wygląd ułamka i pierwiastka przy maksymalnych ustawieniach odległości znaku od kreski

2. Precyzja kliknięcia myszką polega na wyborze takiej wartości, przy której kliknięcie myszką spowoduje poprawne ustawienie kursora przy tym znaku i z tej jego strony, które są najbliższe miejscu kliknięcia. Zdarza się, że regulacja ta jest konieczna, dlatego została udostępniona.

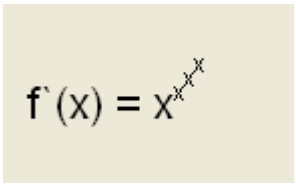
3. Minimalna wysokość czcionki funkcji potęgującej

Podczas pisania funkcji potęgujących, jej znaki są zmniejszane w odpowiednim stosunku względem funkcji potęgowanej. Zdarza się, że funkcja potęgująca jest wielokrotna (potęga do potęgi), dlatego zmniejszanie znaków, bez ustalenia tzw. progu zmniejszania, może do-

prowadzić do tego, że staną się niewidoczne na ekranie, dlatego ustalenie takiego progu zmniejszania jest jak najbardziej uzasadnione. Rysunki 8a i 8b najlepiej zobrazują efekt tej regulacji:



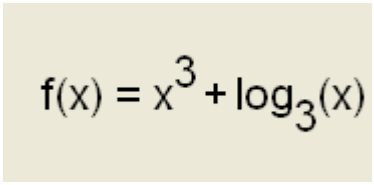
Rys. 8a. Efekt przy wyborze najmniejszej wartości minimalnej dla funkcji potęgującej

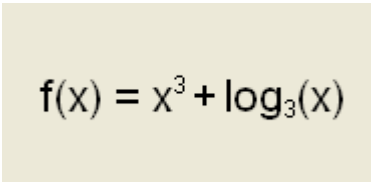


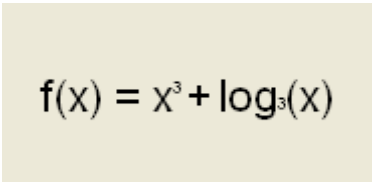
Rys. 8b. Efekt przy wyborze największej wartości minimalnej dla funkcji potęgującej

4. Stopień zmniejszenia wysokości znaków funkcji potęgujących/podstaw logarytmów.

Skalowanie elementów funkcji potęgującej czy podstawy logarytmu odbywa się względem wysokości ostatniego znaku funkcji potęgowanej czy funkcji log. Efekt tej regulacji na przykładzie funkcji potęgującej i logarytmu ilustruje rysunek 9:



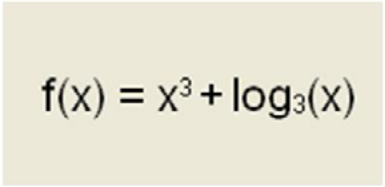


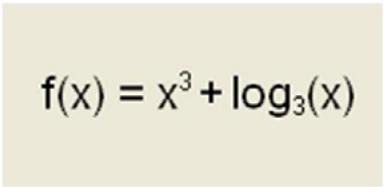


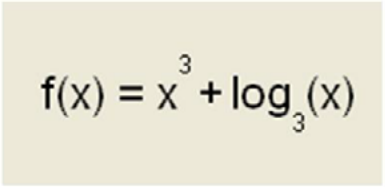
Rys. 9. Wygląd funkcji potęgującej i logarytmu dla trzech wybranych współczynników zmniejszenia
A – dla minimalnej wartości równej 1,0
B – dla typowej wartości równej 1,7
C – dla maksymalnej wartości równej 2,2

5. Stopień obniżenia podstaw logarytmów/podwyższenia funkcji potęgujących

Pozycjonowanie elementów funkcji potęgującej czy podstawy logarytmu odbywa się względem pozycji ostatniego znaku funkcji potęgowanej czy funkcji log. Tak jak w poprzednim przykładzie, rysunek 10 prezentuje rezultat regulacji na przykładzie funkcji potęgującej i logarytmu:





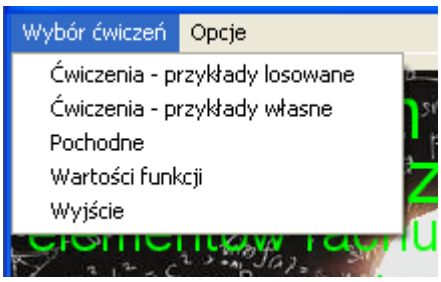


A B C

Rys. 10. Wygląd funkcji potęgującej i logarytmu dla trzech wybranych współczynników pozycjonowania
A – dla minimalnej wartości równej 1,5
B – dla typowej wartości równej 1,7
C – dla maksymalnej wartości równej 4,0

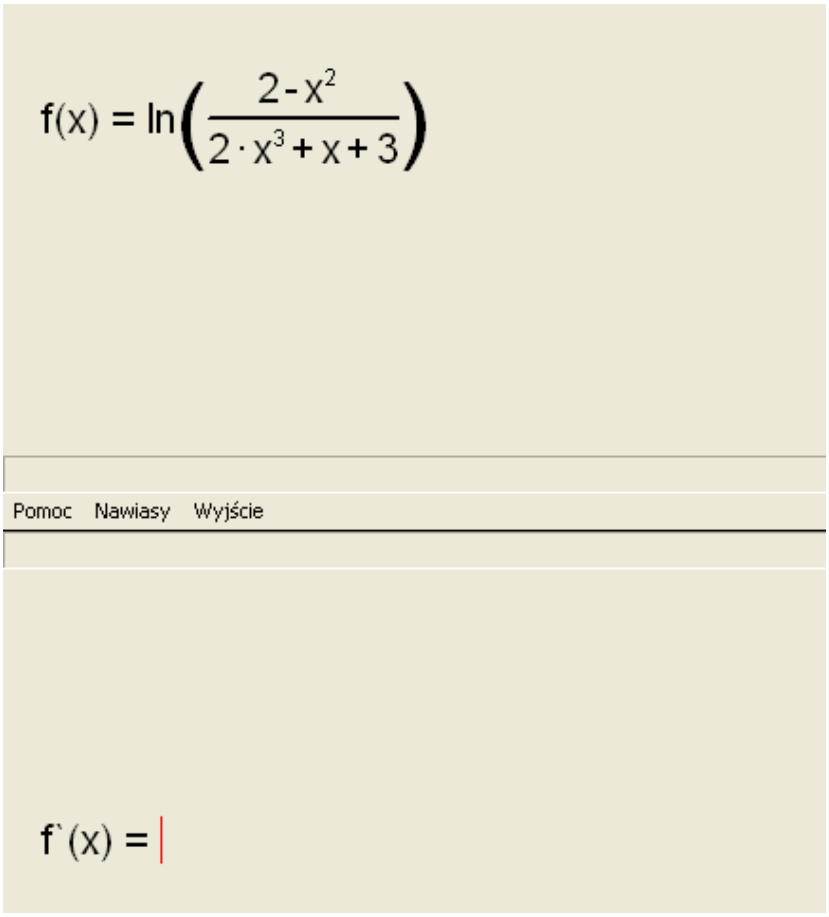
Przycisk „Ustawienia optymalne” przywraca wszystkie ustawiane wartości do tych, jakie były ustalone w fazie projektowania aplikacji. Przywracane są zarówno ustawienia dotyczące form roboczych, jak i wartości zmiennych, z których korzystają edytory równań.

W części menu „Wybór ćwiczeń” istnieje możliwość wyboru zarówno ćwiczeń, jak i pracy programu bez ćwiczeń, czyli wyświetlenia wyniku w postaci pochodnej lub wartości pełnej funkcji (rysunek 11).



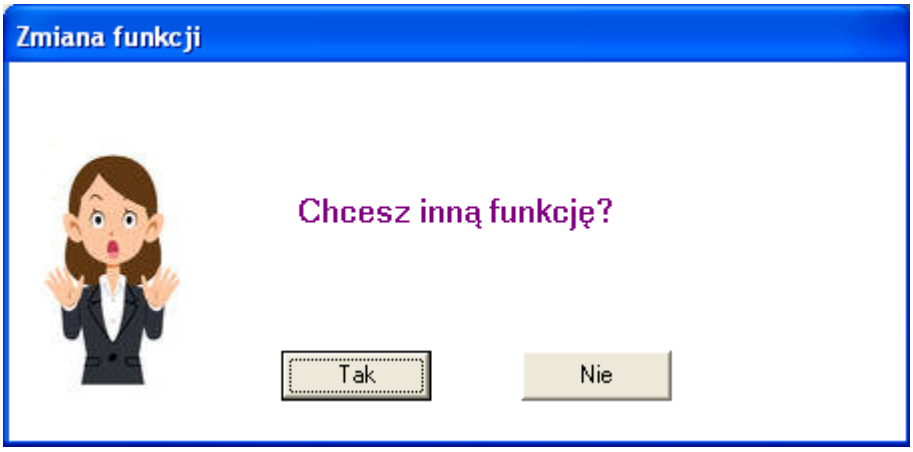
Rys. 11. Rozwinięte menu umożliwiające wybór ćwiczeń

Jeśli wybrana zostanie opcja „Ćwiczenia – przykłady losowane”, na ekranie wyświetlone zostają dwie formy – jedna pod drugą, z czego górna forma przeznaczona jest na wyświetlanie wylosowanej przez program funkcji natomiast dolna forma – na wpisywanie przez użytkownika pochodnych funkcji (rysunek 12).



Rys. 12. Fragmenty sąsiadujących form roboczych z widoczną, wylosowaną funkcją w górnej formie i gotowością na wprowadzenie pochodnej w dolnej formie

Jeśli wylosowana funkcja okaże się dla użytkownika nieodpowiednia, może ją zmienić przez kliknięcie myszką w górną formę lub przez naciśnięcie klawisza ENTER w sytuacji, gdy w dolnej formie użytkownik nie wprowadził ani jednego znaku pochodnej (rysunek 13).

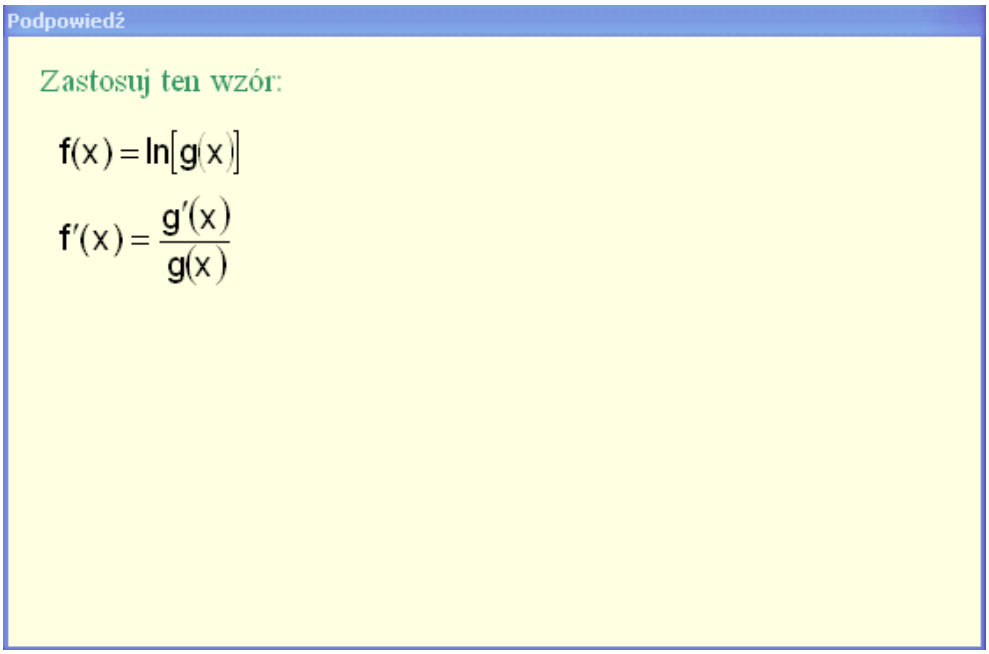


Rys. 13. Monit umożliwiający zmianę funkcji

Pojawienie się monitu zatrzymuje program, dając użytkownikowi możliwość wyboru poprzez kliknięcie myszką w jeden z przycisków osadzonych na formie bądź naciśnięcia klawisza ENTER, które w powyższym monicie jest jednoznaczne z naciśnięciem przycisku „Tak".

Po wybraniu „Tak", program wylosuje następną funkcję. Jeśli użytkownik próbował napisać pochodną i wprowadził co najmniej jeden znak, program usunie z niej wszystkie znaki i przygotuje formę dla następnego przykładu.

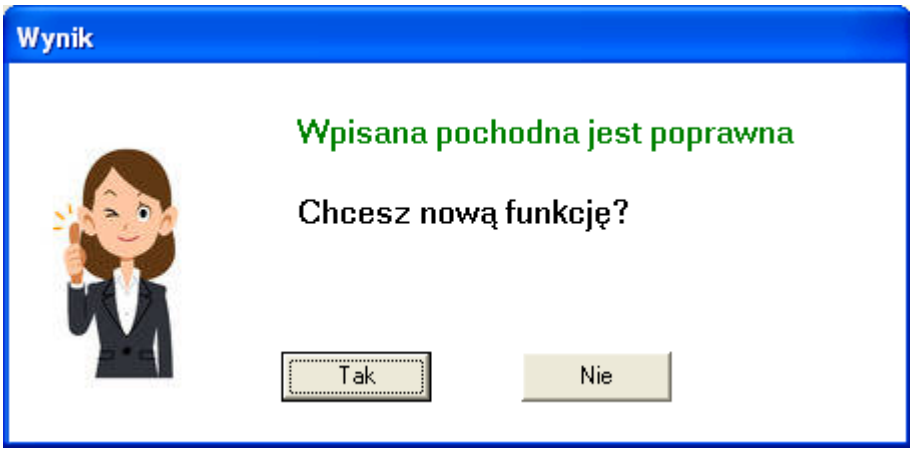
Kliknięcie myszką w dolną formę spowoduje wyświetlenie monitu zachęcającego użytkownika do skorzystania z pomocy. Monit ten jest taki sam jak ten, zamieszczony na rysunku 5. Kliknięcie myszką w przycisk „Tak" lub naciśnięcie na klawiaturze klawisza ENTER spowoduje pojawienie się formy podpowiedzi w prawym, górnym rogu ekranu.



Rys. 14. Forma podpowiedzi – pierwsza podpowiedź

Od tej chwili użytkownik powinien stosować się do wskazówek pojawiających się na formie podpowiedzi. Uruchomiony został także system edukacji dla wylosowanego przykładu funkcji (rysunek 14).

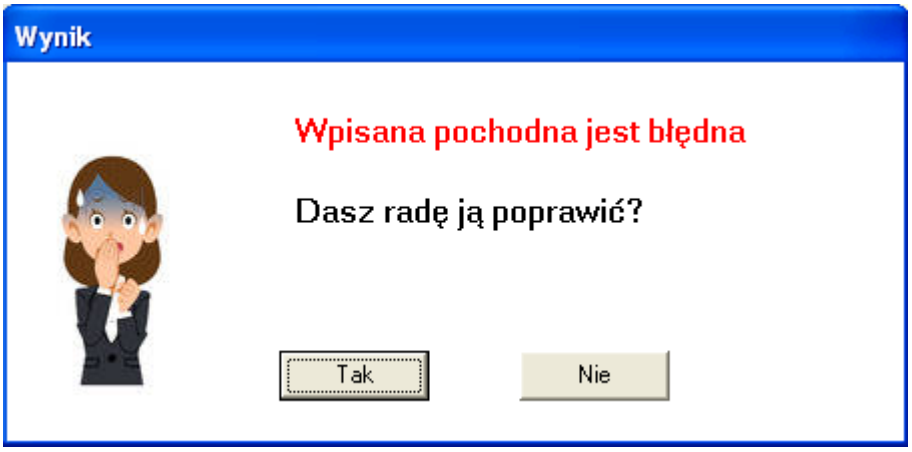
Jeśli wpisana przez użytkownika pochodna będzie poprawna, odpowiedni monit poinformuje go o tym i zaproponuje wylosowanie następnego przykładu (rysunek 15).



Rys. 15. Monit informujący o poprawnie wpisanej pochodnej dla pełnej funkcji

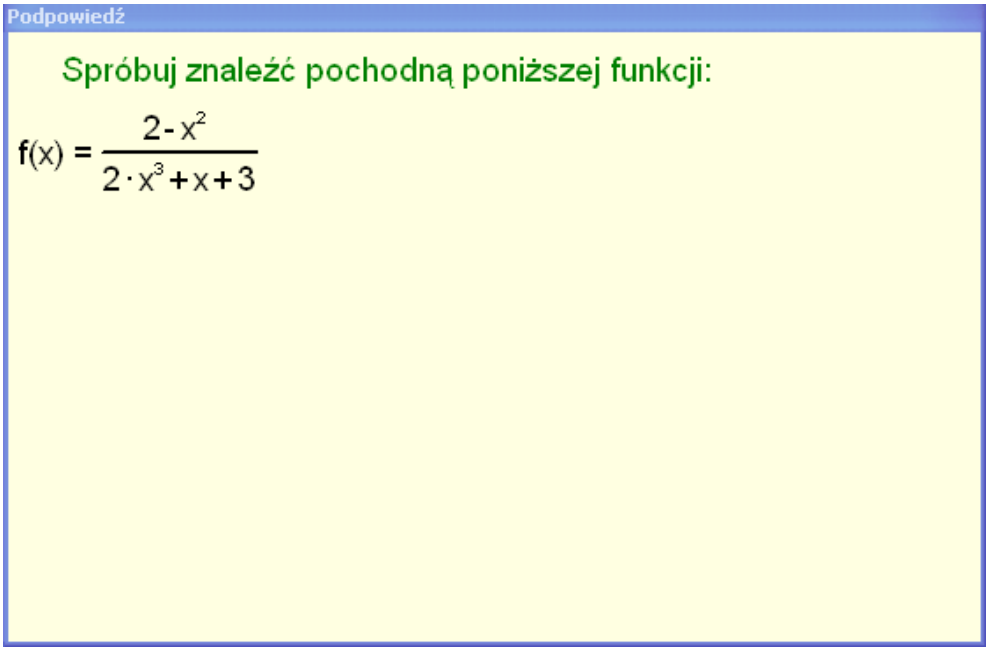
Taki sam monit zostanie wyświetlony, gdy użytkownik wpisze poprawną pochodną bez korzystania z podpowiedzi, lecz wtedy przypadek ten zostanie zsumowany jako poprawny i uwzględniony w podsumowaniu końcowym podczas wychodzenia z programu.

Jeśli pochodna wpisana przez użytkownika okaże się błędna, wyświetlony zostanie monit informujący o jego błędzie i zachęcający do skorzystania z pomocy (rysunek 16). Użytkownik będzie mógł spróbować sam poprawić błąd bez korzystania z pomocy wybierając kliknięciem przycisk z napisem „Nie” w wyświetlonym monicie.



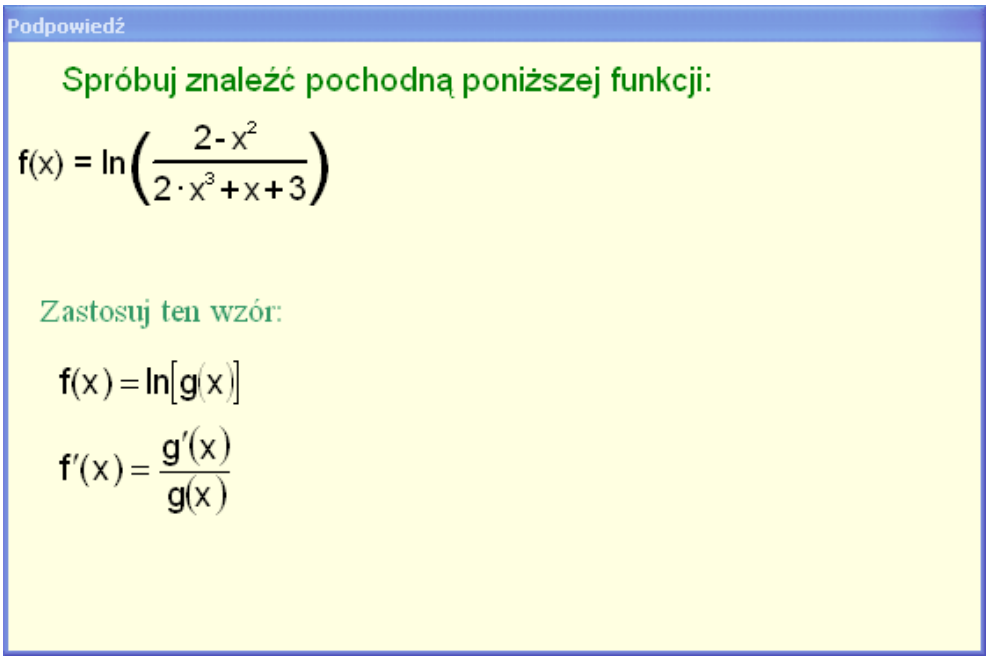
Rys. 16. Monit informujący o błędnie wpisanej pochodnej

Skorzystanie z pomocy uruchamia kolejny etap edukacji dla bieżącego przykładu – forma podpowiedzi wyświetli wówczas jedną z funkcji elementarnych wydzielonych z głównej funkcji bieżącego przykładu. Zadaniem użytkownika jest zatem wpisanie poprawnej pochodnej tej funkcji elementarnej, która pojawi się na formie podpowiedzi (rysunek 17).



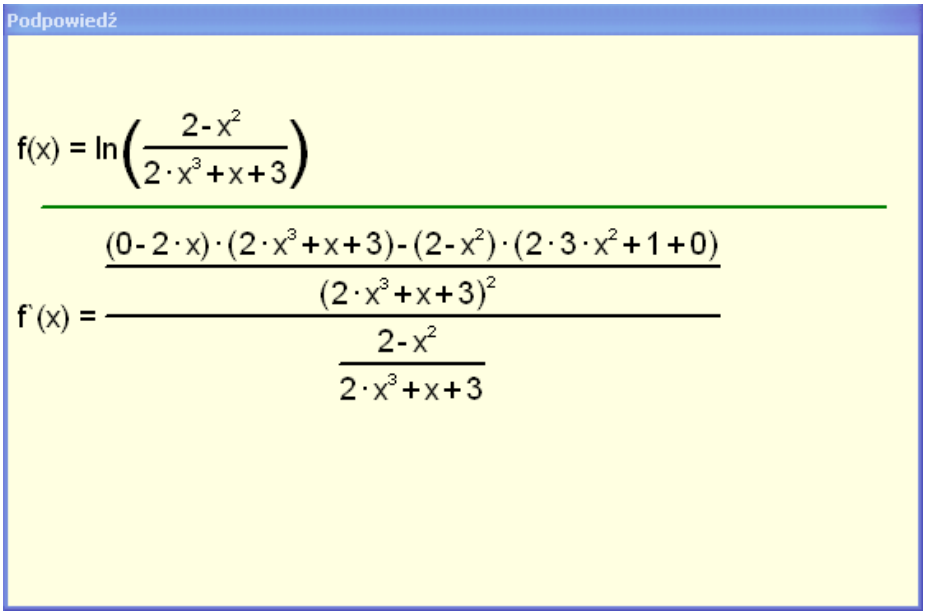
Rys. 17. Forma podpowiedzi – pierwsza funkcja elementarna i pierwszy etap podpowiedzi

Jeśli użytkownik wpisze poprawną pochodną funkcji elementarnej, forma podpowiedzi zaproponuje wpisanie użytkownikowi pochodnej pełnej funkcji rozpatrywanego przykładu. Przy braku powodzenia, forma podpowiedzi zaproponuje wzór, który powinien pomóc użytkownikowi w napisaniu poprawnej pochodnej (rysunek 18).



Rys. 18. Forma podpowiedzi – pełna funkcja i drugi etap podpowiedzi

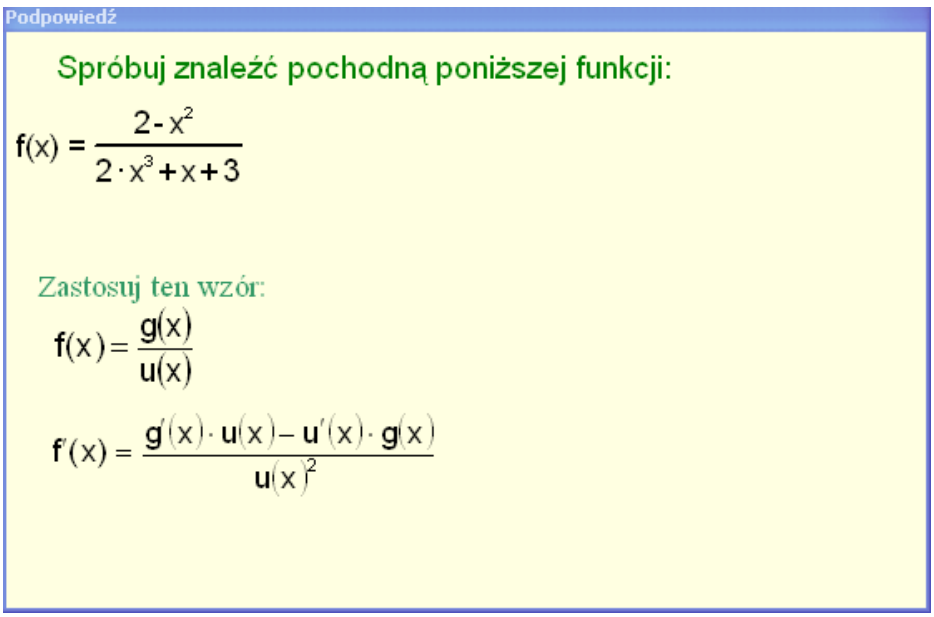
Jeśli użytkownik ponownie popełni błąd, forma podpowiedzi zaprezentuje poprawną pochodną (rysunek 19).



Rys. 19. Forma podpowiedzi – trzeci i końcowy etap podpowiedzi dla pełnej funkcji

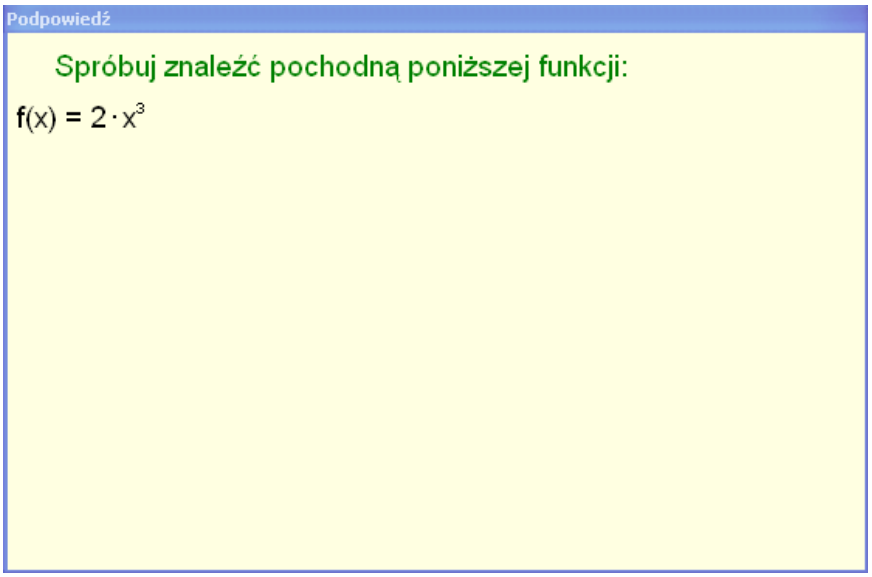
Na przykładzie pochodnej znalezionej przez program i wyświetlonej na formie podpowiedzi, użytkownik może przeanalizować swój błąd.

Na rysunku 20 zaprezentowana została pierwsza funkcja elementarna, którą program za pośrednictwem formy podpowiedzi zaproponował użytkownikowi, by ten spróbował wpisać jej pochodną. Dalsza reakcja programu na poprawnie wpisaną pochodną została opisana powyżej. Gdyby użytkownik nie poradził sobie z wpisaniem poprawnej pochodnej funkcji elementarnej zaproponowanej przez system edukacji, reakcją programu byłoby przejście do drugiego etapu podpowiedzi, czyli wyświetlenia wzoru do zastosowania w tym konkretnym przypadku.



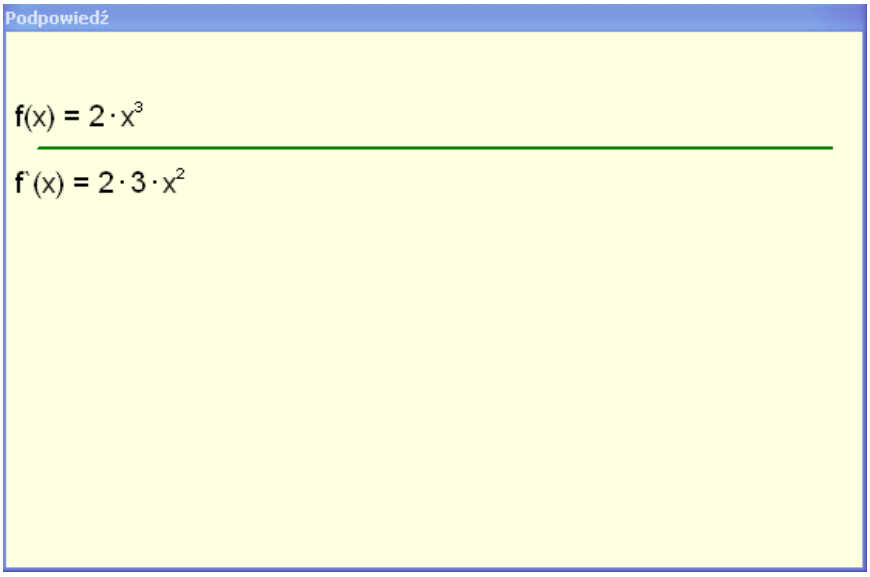
Rys. 20. Forma podpowiedzi – pierwsza funkcja elementarna i drugi etap podpowiedzi

Gdyby użytkownik i tym razem nie poradził sobie z poprawnym wpisaniem pochodnej, program zaprezentuje kolejną funkcję elementarną (rysunek 21).



Rys. 21. Forma podpowiedzi – druga funkcja elementarna i pierwszy etap podpowiedzi

Dalsza praca programu uzależniona jest od tego, czy użytkownik wpisze poprawną czy błędną pochodną. Ostatecznie, gdy program wykorzysta wszystkie możliwości, rozpoczyna się powrót, czyli ponowne wyświetlanie tych funkcji elementarnych, na które użytkownik nie wpisał poprawnej pochodnej. Po pierwszym i drugim etapie podpowiedzi program będzie uaktywniał trzeci jej etap, czyli wyświetlenie funkcji elementarnej a pod nią jej pochodnej (rysunek 22).



Rys. 22. Forma podpowiedzi – druga funkcja elementarna i trzeci etap podpowiedzi

Ostatecznie program zaprezentuje pełną funkcję i jej pochodną, co zostało uwidocznione na rysunku 19.

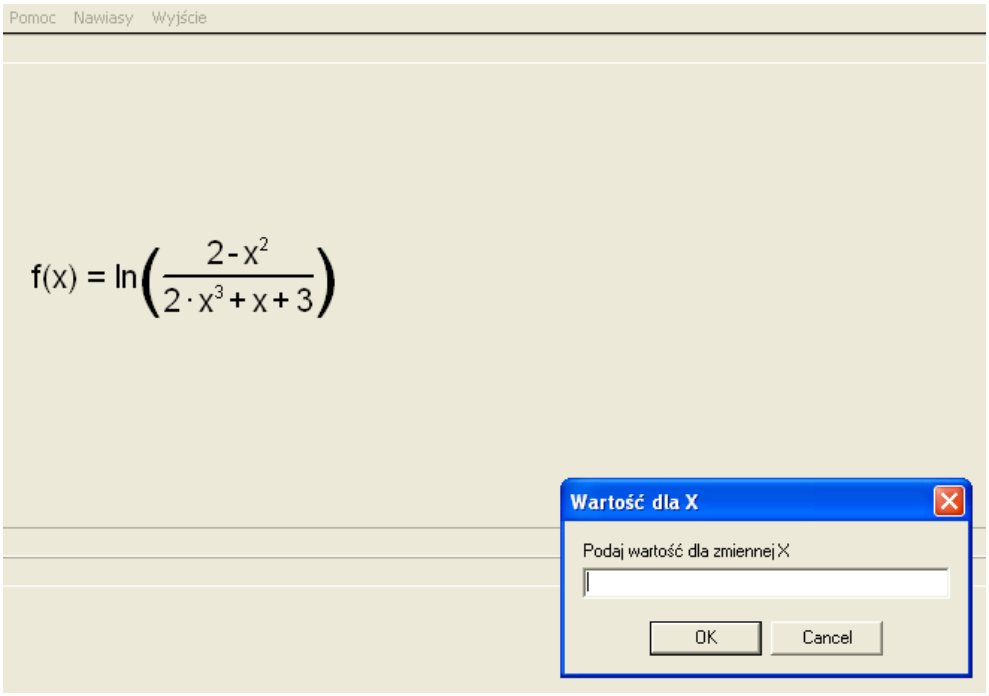
Opisany sposób pomocy użytkownikowi w poprawnym wpisywaniu przez niego pochodnych funkcjonuje także w drugim sposobie ćwiczeń na dowolnych przykładach, czyli „Ćwiczenia – przykłady własne”. Różnica między tym sposobem ćwiczeń a opisanym poprzednio polega na tym, że forma górna przeznaczona na wpisywanie funkcji, wymaga uaktywnienia poprzez kliknięcie w nią myszką i wpisania na niej własnego przykładu (rysunek 23).

Rys. 23. Rozpoczęcie pracy z formami roboczymi po wybraniu opcji „Ćwiczenia – przykłady własne” przed i po kliknięciu myszką w formę

Po poprawnym wpisaniu przez użytkownika funkcji w górnej formie roboczej, działanie programu jest identyczne z tym, opisanym powyżej dla funkcji wylosowanych przez komputer.

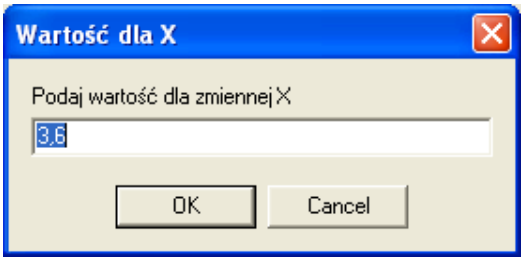
Jeśli z dostępnego menu, użytkownik wybierze pozycję „Pochodne”, praca programu skupi się tylko na wyświetleniu wyniku w dolnej formie roboczej – wystarczy, że użytkownik na górnej formie napisze poprawnie funkcję i naciśnie klawisz ENTER. Tak jak w poprzednim przypadku, górna forma robocza przed rozpoczęciem pracy wymaga kliknięcia w nią myszką – sytuacja ta zaprezentowana została na rysunku 23.

Wybór z menu pozycji „Wartości funkcji” daje użytkownikowi możliwość uzyskania wartości dowolnej funkcji lub wyrażenia, czyli bez zmiennej x. Po wpisaniu w górnej formie funkcji lub wyrażenia i naciśnięciu klawisza ENTER, w dolnej formie roboczej powinien pokazać się wynik, pod warunkiem, że program nie wykryje błędu. Jeśli wpisana została funkcja, po naciśnięciu klawisza ENTER powinno ukazać się okno edycyjne do wprowadzenia wartości dla zmiennej x (rysunek 24).



Rys. 24. Okienko edycyjne do wprowadzenia wartości dla zmiennej x

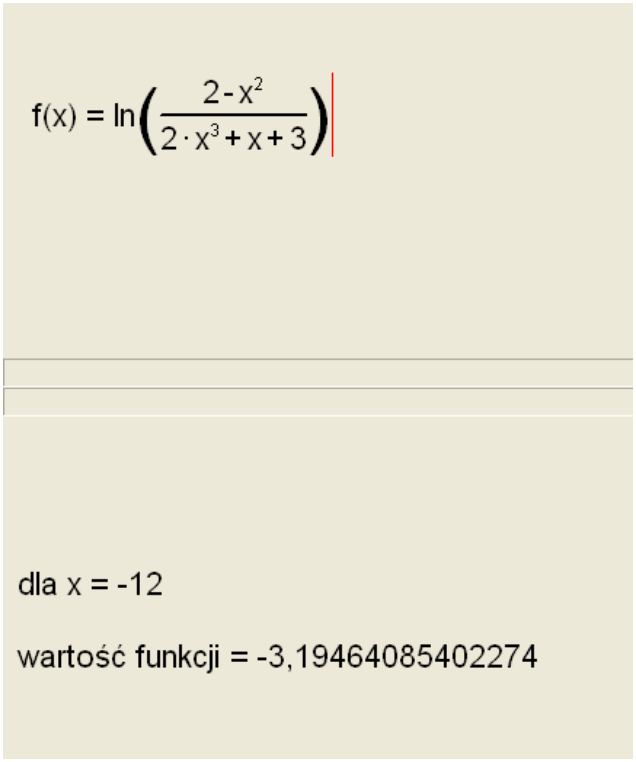
Okienko edycyjne jest typu modalnego, to znaczy, że do momentu zamknięcia przez użytkownika okienka edycyjnego za pomocą osadzonych w nim przycisków, formy robocze oraz zawarte w nich menu są nieaktywne. Podobnie działają opisane wyżej monity informujące użytkownika o błędach. Tak więc w opisywanym przypadku, okienko edycyjne oraz cały program oczekują na wprowadzenie wartości dla znaku x i naciśnięcia przycisku „`OK`.”. Można zrezygnować z wprowadzania wartości naciskając przycisk „`Cancel`” – wówczas okienko edycyjne znika a formy robocze są znowu aktywne. Przycisk „`OK`.” służy do potwierdzenia wpisanej wartości i jeśli jest ona poprawna, okienko edycyjne znika, formy robocze uzyskują swoją aktywność a w dolnej formie pojawi się wynik. Do okienka edycyjnego można wprowadzać tylko cyfry oraz separator ułamka dziesiętnego. Wartość można przedstawić także w postaci naukowej, czyli z wykorzystaniem litery E, np. 5E2 będzie oznaczało 500 w zapisie tradycyjnym. Każdy inny znak jest niedozwolony (rysunek 25).



Rys. 25. Błędnie wprowadzona wartość i możliwość jej poprawienia (błędna wartość jest zaznaczona)

Przy błędnie wprowadzonej wartości, okienko edycyjne nie pozwoli się zamknąć – koniecznie trzeba wprowadzić poprawną wartość.

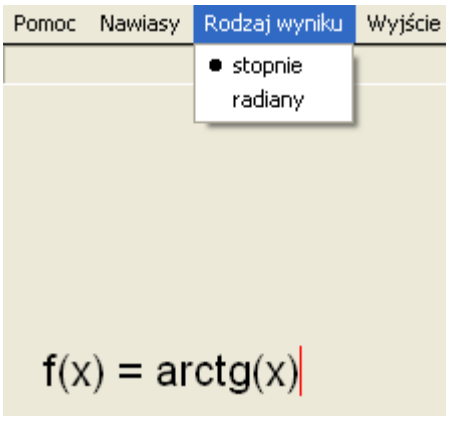
Gdy wartość jest wprowadzona poprawnie, po naciśnięciu przycisku „OK.”, dolna forma powinna wyświetlić wynik w postaci wprowadzonej wartości x oraz wartości funkcji (rysunek 26).



Rys. 26. Wynik końcowy w postaci wartości dla zmiennej x i dla pełnej funkcji

Użytkownik ma możliwość wyliczenia wartości wprowadzonej funkcji dla innej wartości zmiennej x niż tej wprowadzonej poprzednio – wystarczy, że naciśnie klawisz ENTER by ponownie wywołać okienko edycyjne i wpisać do niego nową wartość.

Program daje użytkownikowi możliwość wyboru rodzaju wyniku, to znaczy, że gdy wpisana funkcja składa się przynajmniej z jednej funkcji trygonometrycznej, wynik takiej funkcji może być wyliczony w stopniach lub radianach. Wyboru rodzaju wyniku należy dokonać z poziomu menu formy roboczej. Po dokonaniu wyboru, menu zostaje zwinięte a wynik ponownie wyliczony dla nowego rodzaju wyniku (rysunek 27).



Rys. 27. Rozwinięte menu „Rodzaj wyniku”

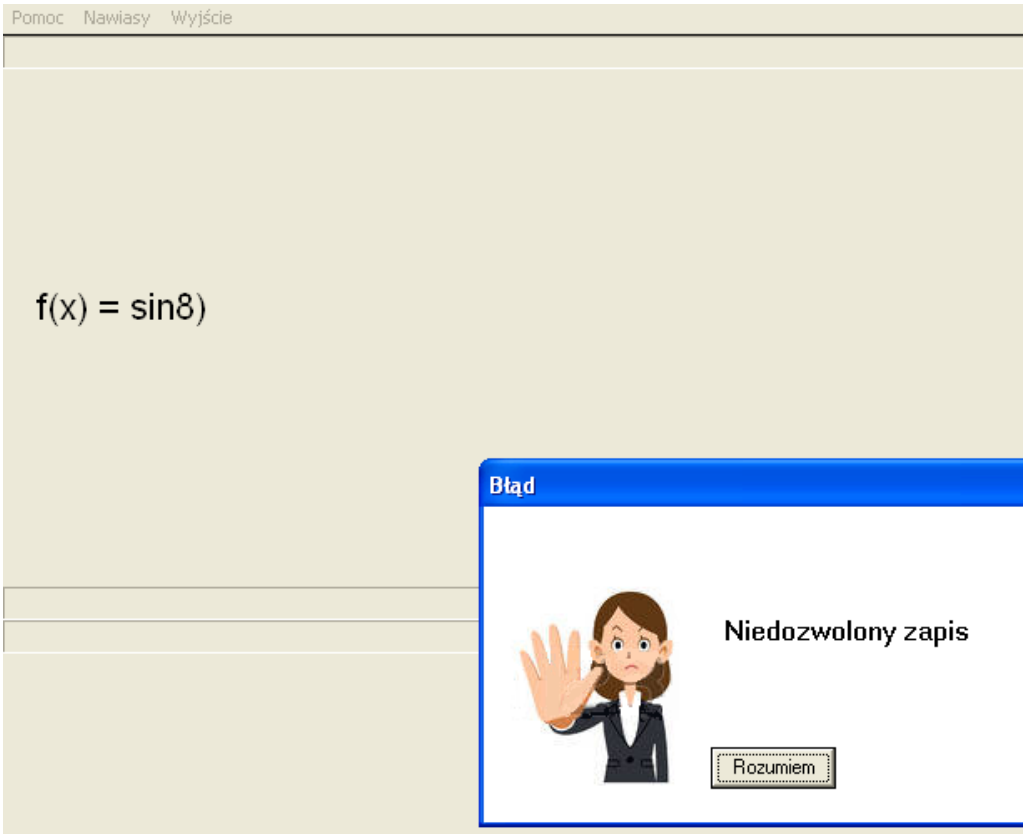
Domyślną pozycją w powyższy menu są stopnie (o czym informuje kropka przed nazwą tej pozycji). Kliknięcie myszką w pozycję „radiany” przenosi kropkę w tę pozycję i zamyka menu – od tej chwili każdy następny wynik będzie przedstawiany w radianach.

Może się zdarzyć, że choć wartość dla znaku x jest wprowadzona poprawnie, program nie będzie mógł wyliczyć wartości funkcji. Przyczyn może być kilka, np.:

- dzielenie przez zero,
- niedozwolona wartość dla wprowadzonej funkcji trygonometrycznej,
- przekroczenie dozwolonego zakresu wartości.

Dwa pierwsze przypadki są raczej oczywiste, natomiast trzeci przypadek świadczy o istnieniu swego rodzaju zabezpieczenia, nie tylko przed błędem systemowym, ale także przed błędami obliczeniowymi, które szczególnie w przypadku sprawdzania poprawnie wpisanej pochodnej przez użytkownika mają ogromne znaczenie, dlatego w programie zastosowana została kontrola przekroczenia zakresu jeszcze przed dokonaniem wyliczenia wartości. Zagadnienie to szczegółowo opisano w części dotyczącej budowy aplikacji.

Poza kontrolą przekroczenia zakresu, program posiada pełną obsługę pozostałych błędów – jeśli się pojawi, odpowiedni monit poinformuje o tym użytkownika (rysunek 28).



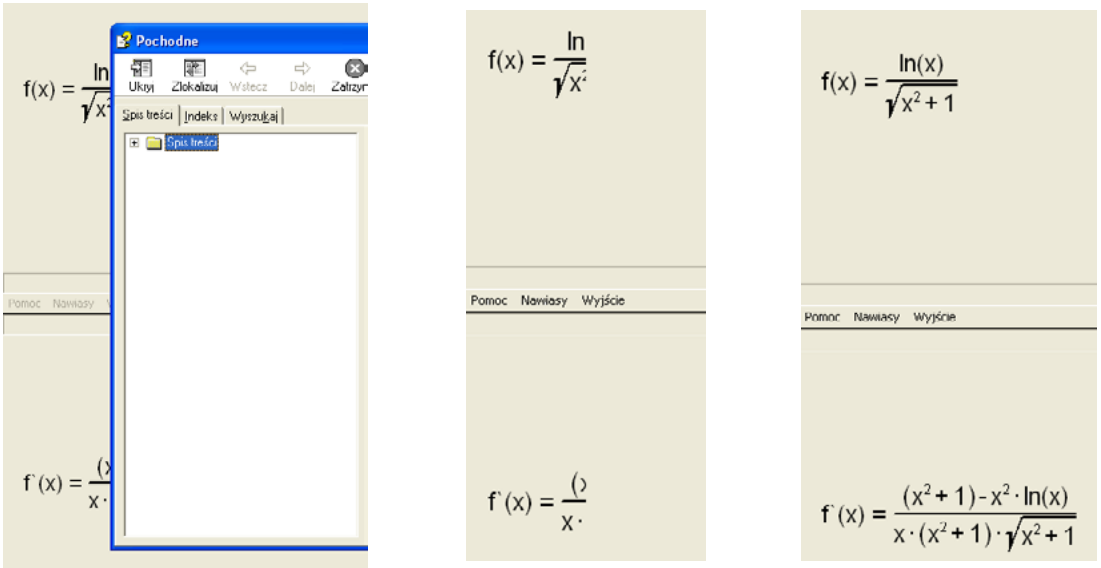
Rys. 28. Komunikat o błędzie na konkretnym przykładzie

Wybierając jedną z czterech możliwości pracy programu, forma powitalna widoczna na rysunku 1 znika, a na jej miejscu pojawiają się dwie formy robocze – jedna pod druga. W trzech przypadkach, jedna z form jest przystosowana do edycji a druga tylko do odczytu, jedynie po wyborze drugiej opcji, czyli „`Ćwiczenia - przykłady własne`”, obydwie formy przystosowane są do edycji. Formy z edycją wyposażone są w kursor, który pokazuje użytkownikowi aktualną pozycję w pisanej funkcji lub pochodnej. Można nim sterować za pomocą klawiszy strzałek kierunkowych, również w górę i dół, pod warunkiem, że funkcja

czy pochodna składa się z ułamków, pierwiastków czy funkcji potęgujących. Kursor można też wstawić poprzez kliknięcie myszką w dowolnym miejscu pisanej funkcji czy pochodnej.

Pewnego wyjaśnienia wymaga określenie tzw. znaku bieżącego. Na pewno jest to znak znajdujący się przy kursorze, ale nie zawsze wiadomo, czy jest nim znak stojący z lewej czy prawej jego strony. W niektórych sytuacjach ma to znaczenie, np. wtedy, gdy użytkownik chce przenieść kursor do licznika ułamka czy zmienić stopień pierwiastka. Czasem wskazówką jest wysokość śladu kursora, która równa jest wysokości znaku bieżącego, ale nie zawsze. W takich sytuacjach dobrze jest poznać pewną funkcjonalność edytora równań i spróbować ją wykorzystać. Polega ona na tym, że po wpisaniu nawiasu zamykającego mianownik ułamka lub funkcję podpierwiastkową, znak stojący za tymi funkcjami jest znakiem bieżącym, przy kursorze ustawionym z jego lewej strony. Użycie w tej sytuacji klawisza kierunkowego „w lewo" spowoduje zmianę znaku bieżącego na ten, znajdujący się z lewej jego strony i ustawia kursor przed tym znakiem, natomiast użycie klawisza kierunkowego „w prawo" nie zmienia znaku bieżącego, jedynie ustawia kursor za tym znakiem. Jeśli zatem, za ułamkiem lub pierwiastkiem, użytkownik nie wprowadził jeszcze znaku, program automatycznie wstawił tam znak spacji, redukując jego szerokość do zera. Gdy po napisaniu kompletnego pierwiastka, użytkownik chce ustawić kursor w polu stopnia pierwiastka, musi przenieść kursor przed pierwiastek, by to on stał się znakiem bieżącym. Dopiero wtedy, używając klawisza kierunkowego „góra", może przejść do pola stopnia pierwiastka. W przypadku ułamka sprawa znaku bieżącego wydaje się prostsza, ponieważ jeśli to kreska ułamkowa jest znakiem bieżącym, wysokość śladu kursora jest pomniejszona mniej więcej o połowę – jest to wyraźna wskazówka dla użytkownika.

Podczas pracy z programem, użytkownik może otworzyć plik pomocy lub inny obiekt spoza programu. Obiekty te mogą przysłonić formy robocze. Jeśli tak się stanie, przysłonięta nimi zawartość ulegnie zatarciu. Nie jest to błąd lecz pewna funkcjonalność graficznych metod wykorzystywanych w programie, z których korzystają edytory równań (rysunek 29).

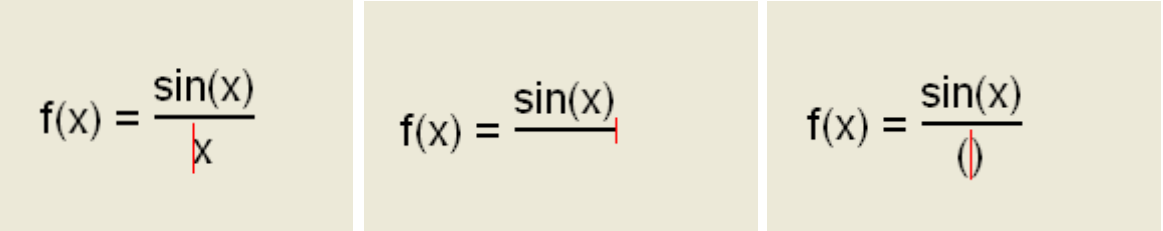


Rys. 29. Etapy zatarcia i odtworzenia zatartej zawartości form roboczych

By odtworzyć zatartą zawartość form roboczych, wystarczy nacisnąć i przytrzymać klawisz `Shift` klikając myszką w tą formę, która posiada pełne menu (na rysunku 29 jest nią forma dolna). Utrata zawartości form roboczych może także zaistnieć po uaktywnieniu się wygaszacza ekranowego. Również w tym przypadku kliknięcie w formę z pełnym menu, przy

naciśniętym klawiszu `Shift`, spowoduje odtworzenie zawartości form roboczych – górnej i dolnej.

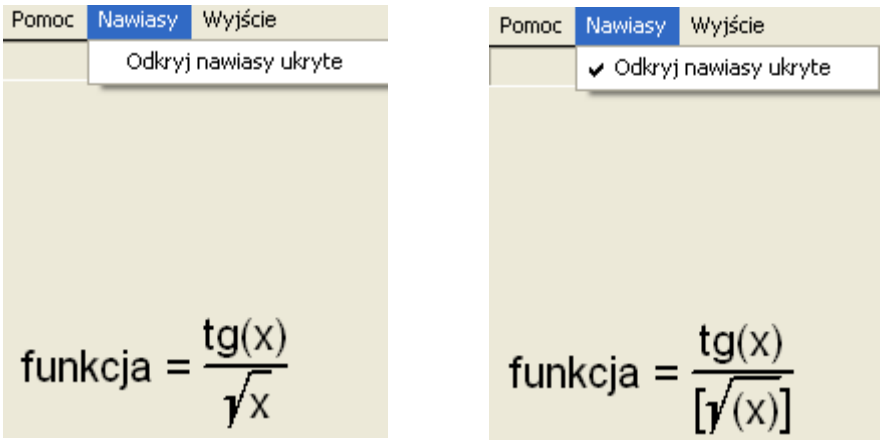
Jak już wcześniej zaznaczono, podczas pisania ułamków czy pierwiastków, skrajne nawiasy licznika, mianownika czy funkcji podpierwiastkowej zostają ukryte w celach estetycznych. Może się jednak zdarzyć, że podczas edycji pisanej funkcji lub pochodnej, pomocnym – czy wręcz koniecznym – okaże się odkrycie tych nawiasów. Przykład pokazany na rysunku 30 jest tego dowodem:



Rys. 30. Przykład funkcji, w której odkrycie nawiasów w mianowniku umożliwiło jego edycję

Na powyższym przykładzie można zaobserwować wpływ ukrycia nawiasów, gdy z mianownika zostanie usunięty jedyny, widoczny znak. Spowodowało to brak możliwości przejścia kursorem do mianownika (rysunek „B"). Dopiero odkrycie nawiasów (rysunek „C") umożliwiło ustawienie kursora miedzy nawiasami i edycję mianownika.

Odkrycie i ponowne ukrycie nawiasów możliwe jest z poziomu menu – każde kliknięcie w pole „`Odkryj nawiasy ukryte`" spowoduje naprzemienne odkrycie i ukrycie nawiasów (rysunek 31).

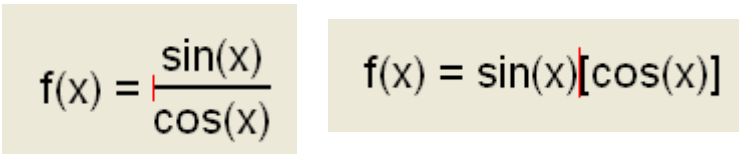


Rys. 31. Uwidocznione menu umożliwiające odkrycie i ukrycie nawiasów

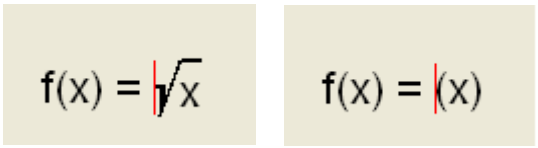
Usuwanie znaków lub ich dopisywanie przebiega w zasadzie tak samo jak w każdym innym edytorze tekstu. Wyjątkowy jest jednak sposób usuwania znaków dzielenia, pierwiastkowania, potęgowania lub logarytmu o dowolnej podstawie – usunięcie tych znaków wymaga ustawienia kursora przy tych znakach, najlepiej z lewej ich strony (w przypadku logarytmu – na jednej z liter 'log' a przy potęgowaniu – za ostatnim znakiem funkcji potęgowanej). Użycie klawisza „DELETE" spowoduje usunięcie wspomnianych znaków i wykonania operacji zależnej od usuwanego znaku, czyli:

– dla znaku dzielenia – licznik i mianownik ustawione zostają w jednej linii, jeden za drugim. Odkryte są także uprzednio ukryte, skrajne nawiasy;
– dla znaku pierwiastkowania – odkryte zostają uprzednio ukryte, skrajne nawiasy funkcji podpierwiastkowej;
– dla znaku potęgowania – funkcji potęgującej przywrócony zostaje pierwotny rozmiar znaków a ona sama ustawiona zostaje przy znaku funkcji potęgowanej;
– dla logarytmu – podstawie logarytmu przywrócony zostaje pierwotny rozmiar znaków.

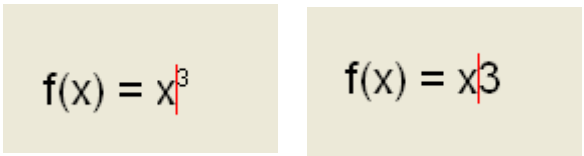
Użycie do usunięcia klawisza „BACKSPACE” wymaga ustawienia kursora za usuwanym znakiem. Trzeba jednak pamiętać, że za pomocą tego klawisza nie można usunąć znaku potęgowania (rysunki od 32 do 35).



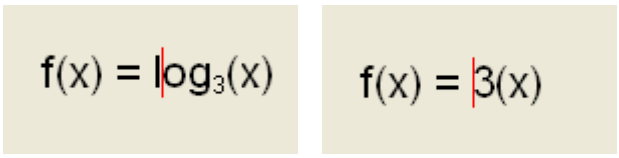
Rys. 32. Ułamek przed i po usunięciu znaku dzielenia



Rys. 33. Pierwiastek przed i po jego usunięciu



Rys. 34. Funkcja potęgująca przed i po usunięciu znaku potęgowania

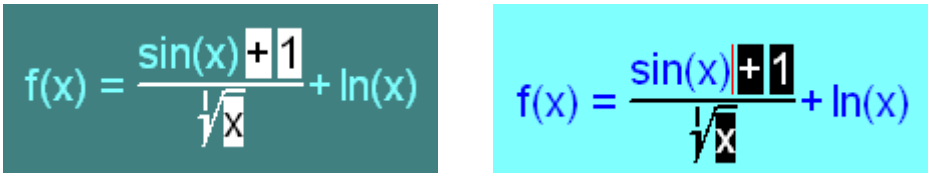


Rys. 35. Logarytm o dowolnej podstawie przed i po jego usunięciu

Usunięcie większej liczby znaków wymaga ich wstępnego zaznaczenia. Możliwe jest to zarówno przez przesuwanie myszki komputerowej po zaznaczanym tekście z wciśniętym lewym przyciskiem myszki lub za pomocą wciśnięcia jednego z klawiszy kierunkowych klawiatury, pod warunkiem, że wcześniej wciśnięty i przytrzymany został klawisz `Shift`. By zaznaczany tekst był widoczny, niezależnie od tego jaki kolor tła formy został przez użytkownika wybrany, kolor zaznaczenia uzależniony został od koloru tła formy, na której odbywa się zaznaczanie. Rozróżniane są dwie kategorie koloru formy: ciemny i jasny.

Dla ciemnych kolorów tła formy kolor obwódki wokół zaznaczanych znaków jest biały, zaś kolor zaznaczanych znaków – czarny, natomiast dla jasnych kolorów formy, kolory za-

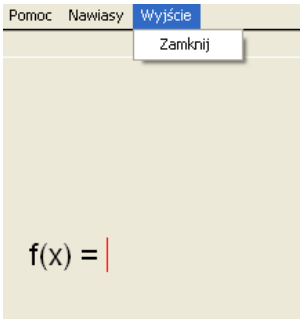
znaczania są odwrotne. Kolor zaznaczenia znaków, takich jak: pierwiastkowania czy dzielenia, jest równy takiemu, jaki wybrany został przez program do zaznaczania obwódki wokół zaznaczanych znaków (rysunek 36).



Rys. 36. Rysunki przedstawiające efekt zaznaczania tekstu na ciemnym i jasnym tle formy (w przypadku pierwiastka, widoczne jest także zaznaczenie pola stopnia pierwiastka)

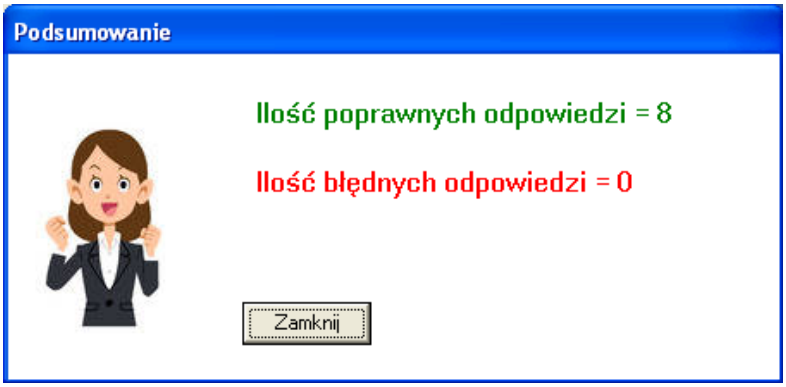
Zaznaczony tekst można usunąć za pomocą klawisza DELETE (klawisz BACKSPACE służy w tym przypadku tylko do odznaczenia zaznaczonego tekstu) lub zastąpić go dowolnym znakiem wprowadzonym z klawiatury.

Zamknięcie programu z dostępnego menu powoduje, że formy robocze zostają usunięte a na ich miejscu pojawia się forma powitalna, która umożliwia wybór innych możliwości pracy programu, m.in. otwarcia bloku ustawień lub całkowite zamknięcie programu (rysunek 37).



Rys. 37. Menu formy roboczej z uwidocznioną opcją zamknięcia

Jeśli użytkownik zdecydował się na zakończenie ćwiczeń, przed ukazaniem się formy powitalnej, wyświetlone zostanie podsumowanie (rysunek 38).



Rys. 38. Przykładowe podsumowanie po zakończeniu ćwiczeń

III. Szkielet programu

Program napisany został w języku programowania Object Pascal. Jako narzędzie programistyczne wykorzystano Delphi 4 do tworzenia aplikacji uruchamianych w środowisku Windows. Sama aplikacja jest typu SDI (*Single Document Interface*). Oznacza to, że jest reprezentowana przez okno główne aplikacji i pozostałe okna niezależne od siebie. Szkielet programu prezentuje poniższy kod:

```
program Project1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1},
  Unit2 in 'Unit2.pas' {Form2},
  Unit3 in 'Unit3.pas' {Form3},
  Unit4 in 'Unit4.pas' {Form4},
  Unit5 in 'Unit5.pas' {Form5},
  Unit6 in 'Unit6.pas' {Form6},
  Unit7 in 'Unit7.pas',
  Unit8 in 'Unit8.pas',
  Unit9 in 'Unit9.pas' {Form9};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm6, Form6);
  Application.CreateForm(Tform9, Form9);
  Application.HelpFile:='.\POMOC.CHM';
  Application.Run;
end.
```

Z powyższego kodu wynika, że tylko trzy formy są lokowane w pamięci zaraz po uruchomieniu programu, lokowanie w pamięci pozostałych form uzależnione jest od wybranej przez użytkownika opcji. Instrukcja `Application.HelpFile:='.\POMOC.CHM'` informuje aplikację o istnieniu pliku pomocy poprzez podanie zmiennej aplikacji `HelpFile` nazwę i lokalizację pliku pomocy – w tym przypadku lokalizacją pliku jest katalog bieżący, o czym informuje kropka i prawy ukośnik przed nazwą pliku pomocy.

Sam program składa się z dziewięciu modułów, które szczegółowo opisano w kolejnych rozdziałach.

IV. Moduł główny – Unit1

Po uruchomieniu programu, forma zadeklarowana w tym module prezentowana jest na ekranie jako pierwsza. Do formy dołączona została kontrolka MainMenu1 z klasy TmenuItem, za pomocą której dokonuje się wyboru ćwiczeń, otwarcia pliku pomocy bądź formy Form6 służącej do wizualnych ustawień pracy aplikacji, zgodnych z upodobaniami użytkownika.

Obrazek widoczny na module został załadowany do elementu Image1 z klasy TImage w fazie projektowania. Element Image1 posiada ustawioną właściwość stretch w pozycji true, co sprawia, że obrazek dopasuje się do rozmiarów komponentu Image1, a że komponent ten obejmuje cały obszar formy, obrazek będzie widoczny na całej formie.

Dokonując wyboru jednej z czterech możliwości rodzaju ćwiczeń, uruchamiane jest zdarzenie OnClick od wybranego polecenia menu, w wyniku którego ustawiane jest środowisko pracy aplikacji. Poniższy opis dotyczy zdarzenia OnClick od wybranej opcji „Ćwiczenia – przykłady losowane".

W pierwszej kolejności przydzielana jest pamięć dla tych form, które będą wykorzystywane w czasie ćwiczeń. Instrukcja warunkowa przed poleceniem lokowania formy w pamięci stanowi zabezpieczenie przed próbą utworzenia formy już istniejącej w pamięci.

```
if Form4=nil then Application.CreateForm(TForm4,Form4);
if Form2=nil then Application.CreateForm(TForm2,Form2);
if Form5=nil then Application.CreateForm(TForm5,Form5);
```

Uzależnienie instrukcji przydziału pamięci dla form roboczych od wybranej opcji programu, wynika z faktu, że dla każdej z tych opcji będzie funkcjonował inny wybór form oraz inne dostosowanie ich środowiska pracy, dlatego instrukcje te zamknięte zostały w osobnych procedurach, oddzielnie dla czterech możliwych opcji pracy aplikacji. Nagłówek procedury uruchamianej po wyborze pozycji „Ćwiczenia– przykłady losowane" znajduje się poniżej:

```
procedure TForm1.Cw1Click(Sender: TObject);
var rozX,rozY,pom1:integer;
begin
     . . .
end;
```

Po dokonaniu wyboru ćwiczeń, forma Form6 musi zostać ukryta, jednak wciąż istnieje w pamięci:

```
if Form6.Visible then Form6.Hide; – jeśli zmienna logiczna widoczności formy...
                                    Visible ustawiona jest w pozycji true,...
                                    ukryj formę ustawień, wywołując procedurę Hide
```

Poniższe instrukcje ustalają pozycję formy na ekranie – jest to punkt usytuowany w lewym, górnym rogu ekranu.

```
Form4.Top:=0; – ustaw współrzędną pionową formy
Form4.Left:=0; – ustaw współrzędną poziomą formy
```

Kolejne dwie instrukcje zbierają informację o aktualnej rozdzielczości ekranu – zmienne `rozX` i `rozY` służą do ustawień rozmiarów form roboczych

```
rozX:=GetSystemMetrics(SM_CXSCREEN); - pobierz rozdzielczość poziomą ekranu
rozY:=GetSystemMetrics(SM_CYSCREEN); - pobierz rozdzielczość pionową ekranu
```

Następne instrukcje służą do określenia wysokości `Form4.Height` i szerokości `Form4.Width` górnej formy. Ponieważ na dole ekranu zwykle znajduje się pasek zadań systemu Windows, wysokości form – górnej i dolnej, zostały zmniejszone tak, by wysokość form była jednakowa a dolna forma nie zachodziła za wspomniany pasek zadań. Szerokość form również została zmniejszona w celach estetycznych:

```
Form4.Height:=Trunc((rozY*0.966)/2); - zmniejszona wysokość formy w stosunku do...
                                       współrzędnej pionowej ekranu i paska...
                                       zadań Windows
Form4.Width:=Trunc(rozX/1.5); - szerokość formy równa 2/3 współrzędnej poziomej ekranu
```

Kolejne instrukcje służą do określenia obrzeża formy, paska tytułowego i dostępnych na pasku przycisków – w tym przypadku zrezygnowano z paska tytułowego i przycisków umożliwiających zamknięcie formy, zminimalizowania jej, czy też jej wyświetlenia na pełnym ekranie.

```
Form4.BorderIcons:=[]; - brak przycisków na pasku tytułowym
Form4.BorderStyle:=bsNone; - brak obrzeża i paska tytułowego (okno jest unieruchomione)
```

Kolejne polecenia ustawiają początkowe parametry formy `Form4`. Poza ustawieniem koloru formy, jak i rozmiarów poziomych marginesów usytuowanych przy górnej i dolnej jej krawędzi, wywołana zostaje funkcja logiczna `Poczatek`, której zadaniem jest zbadanie obecności pliku tekstowego zawierającego przykłady funkcji matematycznych. Gdyby funkcja zwróciła wartość `true`, świadczyłoby to o braku pliku tekstowego, dlatego w takiej sytuacji, pamięć zajmowana przez obiekty, które zostały wcześniej utworzone, musi zostać uwolniona.

```
Form4.Color:=Form6.F3; - ustaw kolor formy Form4 według zmiennej koloru...
                         znajdującej się w bloku regulacji

Form4.StaticText1.Color:=Form6.F3; - przypisz kolor formy górnemu marginesowi
Form4.StaticText1.Height:=20; - określ wysokość górnego marginesu
Form4.StaticText2.Color:=Form6.F3; - przypisz kolor formy dolnemu marginesowi
Form4.StaticText2.Height:=20; określ wysokość dolnego marginesu
pom1:=trunc(Form4.ClientHeight/2)-Form6.RZ; - wylicz współrzędną pionową dla...
                                              wyświetlanego znaku funkcji równą różnicy...
                                              połowy wysokości obszaru roboczego formy...
                                              i wysokości znaku
if Form4.Poczatek(pom1) then - czy funkcja Poczatek zwróciła wartość true?
begin - jeśli tak, to...
     Form4.Destroy; - usuń z pamięci obiekt dostępny przez wskaźnik Form4
     Form4:=nil; - zmiennej klasy przypisz identyfikator adresu pustego, czyli nil
     Form2.Destroy; } (jak wyżej, dla obiektu Form2)
     Form2:=nil;    }
     Form5.Destroy; } (jak wyżej, dla obiektu Form5)
     Form5:=nil;    }
end else
```

Jeśli funkcja `Poczatek` zwróciła wartość `false`, przygotowywanie środowiska pracy jest kontynuowane. Poniższa instrukcja służy do zainicjowania generatora liczb losowych. Konkretne liczby losowe będą wykorzystywane w dwóch przypadkach: losowanie przykładów z pliku tekstowego i wybór wartości dla zmiennej `x`.

```
Randomize(); – zainicjuj pracę generatora liczb losowych
```

Instrukcja poniżej powoduje wyświetlenie formy na ekranie, w tym przypadku górnej formy `Form4`, tworząc ją aktywną.

```
Form4.Show; – odkryj formę Form4 poprzez wywołanie procedury Show
```

Po wyświetleniu formy `Form4` wraz z pierwszym wylosowanym przykładem, pora na ustawienie dolnej formy `Form2` – jest ona umieszczona pod formą `Form4`, dlatego jej górny wierzchołek równy jest sumie górnego wierzchołka formy `Form4` i jej wysokości.

```
Form2.Top:=Form4.Top+Form4.Height; – ustaw współrzędną pionową formy Form2
Form2.Left:=Form4.Left; – ustaw współrzędną poziomą formy Form2
```

Choć szerokość tej formy jest taka sama jak formy usytuowanej nad nią, czyli `Form4`, to jej wysokość musi być pomniejszona o wysokość paska zadań Windows, który mógłby przysłonić dolną formę, dlatego wysokość ta jest różnicą pomniejszonej, pionowej rozdzielczości ekranu i pionowej współrzędnej punktu wstawienia formy na ekranie.

```
Form2.Height:=trunc(rozY*0.966)-Form2.Top; – ustaw wysokość dolnej formy Form2
Form2.Width:=Form4.Width; – ustaw szerokość dolnej formy Form2
```

Instrukcja poniżej powoduje przypisanie zmiennej globalnej `Form2.LastForm` adresu tej formy, czyli formy powitalnej `Form1`. Do przekazania adresu formy wykorzystana została zmienna `self`, która przechowuje adres tego obiektu. Dzięki takiemu rozwiązaniu będzie możliwy powrót do formy głównej, gdy forma powitalna zostanie ukryta a formy robocze, uprzednio wyświetlone, zostaną usunięte z pamięci w wyniku zakończenia ćwiczeń.

```
Form2.LastForm:=self; – zapamiętaj adres formy powitalnej w zmiennej zewnętrznego modułu
```

Poniższe instrukcje służą do wyznaczenia pionowej współrzędnej startowej, od której będzie zaczynał się pierwszy znak pochodnej wprowadzanej przez użytkownika. Wartość ta stanowi różnicę połowy wysokości obszaru roboczego formy i wysokości znaku określonego w zmiennej `Form6.RZ`. Wywołanie procedury `Form2.Poczatek(rozY)` powoduje m.in. zapamiętanie w zmiennej `AY`, należącej do modułu edukacji, wartości startowej współrzędnej pionowej dla pierwszego znaku. W zmiennej `rozY` wyliczana jest więc współrzędna pionowa, której wartość przypisana zostaje do wspomnianej zmiennej `AY` – w taki sam sposób ustalana jest współrzędna startowa dla górnej formy. Wartość ta jest równa różnicy połowy wysokości obszaru roboczego formy i wysokości znaku zapamiętanego w zmiennej `RZ` należącej do bloku ustawień.

```
rozY:=trunc(Form2.ClientHeight/2)-Form6.RZ; – wylicz współrzędną pionową...
                                              dla znaku, który będzie wstawiony...
                                              na formie
Form2.Poczatek(rozY); – zapamiętaj tą współrzędną w lokalnej zmiennej formy
```

Wstępne zabiegi przygotowujące formę Form5 do pracy sprowadzają się do ustawienia współrzędnych formy tak, by po wyświetleniu jej na ekranie była umieszczona w prawym, górnym rogu ekranu. Ustawienia formy dokonują się w procedurze Poczatek. Ponieważ forma ta ma podwójną rolę w programie, wspomnianej procedurze podana zostaje w parametrze wartość logiczna, która niesie informację do jakiej roli procedura ta ma formę przygotować – jeśli wartością tą jest true, procedura przygotuje formę do roli podpowiedzi, w innym przypadku przygotowuje ją do roli formy wynikowej usytuowanej na dole ekranu, na której będą pojawiały się pochodne znalezione przez program lub wartości funkcji.

```
Form5.Top:=Form3.Top; – ustal współrzędną pionową formy na równi ze współrzędną...
                          pionową górnej formy roboczej Form3

Form5.Left:=trunc(rozX/2.2); – przesuń współrzędną poziomą formy w stosunku do..
                    lewej krawędzi ekranu o iloraz rozdzielczości ekranu w osi X i liczbę 2,2

Form5.Width:=rozX-Form5.Left; – ustal szerokość formy równej różnicy…
                                  rozdzielczości poziomej ekranu i wcześniej ustalonej...
                                    współrzędnej poziomej formy przez co…
                                    forma rozciąga się do prawej krawędzi ekranu

Form5.Height:=Form3.Height; – ustal wysokość formy na równi z wysokością ...
                                górnej formy roboczej Form3

Form5.BorderIcons:=[]; – ustaw brak ikon na pasku tytułowym
      (wybór przycisków jest typu zbiorowego, dlatego podczas zaznaczania ich braku,...
       należy zastosować pustą parę nawiasów kwadratowych)

Form5.BorderStyle:=bsToolWindow; – wybierz zwężony pasek tytułowy
Form5.Poczatek(true); – wywołaj procedurę Poczatek ustawiającej pozostałe...
                          parametry formy podpowiedzi
```

Poniższe instrukcje powodują wyświetlenie na ekranie formy dolnej Form2 – teraz ta forma przejmuje aktywność. Następuje też ukrycie formy głównej Form1.

```
Form2.Show; – odkryj formę Form2 (procedura Show, poza ukazaniem formy na ekranie,...
          ustawia też zmienną logiczną widoczności formy Visible w pozycji true)

Form2.Wyczysc; – ustal parametry czcionki oraz wyświetl napis na formie: f'(x) =
Form1.Hide – ukryj formę powitalną (procedura Hide, poza ukryciem formy,...
           ustawia też zmienną logiczną widoczności formy Visible w pozycji false)
```

Po wyborze jednej z pozostałych trzech możliwości pracy programu, ustalanie warunków pracy form roboczych przebiega analogicznie do tych, zaprezentowanych powyżej. Różnica wynika tylko z wyboru form roboczych, czyli:

– po wyborze „Ćwiczenia – przykłady własne”, górną formą jest Form3, formą dolną jest w dalszym ciągu Form2. Formą podpowiedzi jest ponownie Form5,

```
if Form3=nil then Application.CreateForm(TForm3, Form3);
if Form2=nil then Application.CreateForm(TForm2, Form2);
if Form5=nil then Application.CreateForm(TForm5, Form5);
. . .
Form3.czy_zaczac:=false; – ustaw zmienną czy_zaczac w pozycji false,...
                             by wymusić rozpoczęcie pracy z górną formą...
                             Form3 przez jej kliknięcie myszką

Form3.LastForm:=nil; – ponieważ dolna forma Form2 posiada pełne menu, dlatego ...
                       to wskaźnik na tę formę ma już przypisany adres formy...
                       głównej, dlatego by nie dublować adresów, wskaźnikowi....
                       LastForm na formę Form3 musi być przypisany adres pusty
```

```
pom1:=trunc(Form3.ClientHeight/2)-Form6.RZ; - wylicz współrzędną...
                                              pionową dla wyświetlanego znaku funkcji równą różnicy...
                                              połowy wysokości obszaru roboczego formy...
                                              i wysokości znaku
Form3.Poczatek(pom1,0); - wywołaj procedurę Poczatek, ustawiający początkowe...
                          wartości zmiennych dla formy i trybu jej pracy
Form5.Poczatek(true); - przygotuj formę Form5 do pracy jako podpowiedź...
                        w procesie edukacji
Form3.Show; - wyświetl górną formę Form3 i uczyń ją aktywną
Form3.Canvas.TextOut(10,pom1,'Kliknij by zacząć'); - wstaw na górnej...
                                                     Form3 tekst zachęty do rozpoczęcia pracy
. . .
```

– po wyborze „pochodne”, górną formą jest `Form3`, natomiast dolną formą staje się `Form5`. W tym i następnym wyborze nie ma formy podpowiedzi,

```
if Form3=nil then Application.CreateForm(TForm3, Form3);
if Form5=nil then Application.CreateForm(TForm5, Form5);
. . .
Form3.LastForm:=self; - w tym trybie pracy, forma Form3 posiada menu...
                        umożliwiające zakończenie pracy, dlatego jej wskaźnikowi LastForm...
                        przypisz adres formy powitalnej Form1, poprzez zmienną self
Form5.Poczatek(false); - przygotuj formę Form5 jako roboczą, usytuowaną...
                         w lewym, dolnym rogu ekranu
. . .
```

– po wyborze „wartości funkcji”, układ form jest taki jak dla wyboru „pochodne”.

Otwarcie pliku pomocy z poziomu menu sprowadza się do przypisania zmiennej aplikacji nazwę procedury obsługującej ten plik oraz do uruchomienia procedury znajdującej się w bloku ustawień.

```
procedure TForm1.Op1Click(Sender: TObject);
begin
     Application.OnHelp:=Form6.HH; - przypisz zmiennej aplikacji obsługi zdarzenia...
                                     OnHelp procedurę obsługującą plik pomocy
     Form6.HH(0,0,acallhelp); - uruchom procedurę otwarcia pliku pomocy
end;
```

Po kliknięciu w przycisk zamknięcia aplikacji znajdującego się na pasku tytułowym formy lub po wyborze zamknięcia z dostępnego menu, uruchamiana jest procedura `FormClose`, w której znajdują się instrukcje usunięcia z pamięci obiektów, które zostały otworzone po uruchomieniu programu, czyli `Form6` i `Form9`. Forma powitalna wciąż istnieje w pamięci, dlatego by ją także usunąć, zostaje wywołana procedura `FormDestroy`, która usuwa obiekt z pamięci. Kolejna instrukcja przypisuje zmiennej klasy identyfikator adresu pustego.

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
     Form6.Destroy; - usuń z pamięci formę ustawień
     Form6:=nil; - przypisz wskaźnikowi Form6 adres pusty
     Form9.Destroy; - usuń z pamięci formę komunikatów
     Form9:=nil - przypisz wskaźnikowi Form9 adres pusty
end;
```

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
     Form1:=nil - przypisz zmiennej klasy Form1 adres pusty
end;
```

V. Moduł edukacji – Unit2

Ten moduł jest bardzo rozbudowany – poza pełnym edytorem równań, który będzie opisany w osobnym rozdziale, znajduje się m.in. procedura System_edukacji oraz kontrolka MainMenu1, za pomocą której można wyświetlić opis programu zawarty w pliku pomocy bądź zamknąć program.

V.1. Przygotowanie modułu do pracy

Zanim forma uzyska aktywność, wywoływana jest procedura Poczatek, która przypisuje początkowe wartości zmiennym wykorzystywanym w tym module. Z racji tego, że forma ta wyposażona została w edytor równań posiadający pełną edycją oraz dodatkową możliwość zaznaczania przez użytkownika części funkcji przeznaczonej do jednokrotnego usunięcia lub zastąpienia zaznaczonych znaków innym znakiem, musiał zostać ustalony kolor zaznaczania, odróżniający zaznaczane znaki od znaków niezaznaczonych. W tym celu, do zmiennej lokalnej kol typu TColor wprowadzona została suma wartości trzech kolorów podstawowych, z których składa się kolor tła formy. Wykorzystane tu zostały trzy funkcje znajdujące się w module Windows, które wykorzystują rzutowanie ośmiobitowych części liczby reprezentującej kolor tła formy do liczby typu Byte, oddzielnie dla każdego koloru podstawowego. Powstała suma liczb pozwala ocenić, czy kolor tła formy jest jasny czy ciemny – dla kolorów ciemnych, suma będzie mniejsza lub równa 500. Zależnie od wyniku porównania, zmiennym prywatnym kol_c (kolor zaznaczanej czcionki) i kol_t (kolor obwódki wokół zaznaczanego znaku), należącym do klasy TForm2, przypisane zostają literały reprezentujące kolor biały lub czarny. Zmienne te zostaną wykorzystane podczas wyprowadzania tekstu na formę przez procedurę Wyswietl.

Zmiennej SZER2 przypisana zostaje aktualna szerokość formy (za pomocą tej zmiennej będzie można przywrócić właściwą szerokość formie po jej poszerzeniu, gdyby pisana pochodna nie zmieściła się na formie). Zmienna AY przyjmuje wartość początkowej współrzędnej pionowej dla wprowadzanych znaków od dostarczonego jej parametru o nazwie a1. Pozostałym zmiennym przypisane zostały początkowe wartości zero. Wszystkie te zmienne zostały zadeklarowane w module jako prywatne.

```
procedure TForm2.Poczatek(const a1:integer);
var kol:TColor;
begin
     kol:=GetRValue(Form2.Color)+GetGValue(Form2.Color)
     +GetBValue(Form2.Color); – w zmiennej kol wylicz sumę wartości...
                                trzech kolorów podstawowych koloru tła formy

     if kol>500 then – czy suma kolorów jest większa od 500?
     begin jeśli tak, to kolor tła jest jasny, więc...
          kol_c:=clWhite; – przypisz zmiennej koloru zaznaczenia obwódki znaku...
                            kolor biały

          kol_t:=clBlack – przypisz zmiennej koloru zaznaczenia znaku...
                           kolor czarny

     end else – w przeciwnym razie kolor tła jest ciemny, więc...
         begin
```

```
                kol_c:=clBlack; - przypisz zmiennej koloru zaznaczenia...
                                  obwódki znaku kolor czarny
                kol_t:=clWhite - przypisz zmiennej koloru zaznaczenia znaku...
                                 kolor biały
          end;
     zaz_x:=0; - zmiennej prywatnej kierunku zaznaczania przypisz wartość zero
     zaz_odz:=false; - zmiennej prywatnej faktu zaznaczania przypisz wartość false
     wsp:=nil; } zmienne chronione edytora, należącym do klasy TForm2...
     wsb:=nil; } zainicjuj wartością pustą. Są to główne wskaźniki edytora równań
     SZER2:=Form2.ClientWidth; - zapamiętaj początkową szerokość formy
     mx2:=0; - przypisz zmiennej naddatku szerokości formy wartość zero
     AY:=a1; - przypisz zmiennej AY wartość współrzędnej pionowej,...
               wskazującej pozycję w pionie dla wprowadzanych znaków
     dob2:=0; - wyzeruj licznik poprawnych odpowiedzi
     zle2:=0 - wyzeruj licznik błędnych odpowiedzi
end;
```

Tuż po uzyskaniu przez formę aktywności, uruchamiana jest procedura `FormActive`, która uruchamia zliczanie impulsów zegarowych procesora, czego efektem będzie mruganie kursora. Ponadto uaktywniona zostaje podpowiedź w tzw. chmurce, ukazującej się przez krótką chwilę po najechaniu myszką na formę.

```
procedure TForm2.FormActivate(Sender: TObject);
begin
     Timer1.Enabled:=true; - odblokuj komponent Timer1
     sprawdzone:=false; - ustaw zmienną, ustawianą po sprawdzeniu odpowiedzi,...
                          wartość false
     Form2.ShowHint:=true - uaktywnij podpowiedź w tzw. chmurce
end;
```

Kolejną wywoływaną procedurą jest `Wyczysc`. Jest ona wywoływana wielokrotnie, po zmianie przykładu oraz po każdej poprawnej odpowiedzi. Jej zadaniem jest: uwolnienie pamięci zajętej przez wykorzystane w poprzednim przykładzie elementy edytora równań, odświeżenie parametrów czcionki, przywrócenie naturalnej szerokości formie, gdyby była ona poszerzona, oraz wyświetlenie napisu `f'(x)=` dostępnego w stałej `tekst2`.

```
procedure TForm2.Wyczysc;
begin
     wsb:=wsp; - przypisz wskaźnikowi wsb adres pierwszego elementu lisc
     while wsb<>nil do - wykonaj krok pętli, póki wskaźnik wsb nie ma adresu pustego
     begin
          wsb:=wsb^.prawa; - pozyskaj adres swego następnika
          dispose(wsp); - usuń z pamięci bieżący element lisc
          wsp:=wsb - przypisz wskaźnikowi wsp adres swego następnika
     end;
     Refresh; - wyczyść formę z treści graficznych, wstawianych metodami Canvas
     Canvas.Font.Height:=Form6.RZ; - ustal wysokość czcionki
     Canvas.Font.Name:=Form6.F1; - ustal krój czcionki
     Canvas.Font.Color:=Form6.F2D; - ustal kolor czcionki
     Canvas.Brush.Color:=Form6.F3D; - wybierz kolor wypełnienia czcionki...
                                      kolorem tła formy
     Canvas.Brush.Style:=bsSolid; - ustal styl wypełnienia czcionki pełnym kolorem
     if Form2.ClientWidth<>SZER2 then Form2.Width:=SZER2; - jeśli ...
               szerokość formy została zwiększona, przywróć jej poprzednią szerokość
```

```
      mx2:=0; - zmiennej naddatku szerokości formy przypisz wartość zero
      AX:=AXX+Canvas.TextWidth(tekst2); - przypisz zmiennej AX pozycję w osi X...
    dla pierwszego, wprowadzonego z klawiatury znaku, równą sumie wielkości początkowej 20...
    zapisanej w stałej AXX i szerokości tekstu zawartego w stałej tekst2
      Canvas.TextOut(AXX,AY,tekst2); - wyświetl napis f(x) = zawarty w stałej...
                                       tekst2
      stat_kur:=false - ustaw zmienną stat_kur w pozycji false, ustawiającą kursor...
                        z lewej strony znaku
end;
```

V. 2. Wyświetlenie pliku pomocy

Wyświetlenie pliku pomocy z dostępnego menu przedstawia poniższy fragment kodu:

```
procedure TForm2.Jakcwiczyc1Click(Sender: TObject);
begin
     Application.OnHelp:=Form6.HH; - przypisz zmiennej aplikacji obsługi
                                     zdarzenia OnHelp procedurę obsługującą plik pomocy
     Form6.HH(0,0,acallhelp); - uruchom procedurę otwarcia pliku pomocy
     Form2.Timer1.Enabled:=false - zatrzymaj mruganie kursora
end;
```

W pierwszej instrukcji procedury obsługi zdarzenia uruchamianej po wyborze kliknięciem myszką w element menu, w który wyposażone są formy robocze, przypisana zostaje aplikacji procedura obsługi zdarzenia `Form6.HH`, która jest odpowiedzialna za wyświetlenie pliku pomocy. Same wywołanie procedury `OnHelp` prezentuje następna instrukcja. Ponieważ procedura ta znajduje się w oddzielnym module, zostanie ona szczegółowo opisana w rozdziale prezentującym moduł ustawień.

W drugiej linii kodu zatrzymany zostaje kursor, by w czasie, gdy przeglądana jest pomoc, nie był zliczany czas na wpisanie pochodnej.

V. 3. Odtworzenie zamazanej zawartości form roboczych

Otwarcie pliku pomocy może powodować częściowe zasłonięcie form roboczych, czego skutkiem jest zatarcie ukrytej zawartości tych form. By odtworzyć zatartą zawartość form roboczych, konieczne jest ich przerysowanie – poniższy kod wyjaśnia ten mechanizm:

```
procedure TForm2.FormMouseDown(Sender: TObject; Button:TMouseButton;
                               Shift: TShiftState; X, Y: Integer);
var nwsp:lisc;
    x1,x2,y1,y2:integer;
begin
     if ssShift in Shift then - czy w momencie kliknięcia myszką w formę...
                                został naciśnięty klawisz klawiatury Shift?
     begin - jeśli tak, to...
          if Form4<>nil then Form4.Przerysuj - jeśli zmienna klasy Form4...
                                               nie ma adresu pustego, przerysuj zawartość formy
          else - w przeciwnym przypadku...
              if Form3<>nil then Form3.Wyswietl; jeśli zmienna klasy...
```

```
                                    Form3 nie ma adresu pustego, przerysuj jej zawartość
            Form2.Wyswietl; - przerysuj zawartość bieżącej formy
            Form2.Timer1.Enabled:=true  odblokuj komponent Timer1...
                 uruchamiając mruganie kursora i zliczanie czasu na odpowiedź
        end else . . .
```

Kliknięcie myszką w formę powoduje powstanie zdarzenia, w wyniku którego uruchomiona zostaje procedura `FormMouseDown`. Jeśli w momencie kliknięcia naciśnięty został klawisz `Shift`, uruchamiana jest ta część kodu procedury, która odpowiedzialna jest za przerysowanie zamazanej zwartości obydwu form roboczych. Odświeżenie górnej formy roboczej sprowadza się do wywołania procedury `Przerysuj` lub `Wyswietl`, zależnie od tego, czy program pracuje w trybie ćwiczeń losowanych z pliku czy dowolnych przykładów wprowadzanych z klawiatury. Dolna forma robocza odświeżana jest przy pomocy procedury `Wyswietl`. Po odtworzeniu zawartości obu form roboczych uruchamiane jest mruganie kursora w dolnej formie.

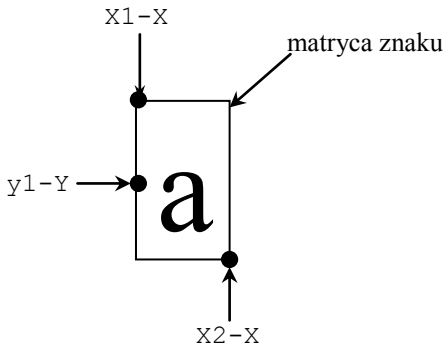
V. 4. Ustawienie kursora przez kliknięcie

Kliknięcie myszką w formę, gdy klawisz `Shift` nie został naciśnięty, uruchamia funkcję `Nowy_biez`, a w niej pętlę sprawdzającą, czy kliknięcie nastąpiło w miejscu pisanej pochodnej (procedura `MouseDown` zwraca w parametrach `X` i `Y` współrzędne kliknięcia i przekazuje je wspomnianej funkcji). Jeśli tak, to kursor ustawi się przy najbliższym znaku kliknięcia. Poniższy kod ilustruje ten mechanizm:

```
function TForm2.Nowy_biez(sX,sY:integer):lisc;;
const MK=8;
var l1,lp1:lisc;
    x1,x2,xz,y1,y2,yz:integer;
    sta_1,sta_k:boolean;
begin
     l1:=wsp; - przypisz lokalnemu wskaźnikowi l1 adres pierwszego elementu lisc
     lp1:=nil; - lokalnemu wskaźnikowi lp1, który przyjmie adres elementu kliknięcia,...
                 przypisz adres pusty

     xz:=maxint; } lokalnym zmiennym przypisz początkowe wartości, będące...
     yz:=maxint; } maksymalnymi wartościami dla typu liczbowego integer

     sta_k:=false; } lokalnym zmiennym logicznym...
     sta_1:=false; } przypisz początkowe wartości false

     while l1<>nil do - wykonaj krok pętli, póki wskaźnik l1 nie ma adresu pustego
     begin
          x1:=abs(l1^.lx-sX); - zmiennej x1 przypisz wartość bezwzględną...
                                różnicy początkowej współrzędnej lx elementu l1...
                                  i współrzędnej poziomej kliknięcia X

          x2:=abs(l1^.px-sX); - (jak wyżej, tylko w stosunku do końcowej...
                                  współrzędnej px)

          if x1>x2 then - czy kliknięcie było bliższe końcowej współrzędnej...
                          bieżącego znaku niż jego współrzędnej początkowej?

          begin - jeśli tak, to...
               x1:=x2; - przypisz zmiennej x1 mniejszą odległość w osi X ...
                         od miejsca kliknięcia

               sta_1:=true - zaznacz za pomocą zmiennej logicznej, że kursor...
```

```
                                        powinien znaleźć się z prawej strony znaku...
            end;
            x2:=round((l1^.y2-l1^.y1)/2); - w zmiennej x2 wylicz połowę...
                                            wysokości znaku
            y1:=abs((l1^.y1+x2)-sY); - wylicz w zmiennej y1 różnicę między...
                                       środkiem znaku a miejscem kliknięcia w osi Y
            if (x1<xz) or (y1<yz) then - czy jedna z najmniejszych odległości od ...
                                         miejsca kliknięcia obecnie analizowanego elementu...
                                         jest mniejsza od wybranych w poprzednich krokach pętli...
                                         najmniejszych odległości kliknięcia?
            begin - jeśli tak, to...
                  xz:=x1; } przypisz zmiennym xz i yz...
                  yz:=y1; } nowe, mniejsze odległości kliknięcia
                  if (xz<=Form6.MK) and (yz<=Form6.MK) then - czy nowe,...
                                    minimalne odległości kliknięcia od współrzędnych znaku...
                                    są mniejsze lub równe maksymalnej odległości, ustalonej...
                                    w zmiennej MK, należącej do bloku ustawień?
                  begin - jeśli tak, to...
                        lp1:=l1; - przypisz wskaźnikowi lp1 adres bieżącego
                                   elementu będącego kandydatem do elementu bieżącego
                        sta_k:=sta_1 - zmiennej logicznej sta_k przypisz...
                                       usytuowanie kursora zgodnie ze zmienną sta_1
                  end;
            end;
            sta_1:=false; - zmienną sta_1 ustaw w pozycji pierwotnej – false
            l1:=l1^.prawa - pozyskaj adres następnego elementu w wiązaniu głównym
      end;
      if lp1<>nil then stat_kur:=sta_k; - jeśli znaleziony został element...
                       kliknięcia (wskaźnik lp1 posiada adres struktury edytora)...
                       ustaw status kursora zgodnie ze zmienną sta_k
      Result:=lp1 - zwróć adres zawarty we wskaźniku lp1
end;
```

W zmiennych lokalnych `x1`, `x2`, `y1` przechowywane są wartości różnic współrzędnych znaku aktualnie sprawdzanego i współrzędnych kliknięcia `X`, `Y`. Pomocnym w zrozumieniu tych wartości jest rysunek 39:



Rys. 39. Wyjaśnienie znaczenia zmiennych przechowujących wartości współrzędnych

Zmienne `xz` i `yz` zapamiętują wartości minimalne powyższych zmiennych, które porównywane są ze zmienną `MK` przechowującą ustawioną, maksymalną wartość odległości

współrzędnych znaku od współrzędnych kliknięcia myszką (zmienna ta znajduje się w module ustawień). Porównanie z tą zmienną pozwala na przyjęcie lub nie takiego znaku jako znaku kliknięcia. Wartość w zmiennej `MK` decyduje zatem o czułości kliknięcia, dlatego za pośrednictwem modułu ustawień można ją zmieniać. Oczywiście w pętli sprawdzane są wszystkie wprowadzone znaki a wybrany zostanie ten, którego współrzędne w osi x i y są najbliższe współrzędnym kliknięcia. Po wybraniu takiego znaku, wskaźnik `lp1` zapamiętuje adres tego znaku. W tej sytuacji trzeba przemieścić znak kursora w nowe miejsce. Najpierw należy jednak wymazać ślad kursora sprzed kliknięcia – poniższy kod wyjaśnia ten mechanizm.

```
 . . .
if ssShift in Shift then – czy w momencie kliknięcia myszką w formę...
                           został naciśnięty klawisz z klawiatury Shift?
 begin
      . . .
 end else – jeśli nie, to...
     begin
          pocz_kur:=stat_kur; – zachowaj w zmiennej pocz_kur aktualne...
                                usytuowanie kursora
          nwsp:=Form2.Nowy_biez(X,Y); – wyszukaj element kliknięcia i wpisz...
                                        jego adres do zmiennej nwsp
          if nwsp<>nil then – czy znaleziony został element kliknięcia?
          begin – jeśli tak, to...
               . . .
               if pocz_kur then x1:=wsb^.px
               else x1:=wsb^.lx;
               pozyskaj współrzędną poziomą z pola lx lub px, zależnie od...
               zapamiętanej w zmiennej pocz_kur pozycji kursora
               if ord(wsb^.znak)=47 then – czy dotychczasowym znakiem...
                                           jest dzielenie?
               begin – jeśli tak, to...
                    x2:=trunc((wsb^.y1-wsb^.ly)/2); – wylicz ćwierć...
                    wysokości poziomu na którym znajduje się kreska ułamkowa
                    (wartość w polu y1 znajduje się na środku pomiędzy...
                    górną współrzędna poziomu ly a dolną py, dlatego...
                    odległość między y1 a ly to połowa wysokości poziomu)
                    y1:=wsb^.ly+x2; – wylicz górną współrzędną znaku kursora
                    y2:=wsb^.py-x2  – wylicz dolną współrzędna znaku kursora
               end else – jeśli to inny znak niż kreska ułamkowa, to...
               begin
                    y1:=wsb^.y1; } przypisz współrzędnym lokalnym...
                    y2:=wsb^.y2 } wartości z pól współrzędne y1 i y2
               end;
               Canvas.Pen.color:=Form6.F3; – określ kolor pędzla równy...
                                             kolorowi tła formy
               Canvas.Pen.width:=1; – określ grubość rysowanej linii...
                                      równej jednemu pikselowi
               Canvas.moveto(x1,y1); – określ początek rysowanej linii
               Canvas.lineto(x1,y2); – określ koniec rysowanej linii...
                                       i narysuj ją od jej początku
               wsb:=nwsp – wskaźnikowi bieżącemu przypisz adres nowego elementu
 end else . . .
```

Pierwsze instrukcje mają za zadanie pozyskanie współrzędnych `x1`, `y1` od dotychczasowego znaku bieżącego zawartego w zmiennej `wsb` i uwzględnienie tzw. statusu kursora, czyli jego umiejscowienia przed czy za znakiem. Informacja ta znajduje się w prywatnej zmiennej klasy `TForm2` – `pocz_kur`, w której zapamiętany został status kursora tuż przed wywołaniem funkcji `Nowy_biez`. Następnie poleceniami `MoveTo` i `LineTo` rysowana jest pionowa linia w miejscu kursora kolorem tła formy, co powoduje zamazanie dotychczasowego śladu kursora. Na koniec wskaźnik edytora `wsb` przyjmuje adres nowego elementu bieżącego, zawartego we wskaźniku lokalnym `nwsp`.

V. 5. Przywołanie podpowiedzi

Gdy kliknięcie nie trafi w miejsce wstawianych z klawiatury znaków pochodnej, czyli gdy wskaźnik `nwsp` będzie miał wartość `nil`, uruchamiany jest mechanizm przygotowania formy do pracy. Warunkiem wyświetlenia monitu jest istnienie przynajmniej jednego tematu podpowiedzi zawartego w zmiennej `poch1.t_nau`. Zastosowany sposób komunikowania się programu z użytkownikiem polega na wywołaniu procedury zawartej w formie komunikatów (`Form9`) i wyświetleniem formy w sposób modalny. Oznacza to, że w momencie wyświetlenia formy w ten sposób, program będzie unieruchomiony do czasu zamknięcia formy komunikatów przez użytkownika poprzez kliknięcie w jeden z przycisków na niej osadzonych. Naciśnięcie przycisku z napisem ‘`Tak`’ zwraca do programu kod zamknięcia formy równy jeden, co powoduje wywołanie procedury `Form2.System_edukacji`.

```
if nwsp<>nil then – czy, po opuszczeniu funkcji, wybrany został element kliknięcia?
begin
     . . .
end else – jeśli nie, to...
    if poch1.t_nau<>nil then czy element t_nau nie jest pusty?
    begin – jeśli nie, oznacza to, że istnieje przynajmniej jedna podpowiedź, czyli...
         Timer1.Enabled:=false; – zablokuj element Timer1
         Form9.Wynik9(23); – wywołaj procedurę komunikatu z opcją 23...
                             zachęcającą do skorzystania z pomocy
         if Form9.ShowModal=1 then Form2.System_edukacji(false)
                                    wyświetl modalnie formę komunikatów,...
                                    zatrzymując program do czasu...
                                    naciśnięcia przez użytkownika przycisku...
                                    umieszczonego na formie komunikatów.
                                    Użycie przycisku TAK powoduje...
                                   wywołanie procedury System_edukacji
         else Form2.Timer1.Enabled:=true; – dla innej, zwracanej wartości...
                                    przez formę komunikatów, uruchamiane jest...
                                    zliczanie impulsów zegarowych procesora...
                                    (uruchomienie mrugania kursora)
         zczas:=0 – wyzeruj licznik zliczający czas na odpowiedź
    end;
```

V. 6. System edukacji

Procedura `System_edukacji` wywoływana jest z parametrem `true`, gdy odpowiedź, czyli wpisana pochodna jest poprawna lub `false`, gdy wpisana pochodna jest błędna lub nie została ona wprowadzona.

V. 6.1. Wartości początkowe zmiennych edukacji

W deklaracji zmiennych edukacji zastosowano rozwiązanie polegające na zapamiętaniu ich wartości po opuszczeniu procedury. Ilustruje to poniższy fragment kodu:

```
procedure TForm2.System_edukacji(wynik:boolean);
const
      kier:boolean=false;
      koniec:boolean=false;
      powrot:boolean=false;
      wah:byte=0;
      lp2:byte=255;
      p_max:byte=0;
      p_bie:byte=0;
```

Zastosowany literał `const` sugeruje, że są to stałe, lecz obecność znaku dwukropka a za nim typu wartości oraz znaku równości i domyślnej wartość, zmienia te stałe w zmienne statyczne, a więc pozwalające się modyfikować. Ma to ogromne znaczenie w procesie edukacji, gdyż zachowanie wartości w tych zmiennych, po poprzednim opuszczeniu procedury, pozwala na kontynuowanie edukacji od miejsca wskazanego tymi właśnie zmiennymi statycznymi.

Pierwszą czynnością po otwarciu procedury jest sprawdzenie, czy forma przeznaczona na podpowiedzi jest otwarta:

```
if not Form5.Visible then – czy widoczność formy Form5 jest ustawiona w pozycji false, a więc czy forma ta jest wyświetlona?
```

Jeśli nie jest otwarta, następuje nadanie wartości początkowych zmiennym edukacji, takich jak w powyższej ich deklaracji. Przeznaczenie tych zmiennych jest następujące:

- `kier` – kierunek wyszukiwania podpowiedzi,
  - – `false` oznacza wyszukiwanie w stronę coraz prostszych funkcji elementarnych, co wiąże się z wyższymi numerami poziomów podpowiedzi;
  - – `true` oznacza kierunek przeciwny, czyli w stronę głównej funkcji przykładu, a więc coraz niższych numerów poziomów podpowiedzi;
- `koniec` – `true` oznacza wyświetlenie wszystkich podpowiedzi i zakończenie ich wyszukiwania;
- `powrot` – `true` oznacza zakończenie pierwszego etapu edukacji, w którym wyświetlane są propozycje funkcji elementarnych i wzory do zastosowania oraz rozpoczęcie drugiego etapu, w którym dodatkowo wyświetlane są pochodne funkcji elementarnych, przy stopniowym wychodzeniu z podpowiedziami w stronę głównej funkcji przykładu;
- `wah` – podstawowe fazy podpowiedzi, uzależnione od następujących wartości:
  - 0 – wyświetlenie propozycji funkcji elementarnej;
  - 1 – dodatkowo wyświetlenie wzoru do zastosowania;

- 2 – przy zmiennej `powrot` ustawionej w pozycji `true`, wyświetlenie pochodnej funkcji elementarnej.

V. 6.2. Analiza podpowiedzi należących do bieżącej funkcji

Po ustaleniu wartości początkowych następuje rozpoznanie podpowiedzi. Odbywa się to na podstawie zmiennych `poch1.t_nau`, `poch1.t_biez` i `poch1.t_pom`. W tym miejscu trzeba zaznaczyć, że powyższe zmienne podpowiedzi są jedynie wskaźnikami na elementy `nauka` zawierające pola potrzebne do przeprowadzenia procesu edukacji. Zarówno wskaźniki, jak i sama struktura `nauka` zostały zadeklarowane w bloku znajdowania pochodnych funkcji (szczegółowo zostaną omówione przy opisywaniu tego bloku). Na początkowym etapie opisywania systemu edukacji, należy wiedzieć, że podpowiedzi są grupowane w tzw. poziomy – im wyższy poziom, tym prostsza funkcja elementarna. Dzięki takiemu rozwiązaniu, proces edukacji może być dostosowany do trybu udzielania poprawnych lub niepoprawnych odpowiedzi przez użytkownika – brak odpowiedzi jest równoznaczny z odpowiedzią błędną. Przed rozpoczęciem pracy systemu edukacji, procedura musi dokonać rozeznania wszystkich podpowiedzi dotyczących bieżącego przykładu funkcji. Podpowiedzi dostępne są za pośrednictwem zmiennej klasy bloku `pochodna`, czyli `poch1` oraz wskaźników `t_nau` i `t_biez`, wskazujących na elementy `nauka`. Podczas rozeznania podpowiedzi, zbierana jest informacja o najwyższym numerze poziomu podpowiedzi – na jego podstawie można określić punk powrotu w stronę głównej funkcji, która zawsze figuruje na poziomie zero. Ponadto pola logiczne `cz_zero` są przygotowywane do nowej roli, dlatego muszą zostać zainicjowane wartością `false`. Wszystkie te czynności mają miejsce tylko wtedy, gdy forma podpowiedzi jest ukryta. Najczęściej rozeznanie podpowiedzi odbywa się w pętli `while..do`, ale może się zdarzyć, że przykład składa się tylko z jednej podpowiedzi, wówczas pętla nie jest potrzebna.

```
if not Form5.Visible then – czy forma podpowiedzi jest ukryta?
begin – jeśli tak, to...
     if not wynik then inc(zle2); – jeśli parametr wynik wskazuje na...
         błędną odpowiedź, zwiększ wartość licznika błędnych odpowiedzi o jeden
     poch1.t_pom:=poch1.t_biez; – przypisz wskaźnikowi pomocniczemu...
                                  t_pom adres ostatniej podpowiedzi
     wah:=0;
     lp2:=255;
     kier:=false;
     koniec:=false;
     powrot:=false;
     pomin:=false;
     if poch1.t_pom=poch1.t_nau then – czy pierwsza i ostatnia podpowiedź...
                                       to ten sam element?
     begin – jeśli tak, to...
          poch1.t_pom.cz_zero:=false; – ustaw pole cz_zero w pozycji...
                                        false
          p_max:=poch1.t_pom.poziom – odczytaj numer poziomu ...
                                      i zapamiętaj go w zmiennej p_max
     end else – jeśli nie, to...
         while poch1.t_pom<>nil do – wykonaj krok pętli, póki wskaźnik...
                                     t_pom nie ma adresu pustego
         begin
```

```
            if poch1.t_pom.poziom>p_max then - czy numer poziomu...
                                              aktualnego elementu jest wyższy niż numer...
                                              zapamiętany w zmiennej p_max?
            p_max:=poch1.t_pom.poziom; - jeśli tak, wpisz nowy numer...
                                         poziomu do zmiennej p_max
            poch1.t_pom.cz_zero:=false; - ustaw pole cz_zero...
                                          w pozycji false
            poch1.t_pom:=poch1.t_pom^.lewa - pozyskaj adres...
                                             poprzedniego elementu nauka
    end;
poch1.t_pom:=poch1.t_biez; - ponownie przekaż wskaźnikowi t_pom...
                             adres ostatniego elementu nauka
Form5.Show; - odkryj formę podpowiedzi
```

V. 6.3. Pierwsza podpowiedź

Sytuacja ta ma miejsce wtedy, gdy dla wylosowanego lub wpisanego przez użytkownika przykładu funkcji, użytkownik chce skorzystać z pomocy po raz pierwszy. Wywoływana jest wówczas procedura `Z_Edukacji`, która na formie podpowiedzi wyświetli podpowiedź w postaci pliku graficznego zawierającego stosowny wzór do zastosowania przez użytkownika dla pełnej funkcji. Podpowiedź taka zawsze znajduje się w ostatnim elemencie `nauka`, której adres znajduje się we wskaźniku pomocniczym `t_pom`.

```
Form5.Z_Edukacji
(poch1.t_pom.fun_A,poch1.t_pom.fun_B,poch1.t_pom.podp1,wah);
```

Choć procedura `Z_Edukacji` otrzymuje cztery parametry, istotny jest tylko trzeci, zawierający ścieżkę do pliku graficznego, czyli `poch1.t_pom.podp1`, pozostałe parametry, choć muszą być przekazane, w samej procedurze zostają zignorowane.

Po otwarciu formy podpowiedzi i wyświetleniu na niej pierwszej podpowiedzi, następuje opuszczenie procedury `System_Edukacji`.

V. 6.4. Poprawna odpowiedź

Gdy użytkownik wpisze pochodną i naciśnie klawisz ENTER w sytuacji, gdy forma podpowiedzi jest już wyświetlona, po ustawieniu zmiennej logicznej `sprawdzone` w pozycji `true` następuje porównanie wartości obu pochodnych: znalezionej przez program oraz wpisanej przez użytkownika. Jeśli weryfikacja wypadnie pozytywnie, po raz kolejny wywoływana zostaje procedura `System_edukacji` z ustawionym parametrem logicznym o nazwie `wynik` w pozycji `true`, informującym procedurę o poprawnie wpisanej pochodnej. Na podstawie wartości w zmiennych: `sprawdzone`, `wynik` oraz `koniec`, następuje podjęcie określonych czynności. Można tu wskazać trzy grupy instrukcji:

- gdy zmienna `koniec` ustawiona jest w pozycji `true` oznaczającej ostatni krok edukacji i wyświetlenie ostatniej podpowiedź dla pełnej funkcji przykładu. Następuje wówczas przywołanie formy komunikatów oraz, zależnie od reakcji użytkownika, przygotowanie form: górnej i dolnej dla następnego przykładu lub tylko zamknięcie formy komunikatów.

```
if not Form5.Visible then - czy forma podpowiedzi nie jest wyświetlona?
begin
     . . .
end else - jeśli jest wyświetlona, to...
    begin
         if sprawdzone then - czy pochodna wpisana przez użytkownika...
                              została sprawdzona?
         begin - jeśli tak, to...
              sprawdzone:=false; - ustaw zmienną sprawdzone...
                                   w pozycji false
              if wynik then - czy pochodna wpisana przez użytkownika...
                              jest poprawna?
                 if koniec then - jeśli tak, to czy osiągnięto pułap…
                                  podpowiedzi?
                 begin - jeśli tak, to...
                      Form9.Wynik9(15); wywołaj procedurę Wynik9...
                      if Form9.ShowModal=1 then - wyświetl modalnie...
                                  formę komunikatów. Czy po jej zamknięciu...
                                  zwrócona została wartość jeden?
                      begin - jeśli tak, to znaczy, że użytkownik użył...
                              przycisku TAK, więc...
                           Form2.Wyczysc; - wyczyść formę Form2 ...
                                            z treści graficznych...
                           if Form4<>nil then - czy zmienna klasy...
                                   TForm4 nie posiada adresu pustego?
                           begin - jeśli nie, to...
                                Form4.Nastepny; - wylosuj następną...
                                                  funkcję
                                Form2.Timer1.Enabled:=true;
                                uruchom zliczanie impulsów zegarowych...
                                procesora (uruchom mruganie kursora)
                                Form2.SetFocus - uczyń formę Form2...
                                                 aktywną
                           end else - jeśli Form4 nie istnieje, to...
                                if Form3<>nil then Form3.Nastepny;
                                gdy forma Form3 istnieje, wyczyść ją z treści...
                                graficznych, umożliwiając wpisanie innej funkcji
                           exit - opuść procedurę
                      end else - jeśli forma komunikatów zwróciła inną ...
                                 wartość niż jeden, czyli gdy użytkownik...
                                 użył przycisku NIE, więc...
                          begin
                               Form2.Timer1.Enabled:=true;
                               uruchom zliczanie impulsów zegarowych...
                               procesora (uruchom mruganie kursora)
                               Form2.SetFocus; - uczyń formę Form2...
                                                 aktywną
                               exit - opuść procedurę
                          end
                 end else
```

– gdy zmienna `koniec` ustawiona jest w pozycji `false`, oznaczać to może kolejny krok edukacji na bieżącym przykładzie funkcji. W tym miejscu, może okazać się, że

aktualnie wskazana użytkownikowi podpowiedź dotyczy pełnej funkcji, dlatego dodatkowo sprawdzany jest numeru poziomu podpowiedzi aktualnie wskazanej użytkownikowi – jeśli jest on zerowy, reakcja procedury jest taka sama jak w poprzednim przypadku, czyli przygotowanie form: górnej i dolnej dla następnego przykładu lub tylko zamknięcie formy komunikatów;

– jeśli zmienna `koniec` ustawiona jest w pozycji `false` a numer poziomu aktualnie wyświetlonej podpowiedzi nie jest zerowy, jest ona zaznaczona jako już wyświetlona, co uchroni przed ponownym jej wyświetleniem podczas kolejnych kroków edukacji, natomiast kierunek wyszukiwania podpowiedzi zostaje zmieniony tak, by numer poziomu kolejnej, wyszukiwanej podpowiedzi był taki jak obecnie wyświetlonej lub niższy, zbliżając się tym samym do podpowiedzi dla pełnej funkcji.

```
if koniec then – czy osiągnięto pułap podpowiedzi?
begin
      . . .
end else – jeśli nie, to...
    if poch1.t_pom.poziom=0 then – czy numer poziomu bieżącej podpowiedzi...
                                   jest równy zero?
    begin
         . . .
    end else – jeśli nie, to...
        begin
             Form9.Wynik9(16); – wywołaj procedurę Wynik9
             Form9.ShowModal; – wywołaj modalnie formę komunikatów...
             (w tym przypadku forma będzie miała aktywny tylko jeden przycisk)
             poch1.t_pom.cz_zero:=true; – ustaw pole cz_zero bieżącej...
                                          podpowiedzi w pozycji true, zaznaczając udzielenie...
                                          poprawnej odpowiedzi, sprawiając, że podpowiedź ta...
                                          nie zostanie powtórzona
             wah:=0; – przypisz zmiennej statycznej wah wartość zero,...
                       umożliwiając wyszukanie następnej podpowiedzi
             Form2.Wyczysc; – wyczyść formę Form2 z treści graficznych
             kier:=true – ustaw zmienną statyczną kier w pozycji true,...
                          zmieniając kierunek wyszukiwania podpowiedzi
        end
```

V. 6.5. Błędna odpowiedź

Gdy procedurę `System_Edukacji` wywołano z opcją `false` a zmienna `sprawdzone` ma wartość `true`, oznacza to, że wpisana pochodna została sprawdzona i jest ona błędna. Dalsze działanie procedury uzależnione jest od tego, na jakim etapie edukacji błędna odpowiedź się pojawiła – informacja ta zawarta jest w zmiennych `koniec` i `wah`. Jeśli błędnej odpowiedzi udzielono na końcu etapu edukacji, zostanie wyświetlony monit z informacją o błędnie wpisanej pochodnej, podając propozycję zmiany funkcji. W tej części można wyróżnić trzy grupy instrukcji:

– gdy zmienna `koniec` ustawiona jest w pozycji `true` a zmienna `wah` posiada wartość zero, oznacza to nie tylko ostatnią możliwą podpowiedź dla bieżącego przykładu funkcji, ale także wykorzystanie wszystkich możliwych faz podpowiedzi, czego efektem jest umieszczenie na formie podpowiedzi pełnej funkcji i jej pochodnej. W tej sytuacji, reakcją procedury jest przywołanie formy komunikatu informującego o błędnie wpisanej pochodnej i propozycja nowej funkcji;

```
if wynik then – czy pochodna wpisana przez użytkownika jest poprawna?
   . . .
else – jeśli nie, to...
if powrot then – czy kierunek przeszukiwania podpowiedzi jest ustawiony w stronę...
                 coraz niższych ich numerów poziomów?
   if koniec then – jeśli tak, to czy jest to ostatnia podpowiedź?
      if wah=0 then – jeśli tak, to czy zmienna wah wskazuje na wykorzystanie...
                      wszystkich trzech faz podpowiedzi?
      begin – jeśli tak, to...
            Form9.Wynik9(17); – wywołaj procedurę Wynik9
            if Form9.ShowModal=1 then – wyświetl modalnie formę...
            komunikatów. Czy po zamknięciu formy zwróciła ona wartość jeden?
            begin – jeśli tak, to znaczy, że użytkownik użył przycisku TAK, więc...
                  if Form4<>nil then – czy zmienna klasy TForm4...
                                       nie ma adresu pustego?
                  begin – jeśli nie, to...
                        Form4.Nastepny; – wylosuj inną funkcję
                        Form2.Timer1.Enabled:=true; uruchom...
                              zliczanie impulsów zegarowych procesora...
                              (uruchom mruganie kursora)
                        Form2.SetFocus – uczyń formę Form2 aktywną
                  end else – jeśli forma Form4 nie istnieje, to...
                       if Form3<>nil then Form3.Nastepny;
                       gdy forma Form3 istnieje, wyczyść ją z treści...
                       graficznych, umożliwiając wpisanie innej funkcji
                       Form2.Wyczysc; – wyczyść formę Form2...
                                        z treści graficznych
                       Form2.SetFocus; – uczyń formę Form2 aktywną
                       exit – opuść procedurę
            end else – gdy forma komunikatów zwróciła inną wartość niż jeden, to...
                 begin
                       Form2.Timer1.Enabled:=true; – uruchom...
                             zliczanie impulsów zegarowych procesora...
                             (uruchom mruganie kursora)
                       Form2.SetFocus; – uczyń formę Form2 aktywną
                       exit – opuść procedurę
                 end
      end else
```

– gdy zmienna `koniec` ustawiona jest w pozycji `true`, ale zmienna `wah` posiada wartość inną niż zero, oznacza to, że nie wszystkie fazy podpowiedzi zostały wykorzystane, dlatego w tym przypadku ukazana zostaje forma komunikatu, która poza informacją o błędnej pochodnej daje możliwość jej poprawienia;

```
if wah=0 then – czy zmienna wah wskazuje na wykorzystanie...
                wszystkich trzech faz podpowiedzi?
begin
     . . .
end else – zmienna wah ma inna wartość niż zero, więc...
```

```
begin
     kier:=false; - ustaw kierunek wyszukiwania podpowiedzi ...
                    w stronę coraz wyższych ich numerów poziomów
     Form9.Wynik9(18); - wywołaj procedurę Wynik9
     if Form9.ShowModal=2 then - wyświetl modalnie formę komunikatów...
                                 czy po jej zamknięciu forma zwróciła wartość dwa?
     begin - jeśli tak, to znaczy, że użytkownik użył przycisku NIE, więc...
          Form2.Timer1.Enabled:=true; - uruchom zliczanie...
             impulsów zegarowych procesora (uruchom mruganie kursora)
          Form2.SetFocus - uczyń formę Form2 aktywną
     end else - jeśli forma komunikatów zwróciła inną wartość niż dwa, to...
         begin
              Form2.Timer1.Enabled:=true; (jak wyżej)
              Form2.SetFocus; (jak wyżej)
              exit - opuść procedurę
         end
end
```

– gdy zmienna `powrot` ustawiona jest w pozycji `false`, oznacza to kolejny krok edukacji, dlatego reakcja procedury jest identyczna jak powyżej.

V. 6.6. Przywołanie pomocy bez weryfikacji wpisanej pochodnej

Sytuacja ma miejsce wtedy, gdy procedura została wywołana automatycznie, czyli po upływie określonego czasu oczekiwania programu na odpowiedź lub po kliknięciu myszką w dolną formę, przywołując pomoc na żądanie użytkownika. Można tu wyróżnić dwie możliwe sytuacje:

– gdy pomoc przywołana została podczas wyświetlenia podpowiedzi odnoszącej się do pełnej funkcji i po wykorzystaniu wszystkich możliwych faz podpowiedzi, co ma miejsce po ustawieniu zmiennej `koniec` w pozycji `true` oraz przypisaniu zmiennej liczbowej `wah` wartości zero. Następuje wówczas przywołanie formy komunikatów informującej użytkownika o braku wpisanej pochodnej, wysuwając propozycję nowej funkcji. Od reakcji użytkownika zależy, czy nowa funkcja się pojawi czy też nie;
– w każdym innym przypadku, realizowany jest kolejny krok edukacji, który zależny jest od wartości zmiennych edukacji. Jedynie zmienna `powrot` ustawiana zostaje w pozycji `false`, traktując przywołanie pomocy jak błędną odpowiedź;

```
if sprawdzone then - czy pochodna wpisana przez użytkownika została sprawdzona?
begin
     . . .
end else - jeśli nie, to...
    if powrot then - czy kierunek przeszukiwania podpowiedzi jest ustawiony...
                     w stronę coraz niższych ich numerów poziomów?
       if koniec then - jeśli tak, to czy jest to ostatnia podpowiedź?
          if wah=0 then - jeśli tak, to czy wykorzystane zostały wszystkie...
                          fazy podpowiedzi?
          begin - jeśli tak, to...
               Form9.Wynik9(19); wywołaj procedurę Wynik9
               if Form9.ShowModal=1 then - wyświetl modalnie formę...
                                           komunikatów. Czy po jej zamknięciu...
```

```
                                                    forma zwróciła wartość jeden?
                    begin - jeśli tak, to znaczy, że użytkownik użył przycisku TAK, więc...
                         Form2.Wyczysc; - wyczyść formę Form2 z treści…
                                          graficznych
                         if Form4<>nil then - czy forma Form4...
                                              istnieje w pamięci?
                         begin - jeśli tak, to...
                              Form4.Nastepny; - wylosuj inną funkcję
                              Form2.SetFocus - uczyń formę Form2 aktywną
                         end else - jeśli nie, to...
                              if Form3<>nil then Form3.Nastepny;
                              jeśli forma Form3 istnieje w pamięci,...
                              przygotuj ją na wpisanie innej funkcji
                         exit - opuść procedurę
                    end else - jeśli nie, użytkownik użył przycisku NIE, więc...
                        begin
                             Form2.Timer1.Enabled:=true; uruchom...
                                    zliczanie impulsów zegarowych procesora...
                                    (uruchom mruganie kursora)
                             Form2.SetFocus; - uczyń formę Form2 aktywną
                             exit - opuść procedurę
                        end
               end else
          else
     else kier:=false; - ustaw zmienną kier w pozycji false
```

V. 6.7. Wyszukiwanie podpowiedzi

Wyszukiwanie podpowiedzi następuje tylko wtedy, gdy zmienna `wah` posiada wartość zero. Zaraz po wykryciu tej wartości w instrukcji warunkowej, zmiennej tej przypisana zostaje wartość jeden, co sprawi, że podczas kolejnego wywołania procedury nastąpi realizacja następnej fazy podpowiedzi dotyczącej wyszukanego elementu podpowiedzi. Szukanie kolejnego elementu podpowiedzi odbywa się w pętli `while..do`. Choć podstawowym warunkiem kontynuowania pętli jest obecność adresu elementu `nauka` we wskaźniku `t_biez`, to tak naprawdę warunek ten będzie spełniony przez cały czas jej pracy, ponieważ wskaźnik ten nie jest w pętli modyfikowany. Modyfikowany jest za to wskaźnik `t_pom`, który kontrolowany jest z każdym krokiem pętli – wykrycie w instrukcji warunkowej adresu pustego w tym wskaźniku, powoduje przypisanie mu adresu pierwszego lub ostatniego elementu `nauka`, zależnie od kierunku wyszukiwania podpowiedzi, a także zmianę wyszukiwanego poziomu podpowiedzi poprzez modyfikację zmiennej `p_bie`. Po tych zmianach, pętla może być kontynuowana.

```
if wah=0 then - czy zmienna wah posiada wartość zero?
begin - jeśli tak, to...
     wah:=1; - przypisz zmiennej wah wartość jeden
     while poch1.t_biez<>nil do - rozpocznij pętlę nieskończoną
     begin
          if kier then - czy kierunek wyszukiwania podpowiedzi ustawiony jest...
                         w stronę zmniejszania się numerów ich poziomów?
             if poch1.t_pom=nil then - czy wskaźnik t_pom...
                                       uzyskał adres pusty?
               if not powrot then - jeśli tak, to czy nie wszystkie...
```

```
                                        podpowiedzi zostały zaprezentowane?
          begin - jeśli nie wszystkie, to...
               poch1.t_pom:=poch1.t_nau; - przypisz…
                                           wskaźnikowi t_pom adres…
                                           pierwszego elementu nauka

               if p_bie>0 then dec(p_bie) - jeśli wyszukiwany...
                                            poziom, w zmiennej p_bie, jest większy od zera,...
                                            zmniejsz poziom przeszukiwania

               else - w przeciwnym razie...
               begin
                    kier:=false; - zmień kierunek wyszukiwania...
                                   podpowiedzi
                    powrot:=true - ustaw zmienną powrot…
                                   w pozycji true, uruchamiając ostatnią fazę podpowiedzi
               end
          end else - gdy wszystkie podpowiedzi zostały…
                     zaprezentowane, to...
              begin
                   poch1.t_pom:=poch1.t_nau; - przypisz...
                   wskaźnikowi t_pom adres pierwszego elementu nauka

                   if p_bie>0 then dec(p_bie); - jeśli poziom...
                     wyszukiwania w zmiennej p_bie jest większy od zera,...
                     zmniejsz poziom przeszukiwania

                   if p_bie=0 then koniec:=true; - jeśli...
                                   zmienna p_bie osiągnęła wartość zero,...
                                   zaznacz kres przeszukiwań, ustawiając...
                                   zmienną koniec w pozycji true
              end
     else - jeśli wskaźnik t_pom nie jest pusty
else - jeśli kierunek wyszukiwania podpowiedzi ustawiony jest...
       w stronę zwiększania się numerów poziomów, to...

if poch1.t_pom=nil then - czy wskaźnik t_pom posiada adres pusty?
   if not powrot then - jeśli tak, to czy nie wszystkie podpowiedzi...
                        zostały zaprezentowane?

      if p_bie<p_max then - jeśli nie wszystkie, to czy numer ...
                            poziomu przeszukiwania podpowiedzi...
                            jest mniejszy od poziomu maksymalnego?

      begin - jeśli jest mniejszy od poziomu maksymalnego, to...
           inc(p_bie); - zwiększ o jeden numer...
                         zwiększ poziom przeszukiwania


           poch1.t_pom:=poch1.t_biez; - przypisz...
           wskaźnikowi t_pom adres ostatniego elementu nauka

      end else - jeśli jest równy poziomowi maksymalnemu, to...
          begin
               poch1.t_pom:=poch1.t_nau; - przypisz ...
                                           wskaźnikowi t_pom adres pierwszego...
                                           elementu nauka

               kier:=true; - zmień kierunek przeszukiwania...
                             podpowiedzi

               powrot:=true; - uaktywnij ostatnią fazę…
                               podpowiedzi
               pomin:=true - ustaw zmienną pomin w pozycji...
```

```
                                    true, w celu pominięcia dwóch pierwszych...
                                    faz podpowiedzi, prezentując na formie...
                                    podpowiedzi pochodną pełnej funkcji
                end;
        if poch1.t_pom=nil then - niezależnie od kierunku przeszukiwania,...
                                  czy wskaźnik t_pom ma przypisany adres pusty?

        begin - jeśli tak, to...
              koniec:=true; - zaznacz koniec przeszukiwań podpowiedzi
              wah:=0; - ustaw zmienną wah w pozycji początkowej
              exit - opuść procedurę
        end;
        . . .
```

Gdy wskaźnik `t_pom` nie ma wpisanego adresu pustego, możliwe jest zweryfikowanie znalezionego elementu `nauka`, porównując jego pola z ustawionymi w pętli zmiennymi edukacji. Jeśli spełnione zostaną wszystkie warunki, a więc:

- poziom podpowiedzi jest zgodny z poziomem ustawionym w zmiennej `p_bie`;
- podpowiedź nie jest powtórzeniem poprzedniej;
- możliwe wcześniejsze wyświetlenie podpowiedzi nie wpłynęło na udzielenie poprawnej odpowiedzi przez użytkownika;

w zmiennej `lp2` zostaje zapamiętany numer podpowiedzi, po czym pętla zostaje opuszczona, w przeciwnym przypadku wskaźnikowi przypisany zostanie adres następnego lub poprzedniego elementu `nauka`, zależnie od kierunku przeszukiwania podpowiedzi. By nie powtarzać dwóch pierwszych faz podpowiedzi, przygotowanej dla pełnej funkcji, zastosowana została blokada w postaci ustawienia zmiennej `pomin` w pozycji `true`. Wykrycie tej wartości w instrukcjach wyszukujących odpowiedni element `nauka` w kierunku malejących numerów poziomów i znalezieniu elementu z wpisanym poziomem zero, powoduje wyświetlenie na formie podpowiedzi pełnej funkcji oraz jej pochodnej.

```
while poch1.t_biez<>nil do
begin
     . . .
     if kier then - czy kierunek wyszukiwania podpowiedzi ustawiony jest...
                    w stronę zmniejszania się numerów ich poziomów?

        if poch1.t_pom.poziom=p_bie then - jeśli tak, to czy numer...
                                  poziomu wpisany w pole poziom, elementu...
                                   t_pom, jest równy z numerem poziomu...
                                   wyszukiwania, wpisany zmiennej p_bie?

          if poch1.t_pom.lipo<>lp2 then - jeśli tak, to czy numer...
                                  kolejny podpowiedzi, zapisany w polu lipo,...
                                  jest różny od numeru poprzedniej podpowiedzi,...
                                  zapamiętanej w zmiennej lp2?

            if not poch1.t_pom.cz_zero then - jeśli tak, to czy ...
                    na zaproponowaną użytkownikowi funkcję elementarną...
                    nie udzielił on poprawnej odpowiedzi?

            begin - jeśli warunek został spełniony, to...
                 lp2:=poch1.t_pom.lipo; - zapamiętaj w zmiennej lp2 ...
                                          numer kolejny podpowiedzi...
                                          by jej nie powtórzyć podczas...
                                          kolejnego wywołania procedury

                 pomin:=false; - ustaw zmienną pomin w pozycji false
                 break - opuść pętlę
            end else poch1.t_pom:=poch1.t_pom^.prawa
              w przeciwnym przypadku, przypisz wskaźnikowi t_pom...
```

```
                    adres następnego elementu nauka
                else - jeśli numer kolejny elementu nauka i wartość w zmiennej lp2...
                       są takie same, to...
                if pomin then - czy zmienna pomin została ustawiona
                                w pozycji true?
                begin - jeśli tak, to...
                     wah:=2; - ustaw w zmiennej wah ostatnią fazę podpowiedzi
                     Form5.Z_Edukacji
                     (poch1.t_pom.fun_A,poch1.t_pom.fun_B,'',wah);
                      wywołaj procedurę Z_Edukacji
                     wah:=0;
                     pomin:=false;
                     if p_bie=0 then koniec:=true; - zaznacz koniec...
                                   szukania podpowiedzi, gdy poziom...
                                   przeszukiwania jest równy zero
                     poch1.t_pom:=poch1.t_pom^.prawa; wskaźnikowi...
                            t_pom przypisz adres następnego elementu nauka
                     exit - opuść procedurę
                end else poch1.t_pom:=poch1.t_pom^.prawa
                gdy zmienna pomin ustawiona została w pozycji false lub...
            else poch1.t_pom:=poch1.t_pom^.prawa - gdy pole poziom...
                   bieżącego elementu nauka jest inny niż poziom wyszukiwania,...
                   wskaźnikowi t_pom przypisz adres następnego elementu nauka
        else - gdy kierunek wyszukiwania podpowiedzi ustawiony jest...
          w stronę zwiększania się numerów ich poziomów, to...
        if poch1.t_pom.poziom=p_bie then (jak wyżej)
           if poch1.t_pom.lipo<>lp2 then (jak wyżej)
              if not poch1.t_pom^.cz_zero then (jak wyżej)
              begin - jeśli spełnione zostały wszystkie warunki, to...
                   lp2:=poch1.t_pom.lipo; zapamiętaj w zmiennej lp2 ...
                                 numer kolejny bieżącego elementu nauka,...
                                 zapisany w polu lipo
                   break - opuść pętlę
              end else poch1.t_pom:=poch1.t_pom^.lewa
           else poch1.t_pom:=poch1.t_pom^.lewa
        else poch1.t_pom:=poch1.t_pom^.lewa;
       nie spełnienie choć jednego z warunków,...
       spowoduje przypisanie wskaźnikowi t_pom...
       adresu poprzedniego elementu nauka
end;
```

Po opuszczeniu pętli, wskaźnik `t_pom` powinien wskazywać na odpowiedni element `nauka`. Na jego podstawie, wyświetlona zostanie podpowiedź w fazie zerowej, czyli zaprezentowanie funkcji elementarnej, której pochodną powinien wpisać użytkownik na formie edukacji. Nim to jednak nastąpi, wyliczona zostaje wartości pochodnej znalezionej przez program znajdującej się w polu `fun_B` elementu `nauka` oraz wyliczony zostaje czas oczekiwania na odpowiedź. Wyliczenie wartości pochodnej znalezionej przez program wymaga przydzielenia pamięci dla łańcucha znakowego typu `PChar` o rozmiarze odpowiednim do umieszczenia w nim zawartości pola `fun_B` oraz wywołania funkcji `wynik`, przekazując jej w parametrze adres utworzonego łańcucha w zmiennej łańcuchowej `tabliczka`. Wynik w postaci wartości rzeczywistej zostaje zapamiętany w zmiennej `waf5`, należącej do klasy `TForm5`. Po wyliczeniu wartości, pamięć zajmowana przez łańcuch wskazywany przez

zmienną tabliczka zostaje uwolniona. Teraz już można wywołać procedurę Z_Edukacji, która wyświetli na formie podpowiedzi funkcję elementarną, zawartą w polu fun_A elementu nauka.

```
if poch1.t_pom<>nil then - czy wskaźnik t_pom nie ma adresu pustego?
begin - jeśli nie, to...
     enp2:=StrLen(poch1.t_pom.fun_B); - zlicz ilość znaków zawartych…
                                        w polu fun_B – pochodna funkcji elementarnej

     Form5.czas:=trunc((enp2*20)/sqrt(enp2))-10; - wylicz czas...
                                        oczekiwania na odpowiedź i zapamiętaj go...
                                        w zmiennej czas należącej do klasy TForm5

     GetMem(tabliczka,enp2*sizeof(PChar)); - przydziel pamięć...
     dla łańcucha typu PChar o rozmiarze odpowiednim do zliczonej liczby znaków

     StrCopy(tabliczka,poch1.t_pom.fun_B); - skopiuj zawartość pola ...
                                        Fun_B do utworzonego łańcucha

     Form5.waf5:=wart1.wynik(tabliczka,Form5.iks5); - wylicz...
                                        wartość pochodnej zawartej w utworzonym łańcuchu

     FreeMem(tabliczka,enp2*sizeof(PChar)); uwolnij pamięć...
           zajmowaną przez łańcuch identyfikowany zmienną tabliczka

     Form5.Z_Edukacji(poch1.t_pom.fun_A,'','',wah); - wywołaj...
                                        procedurę Z_Edukacji

     Form2.Timer1.Enabled:=true; - odblokuj komponent Timer1,…
                                        uruchamiając zliczanie impulsów zegarowych…
                                        procesora (uruchomienie mrugania kursora)

     Form2.SetFocus - uczyń formę Form2 aktywną
end else - gdy wskaźnik t_pom ma adres pusty, to...
    begin
         poch1.t_pom:=poch1.t_nau; - przypisz wskaźnikowi t_pom adres...
                                        pierwszego elementu nauka

         koniec:=true - zakończ przeszukiwanie podpowiedzi
    end
```

V. 6.8. Pozostałe fazy podpowiedzi

Wyświetlenie podpowiedzi w pozostałych dwóch fazach zależne jest od ustawienia zmiennej powrot oraz od niezerowych wartości w zmiennej wah. Gdy zmienna powrot posiada wartość false, wywołanie procedury Z_Edukacji ma miejsce tyko wtedy, gdy zmienna wah posiada wartość jeden. Do wyświetlenia podpowiedzi potrzebna jest funkcja elementarna i ścieżka do pliku graficznego podpowiedzi. Informacje te zawarte są w polach fun_A oraz podp1 bieżącego elementu podpowiedzi. By procedura Z_Edukacji mogła je umieścić na formie podpowiedzi, zostają jej przekazane w parametrach. Gdy zmienna powrót ma wartość true, możliwa jest jeszcze druga możliwość, o ile zmienna wah ma wartość dwa. W tej sytuacji konieczne są instrukcje warunkowe wykrywające dwie możliwe wartości w zmiennej wah i wywołanie procedury Z_Edukacji z odpowiednimi parametrami. Dla zmiennej wah równej dwa, procedurze przekazana jest funkcja elementarna i jej pochodna – w tej fazie podpowiedzi nie jest potrzebny wzór do zastosowania, więc łańcuch zwykle zawierający ścieżkę do pliku graficznego jest w tym przypadku pusty.

Po opuszczeniu procedury `Z_Edukacji`, zmienna `wah` przyjmuje wartość następną lub zerową, przygotowując fazę podpowiedzi do kolejnego wywołania procedury `System_edukacji`.

```
if wah=0 then - czy zmienna wah ma wartość zero?
begin
     . . .
end else - jeśli nie, to...
    begin
         if not powrot then - czy zmienna powrot ustawiona jest...
                              w pozycji false?
         begin - jeśli tak, to...
              Form5.Z_Edukacji(poch1.t_pom.fun_A,'',
                               poch1.t_pom.podp1,wah);
                        wyświetl podpowiedź na formie podpowiedzi,...
                        wywołując procedurę Z_Edukacji,...
                        przy wartości wah równej jeden
              wah:=0 - wpisz zmiennej wah wartość zero
         end else - jeśli zmienna powrot ustawiona jest w pozycji true, to...
             begin
                  if wah=1 then - czy zmienna wah ma wartość jeden?
                  begin
                       Form5.Z_Edukacji(poch1.t_pom.fun_A,'',
                                        poch1.t_pom.podp1,wah);
                        wyświetl podpowiedź na formie podpowiedzi (jak wyżej)

                       wah:=2 - wpisz zmiennej wah wartość dwa
                  end else - jeśli nie, zmienna wah ma wartość dwa, więc...
                      begin
                           Form5.Z_Edukacji(poch1.t_pom.fun_A,
                                            poch1.t_pom.fun_B,
                                            '',wah);
                              wyświetl podpowiedź na formie podpowiedzi...
                              przy zmiennej wah równej dwa

                           wah:=0 - przypisz zmiennej wah wartość zero
                      end;
             end;
         Form2.Timer1.Enabled:=true; - uruchom zliczanie impulsów...
                          zegarowych procesora (uruchom mruganie kursora)

         Form2.SetFocus - uczyń formę Form2 aktywną
    end;
```

V. 6.9. Wyznaczenie czasu oczekiwania na odpowiedź

Na podstawie liczby znaków, z których składa się pochodna znaleziona przez program, wyznaczony jest czas oczekiwania na odpowiedź, który użytkownik ma na udzielenie odpowiedzi, czyli na wpisanie pochodnej na formie edukacji i naciśnięcie klawisza ENTER. Poniższy wzór wyjaśnia sposób wyznaczania czasu oczekiwania:

$$\text{Form5.czas} = \left(\frac{x \cdot 20}{\sqrt{x}}\right) - 10$$ gdzie ‘x’ to liczba znaków.

Zliczanie tego czasu związane jest ze zdarzeniem `OnTimer`, w którym wykorzystany został mechanizm przerwań zegarowych procesora. Mechanizm ten wykorzystuje komponent `Timer1` z klasy `TTimer`, w którym kluczową rolę pełni tzw. interwał, czyli czas między przerwaniami – w tym przypadku ustawiony na 500 milisekund. Działanie procedury zdarzenia od komponentu `Timer1` polega, w tym przypadku, na zliczaniu w zmiennej `zczas` liczby wywołań tej procedury co 500 milisekund (zmienna `zczas` została zadeklarowana w części prywatnej opisywanego modułu). Zliczana liczba wywołań procedury jest w niej porównywana z wyliczonym czasem oczekiwania, zapamiętanym w zmiennej `czas` (zmienna ta należy do klasy `TForm5`). Samo zliczanie uzależnione jest od ustawienia właściwości `Checked` komponentu `CheckBox1` w pozycji `true` (komponent ten znajduje się w module ustawień i jest udostępniony użytkownikowi).

```
procedure TForm2.Timer1Timer(Sender: TObject);
. . .
begin
     . . .
if Form6.CheckBox1.Checked then – czy właściwość Checked komponentu ...
                                 CheckedBox1 ustawiona jest w pozycji true?
   if Form5.czas>0 then – jeśli tak, to czy został określony czas oczekiwania?
   begin – jeśli tak, to...
        inc(zczas); – zwiększ o jeden wartość w zmiennej zczas
        if zczas>=Form5.czas then – czy wartość w zmiennej zczas jest ...
           większa lub równa, określonemu w zmiennej czas, czasu oczekiwania?
           if poch1.t_nau<>nil then – jeśli tak, to czy istnieje przynajmniej...
                                      jedna podpowiedź?
           begin – jeśli tak, to...
                Timer1.Enabled:=false; – zablokuj komponent Timer1
                Form9.Wynik9(23); – wywołaj procedurę treści komunikatu...
                                    Wynik9
                if Form9.ShowModal=1 then – wyświetl modalnie formę...
                                            komunikatów. Czy po zamknięciu formy,...
                                            zwróciła ona wartość jeden?
                   Form2.System_edukacji(false); – jeśli tak, wywołaj...
                                                   procedurę System_edukacji
                zczas:=0; – wyzeruj zmienną zczas
                Timer1.Enabled:=true; – odblokuj komponent Timer1
                Form2.SetFocus – uczyń formę Form2 aktywną
           end;
   end;
. . .
end;
```

Gdy ilość zliczonych wywołań procedury jest większa lub równa ilości wyliczonego czasu oczekiwania, zostaje wyświetlona modalnie forma komunikatów. By nie doszło do konfliktu wątków, czyli zliczania impulsów zegarowych procesora i otwarcia formy komunikatów, na czas wyświetlenia formy komunikatów, komponent `Timer1` musi zostać zablokowany przez przestawienie pola `Enabled` w pozycję `false`.

V. 7. Analiza wpisanej pochodnej

Po wpisaniu pochodnej i naciśnięciu klawisza ENTER, uruchamiane jest zdarzenie `FormKeyDown`. Procedura obsługi tego zdarzenia zwraca parametr `Key`, który zawiera informację o tym, który klawisz klawiatury został naciśnięty. Można go zidentyfikować poprzez tzw. wirtualne kody – dla ENTER będzie to `Vk_Return`.

Zatem, gdy użyty został klawisz ENTER, znaki zawarte w polach `znak` elementów edytora zostają w pętli przeniesione do łańcucha znakowego, którego rozmiar jest sukcesywnie powiększany przed każdym dopisaniem znaku do łańcucha. Ponieważ łańcuch zostanie wykorzystany przez funkcję `wynik`, która akceptuje tylko nawiasy okrągłe, wszystkie inne nawiasy muszą zostać przed wpisaniem do łańcucha zamienione na okrągłe. Podobnie spacje, nieakceptowane przez funkcję `wynik`, muszą zostać odrzucone.

```
procedure TForm2.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
var ax1:lisc;
    bw2:double;
    blad2:boolean;
    wsad2,tym:PChar;
    znak:Char;
    kx:integer;
begin
     if wsb<>nil then - czy bieżący wskaźnik na element lisc nie jest pusty?
     begin - jeśli nie, to...
          . . .
          case Key of
               . . .
               VK_Return: - czy użyty został klawisz ENTER?

               if poch1.t_nau<>nil then - jeśli tak, to czy istnieje...
                                          przynajmniej jeden element nauka?

               begin - jeśli tak, to...
                    Form2.Timer1.Enabled:=false; - zablokuj...
                                                   komponent Timer1

                    wsad2:=nil; - zmiennej łańcuchowej przypisz adres pusty
                    znak:=#0; - zmiennej typu Char wpisz znak pusty
                    ax1:=wsp; - wskaźnikowi ax1 wpisz adres...
                                pierwszego elementu lisc

                    while ax1<>nil do - wykonaj krok pętli, póki wskaźnik...
                                        ax1 nie ma adresu pustego
                    begin
                         if ord(ax1^.znak) in otw then znak:='('
                         jeśli znak należy do grupy nawiasów otwierających,...
                         zmiennej znak wpisz znak nawiasu otwierającego, okrągłego

                         else - jeśli nie, to...
                              if ord(ax1^.znak) in zam then znak:=')'
                              jeśli znak należy do grupy nawiasów zamykających,...
                              zmiennej znak wpisz znak nawiasu zamykającego,...
                              okrągłego

                              else - jeśli nie, to...
                              if ax1^.znak<>' ' then znak:=ax1^.znak;
                              jeśli znak nie jest spacją, to przepisz go...
                              do zmiennej znak

                         if znak<>#0 then - czy zmienna znak ma wpisany ...
```

```
                                        inny znak niż pusty?
                    if wsad2<>nil then  jeśli tak, to czy ...
                     zmienna łańcuchowa wsad2 nie ma adresu pustego?
                    begin - jeśli tak, to dopisz kolejny znak, czyli...
                          kx:=StrLen(wsad2)+1; - zlicz...
                           dotychczasową ilość znaków skopiowaną do...
                           łańcucha i dodaj do tej ilości jeden
                          GetMem(tym,kx*sizeof(PChar));
                           przydziel pamięć dla nowego łańcucha
                          StrCopy(tym,wsad2); - skopiuj...
                                 do nowego łańcucha zawartość...
                                 poprzedniego łańcucha
                          wsad2[0]:=znak; - nowy znak wpisz...
                                        do poprzedniego łańcucha
                          wsad2[1]:=#0; zamknij poprzedni...
                                łańcuch za wstawionym znakiem
                          StrCat(tym,wsad2); do nowego...
                           łańcucha dopisz zawartość poprzedniego
                          FreeMem
                            (wsad2,(kx-1)*sizeof(PChar));
                            zwolnij pamięć zajmowaną przez...
                            poprzedni łańcuch
                          wsad2:=tym; - zmiennej łańcuchowej ...
                          wsad2 przypisz adres nowego łańcucha
                          znak:=#0 - zmiennej znak wpisz znak pusty
                    end else - gdy zmienna łańcuchowa wsad2 ma ...
                                 jeszcze adres pusty, to...
                         begin
                               GetMem(wsad2,sizeof(PChar));
                                przydziel pamięć dla łańcucha...
                                o rozmiarze odpowiednim...
                                dla jednego znaku
                               wsad2[0]:=znak; - wpisz do...
                                łańcucha znak ze zmiennej znak
                               wsad2[1]:=#0; - zamknij łańcuch...
                                           za wstawionym znakiem
                               znak:=#0 - zmiennej znak wpisz...
                                           znak pusty
                         end;
                  ax1:=ax1^.prawa - do wskaźnika ax1 wczytaj adres...
                                        następnego elementu lisc
end;
```

Gdy po opuszczeniu pętli, zmienna łańcuchowa `wsad2` nie jest zainicjowana adresem pustym, czyli gdy łańcuch, na który wskazuje zmienna, posiada przynajmniej jeden znak, następuje wyliczenie wartości pochodnej zawartej w tym łańcuchu. Odbywa się to przez wywołanie funkcji `wynik`, podając jej w parametrze zmienną łańcuchową `wsad2` oraz wartość dla znaku ‘x’, która zapamiętana została w zmiennej `iks5` należącej do klasy `TForm5` (sposób wyszukiwania wartości dla znaku ‘x’ opisano w następnym rozdziale). Do zapamiętania oceny wpisanej pochodnej służy zmienna logiczna `blad2`, która ustawiana jest w pozycji `true`, gdy po powrocie z funkcji `wynik` wystąpi błąd lub gdy funkcja `czy_nierowne`

zwróci wartość `true`, która jest wynikiem różnicy wartości obu pochodnych: znalezionej przez program i wpisanej przez użytkownika. Skoro sprawdzenie pochodnej wpisanej przez użytkownika zostało wykonane, zmienna logiczna `sprawdzone` musi zostać ustawiona w pozycji `true`, która ma wpływ na przebieg procesu edukacji. Dalsza praca programu zależy od wartości zmiennej logicznej `blad2`.

```
if wsad2=nil then - czy zmienna łańcuchowa wsad2 jest zainicjowana wartością pustą?
begin
     . . .
end else - jeśli nie, to...
    begin
         blad2:=false; - ustaw zmienną blad2 w pozycji false (bez błędu)
         bw2:=wart1.wynik(wsad2,Form5.iks5); - wylicz wartość pochodnej
         kx:=StrLen(wsad2); - z ilu znaków składa się łańcuch?
         FreeMem(wsad2,kx*sizeof(PChar)); - zwolnij pamięć zajmowaną przez...
                                            łańcuch, wskazywany przez zmienną wsad2
         if wart1.bl then blad2:=true - jeśli w wyniku liczenia wartości...
                 pochodnej wystąpił błąd, ustaw zmienną blad2 w pozycji true
         else - jeśli nie było błędu, to...
         if czy_nierowne(bw2,Form5.waf5) then blad2:=true;
                                porównaj wartości pochodnych: bw2 – wpisanej
                                przez użytkownika i waf5 – znalezionej przez program.
                                Jeśli funkcja zwróci wartość true, ustaw zmienną
                                blad2 w pozycji true
         sprawdzone:=true; - ustaw zmienną sprawdzone w pozycji true
```

Jeśli zmienna `blad2` będzie ustawiona w pozycji `false`, co oznacza, że użytkownik wpisał poprawną pochodną, dalsza reakcja programu zależy od tego, czy forma podpowiedzi jest już wyświetlona, o czym informuje zmienna widoczności formy `Visible` ustawiona w pozycji `true`, a więc:

- jeśli forma podpowiedzi jest już wyświetlona, obsługa komunikatu należy do systemu edukacji, dlatego w tej sytuacji wywoływana jest procedura `System_edukacji`;
- jeśli forma podpowiedzi nie jest wyświetlona, dalsze instrukcje związane z wyświetleniem formy komunikatu oraz przygotowaniem form roboczych dla następnego przykładu, zawarte są w instrukcjach opisywanej procedury `FormKeyDown`.

```
if not blad2 then - czy pochodna wpisana przez użytkownika była poprawna?
begin - jeśli tak, to...
     if not Form5.Visible then - czy forma podpowiedzi nie jest wyświetlona?
     begin - jeśli nie jest wyświetlona, to...
          inc(dob2); - zwiększ o jeden ilość poprawnych odpowiedzi
          Form9.Wynik9(15); - wywołaj procedurę komunikatu Wynik9
          if Form9.ShowModal=1 then - wyświetl modalnie formę komunikatu. Czy...
                                      po zamknięciu formy, zwróciła ona wartość jeden?
          begin - jeśli tak, to...
               if Form4<>nil then - czy forma Form4 istnieje w pamięci?
               begin - jeśli tak, to...
                    Form4.Nastepny; - wylosuj inny przykład funkcji
                    Form2.Timer1.Enabled:=true - uruchom zliczanie...
                                                 impulsów zegarowych procesora...
                                                 (uruchomienie mrugania kursora)
               end else - jeśli forma Form4 nie została utworzona, to...
```

```
                    if Form3<>nil then - czy istnieje forma Form3?
                    begin - jeśli tak, to...
                         Form3.Nastepny; - przygotuj formę Form3 do wpisania...
                                           następnego przykładu funkcji
                         Form3.SetFocus - uczyń formę górną Form3 aktywną
                    end;
               Form2.Wyczysc - wyczyść formę dolną (edukacji) z treści graficznych
          end else - jeżeli forma komunikatów zwróciła inną wartość niż jeden...
     end else Form2.System_edukacji(true) - jeśli forma podpowiedzi jest ...
     wyświetlona, wywołaj procedurę System_edukacji z opcją poprawnej odpowiedzi
end else
```

Jeśli zmienna `blad2` została ustawiona w pozycji `true`, reakcja programu także uzależniona jest od wyświetlenia lub nie formy podpowiedzi, jednak w tym przypadku, wywołanie procedury `System_edukacji` odbywa się zarówno wtedy, gdy forma podpowiedzi jest już wyświetlona jak również po zamknięciu formy komunikatu przyciskiem zezwalającym na skorzystanie z pomocy.

```
if not blad2 then - czy pochodna wpisana przez użytkownika była poprawna?
begin
     . . .
end else - jeśli nie, to...
    begin
         if not Form5.Visible then - czy forma podpowiedzi nie jest wyświetlona?
         begin - jeśli nie jest wyświetlona, to...
              Form9.Wynik9(21); - wywołaj procedurę komunikatu Wynik9
              if Form9.ShowModal=1 then - wyświetl modalnie formę…
                                          komunikatu czy po zamknięciu formy,…
                                          zwróciła ona wartość jeden?
                 Form2.System_edukacji(false) - jeśli tak, wywołaj procedurę...
                                                System_edukacji z opcją błędnej odpowiedzi
              else - jeśli po zamknięciu formy komunikatów,...
                     zwróciła ona inną wartość niż jeden, to...
              begin
                   Form2.Timer1.Enabled:=true; - uruchom zliczanie...
                                                 impulsów zegarowych procesora...
                                                 (uruchom mruganie kursora)
                   Form2.SetFocus - uczyń formę Form2 aktywną
              end
         end else Form2.System_edukacji(false) - jeśli forma podpowiedzi...
                                                 jest wyświetlona, wywołaj procedurę System_edukacji...
                                                 z opcją błędnej odpowiedzi
    end;
```

W przypadku naciśnięcia przez użytkownika klawisza ENTER, gdy na formie nie ma wprowadzonego znaku pochodnej lub gdy po opuszczeniu pętli `while..do` zmienna łańcuchowa `wsad2` będzie posiadała adres pusty, reakcją programu jest wyświetlenie formy komunikatu z propozycją zmiany funkcji. Gdy użytkownik wyrazi zgodę na zmianę przez naciśnięcie przycisku `TAK` osadzonego na formie komunikatu, górna forma robocza jest przygotowana na wprowadzenie innej funkcji lub zostaje wylosowana inna funkcja, zależnie od wybranej opcji ćwiczeń.

```
if wsb<>nil then - czy bieżący wskaźnik na element lisc nie jest pusty?
begin
     . . .
end else - jeśli jest zainicjowany adresem pustym, to...
    if Key=VK_Return then - czy został naciśnięty klawisz ENTER?
    begin - jeśli tak, to...
         Form2.Timer1.Enabled:=false; - zablokuj komponent Timer1
         Form9.Wynik9(22); - wywołaj procedurę komunikatu Wynik9
         if Form9.ShowModal=1 then wyświetl modalnie formę komunikatu
                        czy po jej zamknięciu, forma zwróciła wartość jeden?
         begin jeśli tak, to...
              inc(zle2); - zwiększ ilość błędnych odpowiedzi o jeden
              if Form4<>nil then - czy forma Form4 istnieje w pamięci?
              begin - jeśli tak, to...
                   Form4.nastepny; - wylosuj inną funkcję
                   Form2.Timer1.Enabled:=true; odblokuj komponent...
                        Timer1 należący do formy Form2, uruchamiając...
                        zliczanie impulsów zegarowych procesora...
                        a tym samy i mruganie kursora
              end else if Form3<>nil then Form3.Nastepny; jeśli...
                        forma Form4 nie istnieje, to gdy istnieje forma Form3...
                        przygotuj ją do wpisania innej funkcji
         end else Form2.Timer1.Enabled:=true; jeśli po zamknięciu...
                        formy komunikatów, zwróciła ona inną wartość niż jeden,...
                        to odblokuj komponent Timer1
    end;
```

W tym miejscu warto zaznaczyć, że odblokowanie komponentu `Timer1` dla formy edukacji (`Form2`) nie może mieć miejsca, gdy górną formą jest `Form3`. W takiej sytuacji to forma `Form3` jest aktywna, zatem odblokowanie komponentu `Timer1` dla formy `Form2` nastąpi dopiero po naciśnięciu przycisku ENTER w sytuacji, gdy forma `Form3` jest aktywna, co jest równoznaczne z wprowadzeniem na formie `Form3` innej funkcji.

V. 7.1. Porównanie wartości pochodnych: programu i użytkownika – `funkcja czy_nierowne`

Funkcja ta porównuje ze sobą dwie dostarczone w parametrze wartości typu `double`. By zminimalizować wpływ zaokrągleń, jakich dokonuje koprocesor arytmetyczny, porównywane wartości są sztucznie ograniczone. Polega to na podzieleniu tych liczb przez stałą o nazwie `prec`. Gdy ilorazy uzyskanych liczb, poddane obcięciu z nich części ułamkowej, są takie same, w kolejnej instrukcji warunkowej porównywane są części ułamkowe otrzymanych liczb zmienno– przecinkowych, jednak o sztucznie ograniczonej wielkości w ten sposób, że uzyskana przez funkcję `frac` część ułamkowa liczby, zostaje przeniesiona do części całkowitej o wielkość wskazaną przez stałą `prec`. Jeśli i w tym przypadku okaże się, że obydwie liczby są takie same, funkcja zwraca wartość logiczną `false`, co oznacza, że dwie liczby zmiennoprzecinkowe są takie same. Nie spełnienie chociaż jednego warunku powoduje, że funkcja zwraca wartość `true`, wskazującą na różnicę porównywanych liczb.

Ograniczenie wielkości porównywanych liczb pozwala zmniejszyć do minimum błędy zaokrągleń, które pojawiają się przy bardzo dużych oraz przy bardzo małych wartościach. Ponadto oddzielne porównywanie części całkowitej i ułamkowej, poprawia efektywność sa-

mego porównywania oraz sprawia, że w miarę skutecznie uniezależniono się od błędów zaokrągleń.

```
function czy_nierowne(const liA,liB:double):boolean;
const prec=1E6;
var v1,v2:LongInt;
begin
     Result:=true; - zwracana domyślnie wartość true
     v1:=Trunc(liA/prec); - zapamiętaj w zmiennej v1 część całkowitą ilorazu...
                            pierwszej porównywanej liczby i stałej prec
     v2:=Trunc(liB/prec); - jak wyżej, w stosunku do drugiej, porównywanej
                            liczby
     if v1=v2 then - czy obie liczby całkowite są takie same?
     begin - jeśli tak, to...
          v1:=Trunc(Frac(liA)*prec); - usuń część całkowitą z pierwszej ...
                            porównywanej liczby i przesuń przecinek o prec...
                            miejsc w prawo, przenosząc prec cyfr po przecinku...
                            do części całkowitej
          v2:=Trunc(Frac(liB)*prec); - jak wyżej, w stosunku do drugiej...
                                       porównywanej liczby
          if v1=v2 then Result:=false - jeśli obie wartości całkowite są...
           takie same, zwróć, za pośrednictwem zmiennej Result, wartość false
     end;
end;
```

V. 8. Kursor

Kursor jest nieodzownym elementem edytora. Jego obecność na formie w postaci czerwonej pionowej kreski, która pojawia się i znika z częstotliwością jednego cyklu na sekundę, pokazuje miejsce, w którym może pojawić się nowy znak lub z którego może zostać usunięty znak. Działanie kursora jest ściśle związane z mechanizmem przerwań zegarowych procesora, który wykorzystuje komponent `Timer1`. Bardzo ważną właściwością tego komponentu jest `Interval`, który wpływa na częstość wywoływania procedury obsługi zdarzenia od tego komponentu – w programie właściwość ta została ustawiona na 500 milisekund, co oznacza, że co pół sekundy procedura jest wywoływana, chyba, że właściwość `Enabled` tego komponentu została ustawiona w pozycji `false`, wówczas na czas istnienia tej wartości, wywołanie procedury zostaje odroczone.

Instrukcje związane z kursorem można podzielić na dwie części. W pierwszej części ustalane jest miejsce pojawienia się kursora. Zależne jest ono od trzech czynników:

- gdy wskaźnik `wsb` posiada adres pusty, co oznacza, że nie wprowadzono jeszcze żadnego znaku, kursor pojawi się w miejscu wskazanym przez zmienne `AX` i `AY`, które zawierają wartości współrzędnych startowych;
- gdy wskaźnik `wsb` nie jest pusty, współrzędne kursora zostają zebrane z pól współrzędnych elementu wskazywanego przez ten wskaźnik. Ustalenie współrzędnej poziomej zależne jest od statusu kursora, czyli jego usytuowania z lewej lub prawej strony znaku. Informacji tej dostarcza zmienna `stat_kur`, która podpowiada, z którego pola współrzędnej poziomej należy pobrać wartość;
- jeśli wskaźnik `wsb` wskazuje na element z wpisanym znakiem dzielenia, następuje wyliczenie ¼ wysokości poziomu, na którym znajduje się znak oraz takie ustalenie

współrzędnych pionowych, by kursor o wyliczonej wysokości, znalazł się na środku kreski ułamkowej.

W drugiej części zawarte są instrukcje rysujące kursor na formie. By uzyskać efekt mrugania kursora, instrukcje podzielone są na dwie części: w pierwszej części kursor rysowany jest w kolorze czerwonym, natomiast w drugiej – w kolorze tła formy, zamazując wcześniej narysowany kursor. O tym, jakim kolorem zostanie narysowany kursor, decyduje wartość zmiennej `tag` należącej do komponentu `Timer1`. Zmienna ta jest bitowo modyfikowana podczas każdego wywołania procedury zdarzenia, czego efektem jest naprzemienne przyjmowanie przez tą zmienną wartości jeden lub zero.

```
procedure TForm2.Timer1Timer(Sender: TObject);
var kx1,ky1,ky2,pom:integer;
begin
     if wsb<>nil then - czy wskaźnik wsb nie posiada adresu pustego?
     begin - jeśli nie, to...
          . . .
          if stat_kur then kx1:=wsb^.px else kx1:=wsb^.lx;
          jeżeli zmienna stat_kur ustawiona jest w pozycji...
          true. (kursor za znakiem), pobierz wartość...
          współrzędnej poziomej z pola px, w przeciwnym...
          przypadku, z pola lx bieżącego elementu lisc
          if ord(wsb^.znak)=47 then - czy bieżący element ma wpisany...
                                      znak dzielenia?
          begin - jeśli tak, to...
               pom:=trunc((wsb^.y1-wsb^.ly)/2); - wylicz ćwierć wysokości,...
                                  poziomu na którym znajduje się kreska ułamkowa
                                  (wartość w polu y1 znajduje się na środku pomiędzy...
                                  górną współrzędna poziomu ly a dolną py, dlatego...
                                  odległość między y1 a ly to połowa wysokości poziomu)
               ky1:=wsb^.ly+pom; - wylicz górną współrzędną pionową
               ky2:=wsb^.py-pom - wylicz dolną współrzędną pionową
          end else - jeśli znak jest inny niż dzielenie, to...
          begin
               ky1:=wsb^.y1; - pobierz górną współrzędną pionową z pola y1
               ky2:=wsb^.y2 - pobierz dolną współrzędną pionową z pola y2
          end
     end else - jeśli wskaźnik wsb jest pusty, to...
     begin
          kx1:=AX; - pobierz współrzędną poziomą ze zmiennej startowej AX
          ky1:=AY; - pobierz górną współrzędną pionową ze zmiennej startowej AY
          ky2:=AY+Form6.RZ - wylicz dolną współrzędną pionową, dodając do ...
                                  zmiennej startowej AY wysokość znaku
     end;
     canvas.Pen.width:=1; - określ grubość kreski równej jednemu pikselowi
     if tag=0 then - czy zmienna tag ma wartość zero?
     begin - jeśli tak, to...
          Canvas.Pen.color:=clRed; - określ kolor rysowanego kursora na czerwony
          Canvas.MoveTo(kx1,ky1); - określ początek rysowanej linii
          Canvas.LineTo(kx1,ky2) - określ koniec rysowanej linii i narysuj ją...
                                  od jej początku
     end else - jeśli nie, to...
     begin
          Canvas.Pen.color:=Form6.F3D; - określ kolor rysowanego kursora...
                                  na kolor tła formy
```

```
            Canvas.moveto(kx1,ky1); (jak wyżej)
            Canvas.lineto(kx1,ky2) (jak wyżej)
        end;
      . . .
      tag:=tag xor 1; - zmodyfikuj zmienną tag wykorzystując operator...
                        różnicy symetrycznej
end;
```

V. 9. Zakończenie pracy z programem

Wybierając z dostępnego menu opcję zamknięcia programu, uruchamiamy procedurę zamknięcia ćwiczeń. Gdy użytkownik w czasie ćwiczeń udzielił przynajmniej jednej odpowiedzi, wyświetlana jest forma z podsumowaniem poprawnych i błędnych odpowiedzi. Forma komunikatów (`Form9`) wyświetlana jest modalnie, tzn. że zamknięcie form roboczych nastąpi dopiero po zamknięciu formy komunikatu przez użytkownika. Kolejną czynnością podczas zamknięcia ćwiczeń jest wywołanie procedur zamknięcia pozostałych form, które wcześniej zostały utworzone (procedury te zostaną opisane w kolejnych rozdziałach). Następnie wyczyszczona zostaje pamięć zajmowana przez elementy edytora równań. Poniższy kod wyjaśnia ten mechanizm:

```
Form2.Timer1.Enabled:=false; - zablokuj komponent Timer1
if (dob2>0) or (zle2>0) then - czy użytkownik udzielił przynajmniej jednej…
                               odpowiedzi?
begin - jeśli tak, to...
      Form9.Wynik9(14,dob2,zle2); - wywołaj procedurę Wynik9 z opcją...
                                    podsumowania ćwiczeń
      Form9.ShowModal - wyświetl modalnie formę komunikatów (podsumowanie ćwiczeń)
end;
if Form4<>nil then Form4.Zamknij; - uwolnij pamięć zajmowaną przez obiekt...
                    wskazywany przez Form4 oraz przez elementy edytora równań
if Form3<>nil then Form3.Zamknij; - (jak wyżej, tylko w stosunku do Form3)
wsb:=wsp; - przypisz wskaźnikowi wsb adres pierwszego elementu edytora
while wsb<>nil do - wykonaj krok pętli, póki wskaźnik wsb nie ma adresu pustego
begin
      wsb:=wsb^.prawa; - do wskaźnika wsb wczytaj adres następnika
      dispose(wsp); - zwolnij pamięć zajmowaną przez element edytora
      wsp:=wsb - wskaźnikowi wsp przypisz adres następnika, ze wskaźnika wsb
end;
```

Wskaźnik `wsb` otrzymuje adres pierwszego elementu edytora zawarty we wskaźniku `wsp`. Kolejne instrukcje wykonywane w pętli `while..do` to pozyskanie adresu następnego elementu i zwolnienie pamięci instrukcją `dispose` elementu bieżącego. W ostatniej instrukcji pętli, adres następnego elementu – zapamiętany we wskaźniku `wsb` – zostaje przepisany do wskaźnika `wsp`. Pętla trwa dopóty, dopóki wskaźnik `wsb` nie uzyska adresu pustego – `nil`.

Kolejną instrukcją jest wyświetlenie formy powitalnej, czyli formy głównej `Form1`, której adres został zapamiętany w zmiennej `LastForm`. Gdy forma główna zostanie wyświetlona, można już zwolnić pamięć zajmowaną przez formę edukacji i przypisać wskaźnikowi wskazującemu na formę edukacji adres pusty. Poniższy fragment kodu wyjaśnia opisane czynności:

```
if LastForm<>nil then LastForm.Show; – otwórz formę powitalną, której adres...
                                      istnieje w zmiennej globalnej LastForm,...
                                      pod warunkiem, że w zmiennej tej nie ma...
                                      adresu pustego
Form2.Destroy; – usuń z pamięci obiekt wskazywany przez wskaźnik Form2
Form2:=nil – wskaźnikowi Form2 przypisz identyfikator adresu pustego
```

VI. Moduł wejściowy – Unit3

Moduł wejściowy jest podobny w swej budowie do wcześniej opisanego modułu edukacji, dlatego w tym rozdziale opisano tylko te elementy, które różnią się od modułu edukacji bądź są elementami nowymi. Pierwszą znaczącą różnicą jest usytuowanie formy w lewym, górnym rogu ekranu. Kolejną różnicą są trzy możliwe tryby pracy formy uwarunkowane wyborem dokonanym przez użytkownika. O tym, w jakim trybie forma ma pracować, nakazuje zmienna `tryb_pracy` typu `Byte`, przyjmująca wartości 0, 1 lub 2. Zmiennej tej przypisana zostaje wartość podczas przygotowywania formy do pracy w wyniku wywołanie procedury `Poczatek`, której właściwy numer trybu przekazywany jest w parametrze.

```
procedure TForm3.Poczatek(const a1:integer;a2:byte);
var kol:TColor;
begin
      . . . (ustalenie koloru zaznaczania, opisanego w rozdziale V.1)
     AY:=a1; – ustal współrzędną pionową startową na podstawie parametru a1
     tryb_pracy:=a2; – ustal tryb pracy modułu na podstawie zmiennej a2
     AX:=AXX+Canvas.TextWidth(tekst1) – wylicz współrzędną poziomą startową...
                                         uwzględniając pozycję w stałej AXX...
                                         i szerokość tekstu „f(x) = „ ...
                                         zawartego w zmiennej tekst1
end;
```

VI. 1. Analiza wpisanej funkcji

Po wpisaniu funkcji i naciśnięciu klawisza ENTER uruchamiane jest zdarzenie `Form-KeyDown`. Po zablokowaniu komponentu `Timer1`, należącego do formy `Form3`, uruchamiana jest pętla `while..do`, w której znaki typu `Char`, znajdujące się w polach `znak` elementów edytora równań, zostają przepisywane do łańcucha, którego rozmiar jest sukcesywnie powiększany tuż przed wpisaniem do niego kolejnego znaku. Sam sposób przepisywania znaków jest porównywalny z tym, opisanym w rozdziale V.7. Nowością natomiast jest reagowanie na znak ‘x’ w sytuacji, gdy zmienna `tryb_pracy` została ustawiona w pozycji 2. W tym trybie pracy, po wykryciu w pętli pierwszego wystąpienia znaku ‘x’, otwierane jest okno dialogowe służące do wprowadzenia liczby dla znaku ‘x’. Funkcja `InputQuery`, która wyświetla to okno, zwraca wartość logiczną, będącą efektem naciśnięcia jednego z dwóch przycisków osadzonych w oknie dialogowym – naciśnięcie przycisku `OK` powoduje zwrócenie przez funkcję wartości `true`, natomiast `CANCEL` lub klawisza klawiatury `ESC` – wartości `false`. Wartość ta zapamiętana zostaje w zmiennej `w_X`. Po wypełnieniu okna dialogowego i naciśnięciu przycisku `OK`, wprowadzona liczba w formacie `string` poddana zostaje konwersji do postaci liczbowej. W obawie przed wystąpieniem błędu konwersji i pojawieniem się komunikatu systemu operacyjnego, funkcja `StrToFloat` została zamknięta w instrukcji `try..except..end`. Pojawienie się błędu ustawia zmienną logiczną `w_X` w pozycji `false`. Samo wprowadzanie liczby oraz jej konwersja zamknięte zostały w pętli `repeat..until`, której opuszczenie nastąpi tylko wtedy, gdy zmienna `w_X` będzie ustawiona w pozycji `true`. Mechanizm ten zapobiega kontynuowaniu pętli, jak i samej procedury zdarzenia, gdyby wprowadzanie liczby dla znaku ‘x’ nie powiodło się, czy to z powodu błędnie

wpisanej liczby w pole edycyjne, czy też w wyniku rezygnacji z wprowadzania liczby. By okno dialogowe nie pojawiało się po wykryciu kolejnego znaku ‘x’, tuż przed wywołaniem procedury `InputQuery` sprawdzana jest wartość zmiennej `w_X` – jeśli ustawiona została w pozycji `true`, otwarcie okna dialogowego zostaje pominięte.

Dalsze instrukcje uzależnione są od wartości w zmiennej `tryb_pracy`.

```
while ax1<>nil do - wykonaj krok pętli, póki wskaźnik ax1 na element edytora równań...
                    nie ma adresu pustego
begin
  ky1:=ord(ax1^.znak); - odczytaj wartość znaku wpisanego w pole znak...
                         elementu edytora
    . . .
    if ky1<>32 then - czy odczytany znak jest różny od spacji?
    begin - jeśli tak, to...
        if tryb_pracy=2 then - czy tryb pracy modułu ustawiony jest w pozycji 2?
           if not w_X then - jeśli tak, to czy zmienna w_X ustawiona jest...
                             w pozycji false?
              if ky1=120 then - jeśli tak, to czy odczytany znak to ‘x’?
              repeat - jeśli tak, to uruchom pętlę
                    w_X:=InputQuery
                    ('Wartość dla X',
                     'Podaj wartość dla zmiennej X',
                      liczba);
                     wywołaj funkcję InputQuery, by otworzyć okno dialogowe
                    if not w_X then - czy funkcja zwróciła wartość false?
                    begin - jeśli tak, to...
                          Form3.Timer1.Enabled:=true; - odblokuj...
                            komponent Timer1 (uruchom mruganie kursora)
                          exit - opuść procedurę
                    end;
                    try - spodziewamy się błędu konwersji
                       x4:=StrToFloat(liczba); dokonaj konwersji...
                                wprowadzonej liczby w formacie string...
                                do postaci liczby zmiennoprzecinkowej
                    except - gdy wystąpi błąd konwersji, to...
                          w_X:=false - ustaw zmienną w_X w pozycji false
                    end;
              until w_X; - opuść pętlę, gdy zmienna w_X ma wartość true
        znak:=ax1^.znak - dla wartości odczytanego znaku różnego od nawiasów...
                          i spacji, wpisz do zmiennej znak znak znajdujący się w...
                          polu znak elementu edytora analizowanego w pętli
    end;
    . . .
    ax1:=ax1^.prawa - we wskaźniku ax1 pozyskaj adres następnika
end;
```

VI. 1.1. Ćwiczenia – przykłady własne

Praca modułu dla tego wyboru ćwiczeń została narzucona przez zerową wartość zmiennej `tryb_pracy`. Zaraz po opuszczeniu pętli `while..do` i dostarczeniu łańcucha znakowego z wpisaną funkcją do dalszej pracy, wywołana zostaje procedura `blok_glowny`, należąca do klasy `pochodna`, której zadaniem jest zwrócenie pochodnej otrzymanej funkcji oraz przygotowanie podpowiedzi. Jeśli po opuszczeniu procedury `blok_glowny`, jej zmienna

błędu `bl` będzie ustawiona w pozycji `true`, co oznacza błąd, wyświetlona zostaje modalnie forma komunikatów, która na podstawie zmiennej kodu błędu `blw` wyświetli stosowny komunikat informujący użytkownika o rodzaju błędu. Po zamknięciu przez użytkownika formy komunikatów, pamięć zajmowana przez wszystkie elementy podpowiedzi zostaje uwolniona. Po uruchomieniu mrugania kursora, co ma miejsce po odblokowaniu komponentu `Timer1`, procedura zostaje opuszczona.

```
if znatab=nil then - czy zmienna łańcuchowa znatab posiada adres pusty?
begin - jeśli tak, to...
     Form5.czas:=0; - wyzeruj zmienną czas oczekiwania na odpowiedź
     Form3.Timer1.Enabled:=true; - odblokuj komponent Timer1...
                                   (uruchom mruganie kursora)
     exit - opuść procedurę
end else - gdy zmienna łańcuchowa nie ma adresu pustego, to...
    if Form3.tryb_pracy=0 then - czy tryb pracy modułu wynosi zero?
    begin - jeśli tak, to...
         poch1.blok_glowny(znatab,true); wywołaj procedurę...
                     blok_glowny, z opcją utworzenia podpowiedzi
         if poch1.bl then - czy zmienna błędu bloku pochodna została...
                            ustawiona w pozycji true?
         begin - jeśli tak, to...
              Form9.Wynik9(poch1.blw); - wywołaj procedurę Wynik9...
                                  podając jej w parametrze liczbowy kod błędu
              Form9.ShowModal; - otwórz modalnie formę komunikatów
              Form3.Usun_podpowiedzi; - usuń podpowiedzi
              Form3.Timer1.Enabled:=true; - odblokuj komponent...
                                    Timer1 (uruchom mruganie kursora)
              exit - opuść procedurę
         end;
```

Bezbłędne znalezienie pochodnej daje możliwość wyznaczenia wartości dla zmiennej ‘x’ oraz wyliczenia wartości pochodnej.

VI. 1.1.1. Wyznaczenie wartości dla zmiennej ‘x’

Nim rozpocznie się szukanie wartości dla zmiennej ‘x’, na podstawie liczby znaków pochodnej znalezionej przez program wyznaczony zostaje czas oczekiwania na wpisanie pochodnej przez użytkownika. Wyznaczenie to opiera się na tym samym wzorze matematycznym, jaki został zaprezentowany w rozdziale V.6.9. Gdy czas został już wyliczony i zapamiętany w zmiennej `czas` należącej do klasy `TForm5`, rozpoczyna się losowanie wartości dla zmiennej ‘x’. Odbywa się ono dwuetapowo – najpierw losowana jest wartość z przedziału od 0,00 do 0,99, następnie do tej wartości dodana zostaje druga wylosowana wartość całkowita, z przedziału od –10 do 9, uzyskując w sumie przedział sumaryczny od –10,00 do 9,99. Instrukcje losujące zamknięte zostały w pętli `repeat..until`, której opuszczenie nastąpi wtedy, gdy funkcja `wynik` – należąca do klasy `wartosc` – również wywoływana w pętli, zwróci bezbłędną wartość. Zdarzają się pochodne, dla których tak mała wartość dla zmiennej ‘x’ nie ma rozwiązania, dlatego w co siódmym kroku pętli, do wylosowanej wartości dodana zostaje liczba 50. Może się jednak zdarzyć, choć niezwykle rzadko, że mimo wielu prób znalezienia wartości dla zmiennej ‘x’, funkcja `wynik` będzie ciągle zwracała błąd, dlatego liczba

iteracji została ograniczona do stu – przekroczenie tej liczby spowoduje bezwzględne opuszczenie pętli instrukcją `break`.

```
kx:=StrLen(znatab); – z ilu znaków składa się łańcuch znatab...
                      zawierający pochodną znalezioną przez program?

Form5.czas:=trunc((kx*20)/sqrt(kx))-10; – wylicz czas...
                      oczekiwania na wpisanie pochodnej przez użytkownika

kx:=1; – wpisz cyfrę jeden – wartość początkową ilości iteracji
(cyfra jeden nie spowoduje dodania liczby 50 do wylosowanej wartości...
w pierwszej iteracji pętli)

x4:=0.0; – wpisz wartość początkową pochodnej
repeat
      x4p:=Random(100)/100; – wylosuj liczbę z zakresu 0,00...0,99
      x4p:=x4p+Random(20)-10; – do wylosowanej liczby dodaj...
                      drugą wylosowaną liczbę z zakresu –10 do 9

      if (kx mod 7)=0.0 then x4p:=x4p+50; – gdy liczba...
                      iteracji pętli jest podzielna bez reszty przez siedem,...
                      dodaj 50 do wylosowanej liczby

      inc(kx); – zwiększ liczbę iteracji o jeden
      if kx>100 then break; – gdy liczba iteracji jest większa niż sto...
                      opuść pętlę

      x4:=wart1.wynik(znatab,x4p); – wywołaj funkcję wynik,...
                      podając jej w parametrze łańcuch zawierający pochodną...
                      i wylosowaną wartość dla znaku 'x'

until not wart1.bl; – opuść pętlę, gdy zmienna błędu bl należąca do...
                      klasy wartosc, jest ustawiona w pozycji false
```

Jeśli po opuszczeniu pętli zmienna błędu `bl` klasy `wartosc` jest ustawiona w pozycji `true`, pozostaje jedynie poinformować użytkownika o niemożności znalezienia wartości dla znaku 'x', wywołując modalnie formę komunikatów. Jeśli jednak wartość pochodnej została wyliczona bezbłędnie, obydwie wartości: pochodnej i znaku 'x' zostają zapamiętane w zmiennych globalnych należących do klasy `TForm5`.

```
if wart1.bl then – czy zmienna bl należąca do klasy wartosc...
                      została ustawiona w pozycji true?

begin – jeśli tak, to...
     Form9.Wynik9(24); – wywołaj procedurę z opcją numeru komunikatu
     Form9.ShowModal; – wyświetl modalnie formę komunikatów
end else – jeśli nie, to...
    begin
         Form5.iks5:=x4p; – zapamiętaj znalezioną wartość dla znaku 'x'...
                      w zmiennej iks5 należącej do klasy TForm5

         Form5.waf5:=x4; – zapamiętaj wartość pochodnej znalezionej...
                      przez program w zmiennej waf5...
                      należącej do klasy TForm5

         Form3.ShowHint:=false; – zablokuj pojawianie się…
                      podpowiedzi w tzw. chmurce

         Form2.SetFocus – uczyń formę Form2 aktywną
    end;
```

VI. 1.2. Pochodne bez ćwiczeń

Ten wybór pracy programu polega na wpisaniu przez użytkownika funkcji na górnej formie roboczej `Form3` oraz, po naciśnięciu klawisza ENTER, wyświetlenia pochodnej funkcji na dolnej formie roboczej.. W tej sytuacji nie ma potrzeby wyznaczania wartości dla znaku ‘x’, gdyż w tej opcji programu, funkcja `wynik` nie będzie wywoływana. Zaraz po zidentyfikowaniu w zmiennej `tryb_pracy` wartości jeden, wywołana zostaje procedura `blok_glowny` z pominięciem tworzenia podpowiedzi, gdyż w tym trybie pracy programu nie byłyby wykorzystane. Po bezbłędnym powrocie z procedury `blok_glowny`, wywołana zostaje kolejna procedura `Tylko_wynik`, która wstawia na dolnej formie roboczej `Form5` otrzymaną w parametrze pochodną funkcji znalezioną przez program. Gdyby zmienna błędu `bl` klasy `pochodna` była ustawiona w pozycji `true`, podobnie jak w poprzednim przykładzie, wywołana zostaje procedura obsługi błędów `Wynik9`, którego treść pojawi się na wyświetlonej modalnie formie komunikatów.

```
if Form3.tryb_pracy=0 then – czy zmienna tryb_pracy posiada wartość zero?
begin
     . . .
end else – jeśli nie, to...
    if Form3.tryb_pracy=1 then – czy zmienna tryb_pracy posiada wartość...
                                 jeden?

    begin – jeśli tak, to...
         poch1.blok_glowny(znatab,false); wywołaj procedurę...
         blok_glowny, podając jej w parametrach: zmienną łańcuchową znatab...
         i wartość false blokującą tworzenie podpowiedzi

         if poch1.bl then – czy zmienna błędu bl klasy pochodna...
                            ustawiona została w pozycji true?

         begin – jeśli tak, to...
              ky2:=poch1.blw; – przypisz zmiennej ky2 kod błędu
              Form9.Wynik9(ky2); – wywołaj procedurę obsługi komunikatów...
                                   podając jej w parametrze wartość kodu błędu

              Form9.ShowModal; – wyświetl modalnie formę komunikatów
              Form3.SetFocus; – uczyń formę Form3 aktywną
              Form3.Timer1.Enabled:=true; – odblokuj komponent Timer1...
                                            (uruchom mruganie kursora)
              exit – opuść procedurę
         end;
         Form5.Tylko_wynik(znatab); – wywołaj procedurę Tylko_wynik...
                                      wstawiającą na dolnej formie Form5 pochodną...
                                      zawartą w dostarczonym łańcuchu

         Form3.ShowHint:=false; – zablokuj pojawianie się podpowiedzi ...
                                  w tzw. chmurce

         Form3.Timer1.Enabled:=true; – odblokuj komponent Timer1...
                                       (uruchom mruganie kursora)

         Form3.SetFocus – uczyń formę Form3 aktywną
```

VI. 1.3. Wartości funkcji

Jeśli wybór użytkownika wskaże na zwróceniu wartości funkcji czy też wyrażenia nie posiadającego niewiadomej w postaci znaku ‘x’, praca modułu skupi się tylko na liczeniu wartości. Jeśli wprowadzony został znak ‘x’, jego wartość została już zapamiętana w zmiennej liczbowej `x4`. Zaraz po zbadaniu wartości zmiennej `tryb_pracy` i przyjęciu przez domniemanie, że skoro nie jest równa zero ani jeden więc musi wynosić dwa, wywołana zostaje funkcja `wynik` która – oprócz łańcucha znakowego zawierającego wpisaną funkcję i wartość dla zmiennej ‘x’ – otrzymuje także logiczne pole menu, którego wartość podpowiada wywoływanej funkcji sposób liczenia wartości funkcji trygonometrycznych, zgodnie z jego ustawieniem przez użytkownika. Ponieważ trzeci parametr funkcji posiada wartość domyślną, możliwe jest jego pominięcie, ale wtedy wszystkie funkcje trygonometryczne lub cyklometryczne byłyby liczone w radianach.

Jeśli po powrocie z funkcji `wynik`, zmienna błędu `bl` klasy `wartosc` zostanie ustawiona w pozycji `true`, wyświetlony zostaje komunikat o błędzie, a po jego zamknięciu, odblokowany zostaje komponent `Timer1`, po czym procedura zostaje opuszczona.

Po bezbłędnie wyliczonej wartości przez funkcję `wynik`, wyliczane zostają współrzędne względem dolnej formy `Form5`, na podstawie których będą umieszczone dwie wynikowe wartości: zmiennej ‘x’ oraz wprowadzonej funkcji. Jeśli zmienna logiczna `w_X` będzie ustawiona w pozycji `false`, wówczas informacja o wartości dla znaku ‘x’ zostaje pominięta. Przygotowane teksty do wyprowadzenia na ekran zostają dodatkowo zapamiętane w dwóch zmiennych chronionych klasy `TForm3` – `lx` i `lw`, które mogą być wykorzystane podczas odświeżenia formy `Form5` w przypadku częściowej lub całkowitej utraty ich zawartości w wyniku przysłonięcia formy innym obiektem lub po uaktywnieniu się wygaszacza ekranowego. Do wyprowadzania tekstu na formie wykorzystana została procedura `TextOut` należąca do klasy `Canvas`, której w parametrach podane zostają zarówno współrzędne wstawienia tekstu na formie, odpowiadające lewemu górnemu narożnikowi początku tekstu, jak i sam tekst.

Po wstawieniu tekstów na formie zablokowana zostaje podpowiedź w postaci tzw. chmurki, gdyż po naciśnięciu klawisza ENTER jest ona zbędna. Odblokowany zostaje komponent `Timer1`, uaktywniający mruganie kursora a sama forma `Form3` staje się aktywną.

```
if Form3.tryb_pracy=0 then – czy zmienna tryb_pracy posiada wartość zero?
begin
     . . .
end else – jeśli nie, to...
    if Form3.tryb_pracy=1 then – czy zmienna tryb_pracy posiada wartość...
                                 jeden?
    begin
         . . .
    end else jeśli nie, to ma wartość dwa, czyli...
        begin
             x4p:=wart1.wynik(znatab,x4,Form3.stopnie1.Checked);
              wywołaj funkcję wynik, podając jej w parametrach...
              adres łańcucha znakowego z wpisaną funkcją (znatab),...
              wartość dla znaku ‘x’ oraz wartość logiczną stopnie1...
              wskazującą na sposób liczenia funkcji trygonometrycznych
```

```
if wart1.bl then - czy zmienna błędu bl należąca do klasy ...
                   wartosc została ustawiona w pozycji true?
begin - jeśli tak, to...
     ky2:=wart1.blw; - przypisz zmiennej ky2 kod liczbowy...
                       błędu
     Form9.Wynik9(ky2); - wywołaj procedurę obsługi...
     komunikatów Wynik9 z podanym w parametrze kodem błędu
     Form9.ShowModal; - wyświetl modalnie formę komunikatów
     Form3.Timer1.Enabled:=true; - odblokuj komponent ...
                                   Timer1 (uruchom mruganie kursora)
     exit - opuść procedurę
end else - jeśli nie, to...
    begin
         Form5.Refresh; wyczyść formę Form5 z treści...
         graficznych, wstawianych metodami Canvas
         kx:=10; - ustaw współrzędną poziomą dla wstawianych...
                   na formie Form5 wynikowych wartości
         ky1:=Trunc(Form5.ClientHeight/3); - wylicz ...
                 współrzędną pionową dla pierwszej wynikowej...
                 wartości (wartość dla zmiennej 'x') równa...
                 współrzędnej pionowej dolnej formy powiększonej..
                 o 1/3 wysokości jej obszaru roboczego
         ky2:=ky1+2*Form6.RZ; -wylicz współrzędną pionową...
                   dla drugiej wynikowej wartości (wartość funkcji)...
                   równa pierwszej wyliczonej współrzędnej,...
                   powiększonej o podwójną wysokość znaku...
                   ustawionego w zmiennej RZ, należącej do...
                   bloku regulacji
         if w_X then - czy zmienna w_X została ustawiona...
                       w pozycji true?
         begin - jeśli tak, oznacza to ustalenie wartości dla
                 znaku 'x', więc...
              liczba:=FloatToStr(x4); - dokonaj
                                        konwersji wartości liczbowej dla
                                  zmiennej 'x' do postaci łańcucha typu string
              liczba:=Concat('dla x = ',liczba);
              dodaj przedrostek do wstawianej wartości,...
              poprzez łączenie dwóch łańcuchów typu string
              lx:=liczba; - zapamiętaj zawartość łańcucha...
                            w zmiennej chronionej modułu
              Form5.Canvas.TextOut(kx,ky1,liczba)
              wstaw na formie Form5 treść pierwszej wartości...
              w miejscu wskazanym przez zmienne kx i ky1
         end else lx:=''; - jeśli zmienna w_X nie została...
                            ustawiona w pozycji true, przypisz...
                            zmiennej chronionej lx łańcuch pusty
         liczba:=FloatToStr(x4p); - dokonaj konwersji ...
                                    wartości liczbowej funkcji, zawartej
                           w zmiennej x4p, do postaci łańcucha typu string

         liczba:=Concat
         ('wartość funkcji = ',liczba); - dodaj...
```

```
                                        przedrostek do wstawianej wartości,...
                                        poprzez łączenie dwóch łańcuchów...
                                        typu string
                        lw:=liczba; - zapamiętaj zawartość łańcucha...
                                      w drugiej zmiennej chronionej modułu
                        Form5.Canvas.TextOut(kx,ky2,liczba);
                        wstaw na formie Form5 treść drugiej wartości...
                        w miejscu wskazanym przez zmienne kx i ky2
                        Form3.ShowHint:=false; - zablokuj pojawianie się...
                                                 podpowiedzi w postaci tzw. chmurki
                        Form3.Timer1.Enabled:=true; - odblokuj...
                          komponent Timer1 (uruchom mruganie kursora)
                        Form3.SetFocus - uczyń formę Form3 aktywną
                    end;
            end;
```

VI. 2. Stopnie albo radiany

Wartość funkcji może być wyliczona zarówno w stopniach, jak i w radianach. By użytkownik mógł dokonać wyboru rodzaju wyniku, zostało odkryte dodatkowe menu na formie `Form3` z napisem „Rodzaj wyniku” (podczas pracy modułu w innych trybach niż wartości funkcji, dodatkowe menu jest ukryte). Po jego rozwinięciu, zostają uwidocznione dwie możliwości wyboru: „stopnie” oraz „radiany”. Obydwie pozycje wyboru w powyższym menu mają ustawione właściwości `RadioItem` w pozycji `true`, co pozwala na wybór tylko jednej pozycji (stopnie albo radiany) oraz uwidacznia zaznaczoną pozycję w postaci czarnej kropki obok wybranej pozycji menu. Domyślna pozycja została wybrana przez ustawienie jej właściwości `Checked` w pozycji `true`, podczas przygotowywania formy do pracy, czyli po wybraniu przez użytkownika pozycji „Wartości funkcji” na poziomie formy powitalnej.

```
Form3.stopnie1.Checked:=true; - zaznacz domyślną pozycję stopnie1
```

Wybór myszką jednej z dwóch pozycji z menu „Rodzaj wyniku” uruchamia zdarzenie `OnClick` od wybranej pozycji. Procedura obsługująca to zdarzenie przypisuje właściwości `Checked` wybranemu elementowi pola wartość `true` oraz ponownie wywołuje procedurę zdarzenia `FormKeyDown`, by funkcja `Wynik` ponownie wyliczyła wartość funkcji po zmianie rodzaju wyniku. Poniżej znajduje się kod procedury obsługi zdarzenia `OnClick` po wybraniu pozycji „radiany” – dla pozycji „stopnie”, kod procedury jest porównywalny.

```
procedure TForm3.radiany1Click(Sender: TObject);
var z2x:Word;
begin
     radiany1.Checked:=true; - ustaw pole Checked należące do...
                               podmenu radiany1 w pozycji true
     z2x:=13; - przypisz zmiennej z2x wartość odpowiadającą kodowi ASCII...
                znaku ENTER
     Form3.FormKeyDown(self,z2x,[]) - wywołaj procedurę zdarzenia
end;
```

By wywołać procedurę `FormKeyDown`, trzeba jej przekazać trzy parametry:

– `self` jest referencją do bieżącego obiektu, czyli tej formy;

– wartość liczbowa znaku ENTER musi być dostarczona poprzez zmienną, stąd dodatkowa zmienna `z2x` z przypisanym kodem ASCII znaku ENTER równym 13;
– `[]` zapis ten informuje procedurę, że razem z klawiszem ENTER nie został naciśnięty inny klawisz klawiatury ani myszki komputerowej.

VI. 3. Zmiana funkcji

Zmiany funkcji może dokonać system edukacji na pewnym jego etapie, poprzez wywołanie procedury `Nastepny`, która po ustawieniu zmiennej logicznej `czy_zaczac` w pozycji `false` wywołuje kolejną procedurę `Dla_nastepny`. Rozdzielenie tych dwóch procedur pozwala na automatyczne przygotowanie górnej formy `Form3` dla następnego przykładu funkcji, gdy zmianę tę wymusza system edukacji.

```
procedure TForm3.Nastepny;
begin
     czy_zaczac:=false; – ustaw zmienną czy_zaczac w pozycji false
     Form3.Wyczysc(false); – wyczyść formę Form3 z treści graficznych,...
                             wstawianych metodami Canvas

     Form3.Dla_nastepny  wywołaj procedurę Dla_nastepny
end;
```

Zmiany funkcji może dokonać także użytkownik przez kliknięcie w formę `Form3` w pewnej odległości od pisanej funkcji. Wywoływane jest wówczas zdarzenie `FormMouseDown`, w którym współrzędne kliknięcia są porównywane ze współrzędnymi wprowadzonych znaków funkcji. Jeśli żaden znak nie został zakwalifikowany jako znak kliknięcia (mechanizm ten szczegółowo opisany został w rozdziale III.3), wywołana zostaje procedura `Dla_nastepny`, która uruchamia monit z pytaniem o zmianę funkcji. Jeśli odpowiedź będzie twierdząca, wywoływana jest kolejna procedura `Wyczysc` z opcją `false`, nakazująca procedurze dodatkowe zwolnienie pamięci zajmowanej przez elementy podpowiedzi. Sama procedura, niezależnie od podanej opcji, zwalnia pamięć zajmowaną przez elementy edytora równań, usuwa wpisaną funkcję z formy, przywraca parametry graficzne dotyczące czcionki i ustawia zmienną `stat_kur` w pozycji `false`, co jest niezbędne przed wprowadzeniem pierwszego znaku.

```
procedure TForm3.Dla_nastepny;
begin
     if not czy_zaczac then – czy procedura została wywołana przez system…
                              edukacji?
     begin – jeśli tak, to...
          Refresh; – wyczyść formę z treści graficznych, wstawianych
                     metodami Canvas
          AX:=AXX+Canvas.TextWidth(tekst1); – wylicz wartość współrzędnej ...
                          poziomej, uwzględniając odstęp od krawędzi formy AXX...
                          i długość tekstu zawartego w zmiennej tekst1

          Canvas.TextOut(AXX,AY,tekst1); – wstaw na formie tekst...
                                           zawarty w zmiennej tekst1

          Form3.Timer1.Enabled:=true; – odblokuj komponent Timer1...
                                        (uruchom mruganie kursora)
```

```
                czy_zaczac:=true; - ustaw zmienną czy_zaczac w pozycji true, by ...
                   kolejne wywołanie procedury wymusiło pojawienie się formy komunikatów
                Form3.SetFocus; - uczyń formę Form3 aktywną
                Form3.ShowHint:=true - odblokuj pojawianie się podpowiedzi...
                                       w postaci tzw. chmurki
            end else - jeśli procedura wywołana została przez kliknięcie w formę, to...
                begin
                     Form3.Timer1.Enabled:=false; - zablokuj komponent Timer1
                     Form9.Wynik9(22); - wywołaj procedurę komunikatów z opcją...
                                         propozycji zmiany funkcji
                     if Form9.ShowModal=1 then - wywołaj modalnie formę...
                     komunikatów. Czy po jej zamknięciu, forma zwróciła wartość jeden?
                     begin - jeśli tak, to...
                          if tryb_pracy>0 then - czy zmienna tryb_pracy...
                                                 nie wskazuje na wybór ćwiczeń?
                          begin - jeśli forma nie pracuje w trybie ćwiczeń, to...
                               Form3.Wyczysc(true); - wyczyść górną formę...
                                          Form3 z treści graficznych...
                                          oraz pamięć zajmowaną przez elementy...
                                          edytora równań górnej formy Form3
                               Form5.Wyczysc; - uwolnij pamięć zajmowaną przez...
                                        elementy edytora równań dolnej formy Form5
                               Form5.Refresh - wyczyść dolną formę...
                                               z treści graficznych
                          end else - jeśli tryb pracy wskazuje na ćwiczenia, to...
                              begin
                                   if Form2<>nil then - czy dolna forma…
                                                        edukacji istnieje w pamięci?
                                      Form2.Zwieksz_zle_odp; jeśli tak, to...
                                        zwiększ ilość błędnych odpowiedzi o jeden
                                   Form3.Wyczysc(false) - wyczyść górną formę...
                                                          z treści graficznych, uwolnij pamięć...
                                                          z elementów edytora równań oraz,...
                                                          dzięki parametrowi false, usuń...
                                                          z pamięci elementy podpowiedzi
                              end;
                          AX:=AXX+Canvas.TextWidth(tekst1); - wylicz wartość...
                          współrzędnej poziomej, uwzględniającą odstęp od krawędzi formy...
                          AXX i długość tekstu zawartego w zmiennej tekst1
                          Canvas.TextOut(AXX,AY,tekst1) - wstaw na formie tekst...
                                                          zawarty w zmiennej tekst1
                     end;
                     Form3.Timer1.Enabled:=true; - odblokuj komponent Timer1...
                                                   (uruchom mruganie kursora)
                end;
            Form3.SetFocus; - uczyń formę Form3 aktywną
        end;
```

Szczegółowego opisu wymaga procedura usuwająca podpowiedzi, czyli `Usun_podpowiedzi`. W opisywanym module dostępne są jedynie wskaźniki na elementy podpowiedzi, będące zmiennymi globalnymi należącymi do klasy `pochodna`, dlatego dostęp do nich odbywa się poprzez zmienną klasy `poch1`. Przed zwolnieniem pamięci zajmowanej przez każdy z elementów podpowiedzi, najpierw uwalniana jest pamięć zajmowana przez pola

fun_A i fun_B, w które wpisane zostały odpowiednio funkcja i pochodna. Owe pola są wskaźnikami na łańcuchy znakowe typu PChar, dla których pamięć została przydzielona dynamicznie o wielkości proporcjonalnej do ich zawartości. Do zwolnienia pamięci przez nie zajmowane zastosowana została procedura FreeMem, której oprócz wskaźnika na usuwany łańcuch, podaje się ilość bajtów jaką ten łańcuch zajmuje. Wartość ta jest iloczynem liczby znaków zajmujących usuwany łańcuch oraz ilość bajtów zajmowanych przez pojedynczy znak typu PChar – ilość bajtów zwraca funkcja sizeof z podanym w parametrze typem zmiennej. Po uwolnieniu pamięci zajmowanej przez łańcuchy znakowe, można już uwolnić pamięć zajmowaną przez element podpowiedzi. Odpowiednią instrukcją do tego celu jest dispose, której w parametrze podaje się jedynie wskaźnik na usuwany element podpowiedzi. Wszystkie instrukcje uwalniające pamięć zostały zamknięte w pętli while..do, której opuszczenie nastąpi po uwolnieniu ostatniego elementu podpowiedzi.

```
procedure TForm3.Usun_podpowiedzi;
var poz1:byte;
begin
     poch1.t_biez:=poch1.t_nau; - przypisz wskaźnikowi t_biez adres...
                                  pierwszego elementu podpowiedzi
     while poch1.t_biez<>nil do - wykonaj krok pętli, póki wskaźnik t_biez...
                                  nie ma adresu pustego
     begin
          if poch1.t_biez.fun_A<>nil then - czy łańcuch fun_A należący
                                            do bieżącego elementu podpowiedzi nie jest pusty?
          begin - jeśli nie jest pusty, to...
               poz1:=StrLen(poch1.t_biez.fun_A); - zapamiętaj...
               w zmiennej poz1 liczba znaków zawartych w łańcuchu fun_A
               FreeMem(poch1.t_biez.fun_A,poz1*sizeof(PChar));
                na postawie liczby znaków i zajętości pamięci...
                przez jeden znak, uwolnij pamięć zajmowaną...
                przez łańcuch fun_A
               poch1.t_biez.fun_A:=nil - przypisz zmiennej łańcuchowej...
                                         adres pusty
          end;
          if poch1.t_biez.fun_B<>nil then - (jak wyżej, w stosunku do...
                                            łańcucha fun_B)
          begin
               poz1:=StrLen(poch1.t_biez.fun_B);
               FreeMem(poch1.t_biez.fun_B,poz1*sizeof(PChar));
               poch1.t_biez.fun_B:=nil;
          end;
          poch1.t_biez:=poch1.t_biez^.prawa; - przypisz wskaźnikowi...
                                               t_biez adres następnego elementu podpowiedzi
          dispose(poch1.t_nau); - uwolnij pamięć zajmowaną przez...
          element podpowiedzi, której adres znajduje się we wskaźniku t_nau
          poch1.t_nau:=poch1.t_biez przypisz wskaźnikowi t_nau...
           adres elementu podpowiedzi, zawarty we wskaźniku t_biez
     end;
end;
```

VII. Moduł przykładów losowanych – Unit4

Moduł ten jest aktywny tylko wtedy, gdy użytkownik wybierze z menu formy powitalnej Form1 pozycję „ćwiczenia – przykłady losowane". Składa się on z edytora równań tylko do odczytu oraz m.in. procedury zajmującej się losowaniem przykładów funkcji. Nim losowanie będzie możliwe, musi być sprawdzona obecność pliku tekstowego zawierającego przykłady funkcji. Dokładna ścieżka do tego pliku wraz z jego nazwą znajduje się w stałej modułu o nazwie sciezka.

```
const sciezka='.\o2.txt'; – ścieżka do pliku tekstowego z przykładami funkcji
```

Jeśli plik został zlokalizowany, zostaje zliczona liczba rekordów zawartych w pliku. Liczba ta zostaje zapamiętana w zmiennej prywatnej ilość należącej do klasy TFotm4 modułu – zmienna ta będzie wyznacznikiem zakresu losowanych przykładów. Gdyby plik nie został zlokalizowany, zostaje wyświetlona forma komunikatów z informacją o nie znalezieniu pliku. Opisane czynności odbywają się w funkcji Poczatek, która zwraca wartość logiczną, zależną od zidentyfikowania lub nie pliku tekstowego. Wartość ta wykorzystywana jest przez moduł główny do kontynuowania lub zaniechania przygotowań form roboczych do pracy.

```
function TForm4.Poczatek;
var ciag:string;
    b1:byte;
begin
     AY:=a1; – w zmiennej AY zapamiętaj wartość współrzędnej pionowej,...
               od której rozpocznie się wstawianie na formie wylosowanej funkcji

     if FileExists(sciezka) then – czy plik o podanej nazwie i lokalizacji istnieje?
     begin – jeśli tak, to...
          AssignFile(zm_pl,sciezka); – skojarz zlokalizowany plik ze zmienną...
                                       plikową zm_pl

          FileMode:=fmOpenRead; – określ sposób dostępu do pliku – tylko odczyt
          Reset(zm_pl); – otwórz plik do odczytu
          b1:=0; – zmiennej b1 wpisz liczbę zero – licznik iteracji pętli
          while not eof(zm_pl) do – wykonaj krok pętli, póki funkcja eof nie zwróci...
                                    wartości true oznaczającej koniec pliku
          begin
               Readln(zm_pl,ciag); – do zmiennej ciag wczytaj kolejny wiersz ...
                                     z pliku, za pośrednictwem zmiennej plikowej...
                                     zm_pl

               inc(b1) zwiększ –zwiększ o jeden wartość w zmiennej b1
          end;
          CloseFile(zm_pl); – zamknij plik
          czy_zaczac:=false; – ustaw zmienną logiczną czy_zaczac w pozycji...
                               false, by pierwszy wylosowany przykład został ...
                               wstawiony na formie w sposób automatyczny

          ilosc:=b1; – zapamiętaj w zmiennej prywatnej liczbę iteracji równą liczbie…
                       wierszy zawartych w pliku, powiększoną o jeden

          Result:=false – za pośrednictwem zmiennej Result zwróć wartość false
     end else jeśli plik nie został zlokalizowany, to...
         begin
```

```
                Form9.Wynik9(20); – wywołaj procedurę komunikatów z opcją...
                                    powodującą wyświetlenie informacji o braku pliku
                Form9.ShowModal; – wyświetl modalnie formę komunikatów
                Result:=true – za pośrednictwem zmiennej Result zwróć wartość true
            end;
end;
```

Choć plik otwierany będzie tylko do odczytu, to obowiązkowo należy zmodyfikować zmienną globalną `FileMode` należącą do modułu `System` tak, by próba otwarcia pliku zlokalizowanego na płycie CD nie spowodowała pojawienie się błędu systemu, informującego o niemożności otwarcia pliku. Wykorzystany tu został literał `fmOpenRead`, który jest równoważny z wartością zero i akceptowany przez zmienną `FileMode`. Ustawienie tej zmiennej spowoduje, że otwierany plik może mieć atrybut tylko do odczytu.

VII. 1. Losowanie funkcji z pliku – funkcja `wylosowany`

Funkcja ta ma za zadanie wylosowanie przykładu funkcji z pliku tekstowego i zwrócenie tego przykładu w postaci łańcucha typu `string`. Ponieważ plik ma strukturę tekstu, w którym każdy wiersz zakończony jest znakiem końca wiersza, zastosowana zmienna plikowa, zadeklarowana w części prywatnej klasy `TForm4`, jest typu `Text`. Wczytywanie kolejnych wierszy ze skojarzonej z plikiem zmiennej plikowej do zmiennej typu `sting` odbywa się w pętli `while..do` i trwa do momentu, aż numer kolejny iteracji pętli będzie taki sam jak numer wylosowany przez funkcję `Random`. Funkcja losująca otrzymuje w parametrze zmienną `ilosc`, przechowującą wcześniej zliczoną liczbę wierszy zawartych w pliku. Zaraz po wylosowaniu przykładu funkcji, pętla jest opuszczona instrukcją `break`, a plik zostaje zamknięty.

```
function TForm4.Wylosowany;
var ciag:string;
    los1,i2:integer;
begin
     los1:=Random(ilosc); – wylosuj liczbę z zakresu od 0 do wartości...
                           przechowywanej w zmiennej ilosc,...
                           pomniejszonej o jeden
     AssignFile(zm_pl,sciezka); – skojarz zlokalizowany plik ze zmienną...
                                  plikową zm_pl
     Reset(zm_pl); – otwórz plik do odczytu
     i2:=0; – wyzeruj zmienną i2
     while not eof(zm_pl) do – wykonaj krok pętli, póki funkcja eof nie zwróci...
                               wartości true oznaczającej koniec pliku
     begin
          Readln(zm_pl,ciag); – do zmiennej ciag wczytaj kolejny wiersz z pliku...
                                za pośrednictwem zmiennej plikowej zm_pl
          if i2=los1 then break; – jeśli kolejny numer iteracji pętli jest równy...
                                   wylosowanej liczbie, opuść pętlę
          inc(i2) – zwiększ o jeden wartość w zmiennej i2 – licznik iteracji pętli
     end;
     Result:=ciag; – za pośrednictwem zmiennej Result, zwróć wylosowany wiersz
     CloseFile(zm_pl) – zamknij plik
end;
```

VII. 2. Wylosowanie pierwszej funkcji

W wyniku wywołania procedury `Show` należącej do klasy `TForm1`, uaktywnia się zdarzenie uzyskania aktywności przez formę `Form4`, czego efektem jest wywołanie procedury `FormActive`. W ciele tej procedury znajdują się instrukcje utworzenia w pamięci dwóch klas: `pochodna` oraz `wartosc`. Za pomocą pierwszej klasy znajdowana będzie pochodna wylosowanej funkcji, natomiast druga klasa wyliczy wartość liczbową pochodnej. Instrukcje warunkowe, poprzedzające polecenia tworzenia klas, stanowią zabezpieczenie przed próbą utworzenia klas już istniejących w pamięci. Trzecią instrukcją jest wywołanie procedury `FormClick`, inicjowaną po kliknięciu myszką w formę. Za pomocą zmiennej `self` przekazany jest procedurze adres pamięci, w której rezyduje bieżący obiekt, czyli forma `Form4`.

```
procedure TForm4.FormActivate(Sender: TObject);
begin
     if poch1=nil then poch1:=pochodna.Create; - jeżeli zmienna poch1...
                                klasy pochodna jest zainicjowana adresem pustym,...
                                utwórz w pamięci obiekt, wywołując metodę Create

     if wart1=nil then wart1:=wartosc.Create; - jeżeli zmienna wart1...
                                klasy wartosc jest zainicjowana adresem pustym,...
                                utwórz w pamięci obiekt, wywołując metodę Create

     Form4.FormClick(self); - wywołaj procedurę FormClick, podając jej...
                              w parametrze referencję do bieżącego obiektu
end;
```

Po wywołaniu procedury `FormClick`, realizowana jest ta część kodu, której wykonanie uzależnione jest od ustawienia zmiennej logicznej `czy_zaczac` w pozycji `false`. Pozwala to na automatyczne, czyli bez udziału użytkownika, wstawienie na formie funkcji wylosowanej z pliku. Zaraz po spełnieniu warunku, zmienna `czy_zaczac` zostaje ustawiona w pozycji `true`, umożliwiając przy kolejnym wywołaniu procedury komunikację z użytkownikiem. Po wylosowaniu funkcji, zostaje przydzielona pamięć dla łańcucha typu `PChar`, którego rozmiar jest proporcjonalny do ilości znaków zawartych w łańcuchu `t1` typu `string`, zawierającym wylosowaną funkcję. Za pomocą funkcji `StrPCopy`, zawartość łańcucha `t1` zostaje skopiowana z dodatkową konwersją do łańcucha `znatab` typu `Pchar`, który przekazany zostaje procedurze `Na_ekran`. Po opuszczeniu procedury łańcuch `znatab` będzie zawierał już tylko pochodną wylosowanej funkcji, dlatego przed uwolnieniem pamięci zajmowanej przez ten łańcuch, zliczona zostaje liczba znaków zawartych w łańcuchu – posłuży ona do określenia wielkości pamięci zajmowanej przez łańcuch. Na jej podstawie, pamięć zostaje uwolniona.

```
procedure TForm4.FormClick(Sender: TObject);
var t1:string;
    znatab:PChar;
    iz2:Integer;

    procedure Na_ekran;
    . . .
```

```
begin
     if not czy_zaczac then - czy zmienna czy_zaczac jest ustawiona...
                              w pozycji false?
     begin - jeśli tak, to...
          czy_zaczac:=true; - ustaw zmienną czy_zaczac w pozycji true
          t1:=Wylosowany; - wywołaj funkcję Wylosowany i zapamiętaj...
                            wylosowaną funkcję w zmiennej t1 typu string
          iz2:=Length(t1); - z ilu znaków składa się funkcja w łańcuchu t1?
          GetMem(znatab,iz2*sizeof(PChar)); - przydziel pamięć dla...
          łańcucha znatab typu PChar o wielkości proporcjonalnej do liczby znaków
          StrPCopy(znatab,t1); - skopiuj zawartość łańcucha t1 do łańcucha ...
                         znatab, dokonując konwersji z typu string do PChar
          Na_ekran; - wstaw wylosowaną funkcję na formie
          iz2:=StrLen(znatab); - z ilu znaków składa się łańcuch znatab po...
                                 opuszczeniu procedury Na_ekran?
          FreeMem(znatab,iz2*sizeof(PChar)) - zwolnij pamięć...
                                              zajmowaną przez łańcuch znatab
     end else
     . . .
```

VII. 3. Losowanie funkcji wymuszone przez system edukacji

Gdy proces edukacji dla danego przykładu funkcji dobiegł końca a użytkownik chce kontynuować edukację, procedura `System_edukacji` znajdująca się w module edukacji może wymusić wylosowanie innej funkcji. Odbywa się to przez wywołanie procedury `Na-stepny`, która ustawi zmienną logiczną `czy_zaczac` w pozycji `false` oraz wywoła procedurę `FormClick` powodując wylosowanie innej funkcji. Samo losowanie funkcji i jej wstawianie na formie odbywa się identycznie jak dla pierwszej funkcji.

```
procedure TForm4.Nastepny;
begin
     czy_zaczac:=false; - ustaw zmienną logiczną czy_zaczac w pozycji...
                          false, by wylosowany przykład został wstawiony ...
                          na formie w sposób automatyczny
     FormClick(self) - wywołaj procedurę FormClick, podając jej w parametrze...
                       referencję do bieżącego obiektu
end;
```

VII. 4. Losowanie funkcji wymuszone przez użytkownika

Kliknięcie myszką w formę spowoduje wygenerowanie zdarzenia, w wyniku którego wywołana zostaje procedura `FormClick`, a skoro zmienna `czy_zaczac` jest już ustawiona w pozycji `true`, wyświetlona zostaje forma komunikatów zawierająca propozycję wylosowania kolejnego przykładu. Naciśnięcie przycisku z napisem `TAK` spowoduje zamknięcie formy komunikatów i zwrócenie za pośrednictwem zmiennej `ModalResult` wartości jeden. Wykrycie tej wartości w instrukcji warunkowej spowoduje: uruchomienie pętli uwalniającej

pamięć zajmowaną przez elementy edytora równań, zwiększenie o jeden liczby błędnych odpowiedzi oraz wylosowanie i wstawienie wylosowanej funkcji na formie.

```
if not czy_zaczac then – czy zmienna czy_zaczac jest ustawiona w pozycji false?
begin
     . . .
end else – jeśli nie, to...
    begin
         Form9.Wynik9(22); – wywołaj procedurę komunikatów z opcją zmiany funkcji
          if Form9.ShowModal=1 then – wyświetl modalnie formę komunikatów. Czy
                                      po zamknięciu formy, zwróciła ona wartość jeden?
         begin – jeśli tak, to...
              wsb:=wsp; – przypisz wskaźnikowi bieżącemu wsb edytora równań
                          adres pierwszego elementu, zapamiętanego we
                          wskaźniku wsp
              while wsb<>nil do – wykonaj krok pętli, póki wskaźnik wsb...
                                  nie ma adresu pustego
              begin
                   wsb:=wsb^.prawa; – do wskaźnika wsb wczytaj adres…
                                      następnika
                   dispose(wsp); – zwolnij pamięć zajmowaną przez element...
                                   edytora wskazywany przez wskaźnik wsp
                   wsp:=wsb – przypisz wskaźnikowi wsp adres następnika
              end;
              Form2.Zwieksz_zle_odp; – zwiększ o jeden liczby błędnych…
                                       odpowiedzi
              t1:=Wylosowany; – wywołaj funkcję Wylosowany – wylosowaną...
                                funkcję zapamiętaj w zmiennej t1
              iz2:=Length(t1); – z ilu znaków składa się łańcuch t1?
              GetMem(znatab,iz2*sizeof(PChar)); – przydziel pamięć dla...
                             łańcucha typu PChar o wielkości proporcjonalnej...
                             do zliczonej liczby znaków
              StrPCopy(znatab,t1); – skopiuj zawartość łańcucha t1 do znatab...
                                     dokonując konwersji z typu string do PChar
              Na_ekran; – wstaw wylosowaną funkcję na formie
              iz2:=StrLen(znatab); – z ilu znaków składa się łańcuch znatab...
                                     po opuszczeniu procedury Na_ekran?
              FreeMem(znatab,iz2*sizeof(PChar)); – na podstawie zliczonej...
                      liczby znaków, uwolnij pamięć zajmowaną przez łańcuch znatab
              Form2.SetFocus – przywróć aktywność formie edukacji Form2
         end else Form2.SetFocus; – jeśli po zamknięciu formy komunikatów,...
                                    zwróciła ona inną wartość niż jeden,...
                                    przywróć aktywność formie edukacji Form2
    end;
```

VII. 5. Wyprowadzenie wylosowanej funkcji na ekran – procedura `Na_ekran`

Procedura ta rozpoczyna pracę od uwolnienia pamięci zajmowanej przez elementy edytora równań oraz przez elementy podpowiedzi. Ponadto, ukryta zostaje forma podpowiedzi oraz odświeżone zostają ustawienia dotyczące czcionki. Wszystkie wymienione czynności wykonuje procedura `Wyczysc`. Po jej opuszczeniu, wyznaczana zostaje początkowa współrzędna pozioma dla pierwszego, wprowadzanego znaku funkcji. Współrzędna ta jest sumą wartości równej 20 zawartej w stałej `AXX` i długości przedrostka '`f(x)= `' wyliczonej przez funkcję `TextWidth` należącej do klasy `Canvas`. Po wstępnych przygotowaniach, w pętli `while..do` wywoływana jest procedura `Wstaw_znak`, której w parametrze podawany jest kolejny znak z łańcucha `znatab` zawierający wcześniej wylosowaną funkcję. Procedura `Wstaw_znak` należy do edytora równań, dlatego jej opis znajduje się w rozdziale XV. Po opuszczeniu pętli, wywołana zostaje procedura `blok_glowny`, której zadaniem jest zamiana funkcji, zawartej w dostarczonym w parametrze łańcuchu `znatab`, na pochodną (procedura znajduje się w osobnym module, dlatego wywołanie jej odbywa się poprzez zmienną klasy `pochodna` o nazwie `poch1`). Jeśli zamiana przebiegła bezbłędnie, losowana jest wartość dla zmiennej 'x' oraz pełnej pochodnej w sposób identyczny jak ten, opisany w rozdziale VI.1.1.1. Po znalezieniu wartości i zapamiętaniu ich w zmiennych globalnych zawartych w osobnym module, wyznaczony zostaje czas oczekiwania na wpisanie pochodnej przez użytkownika – sposób wyznaczania czasu oczekiwania został dokładnie opisany w rozdziale V.6.9. Po wyczyszczeniu formy z treści graficznych, wstawianych metodami `Canvas` oraz po umieszczeniu na formie przedrostka '`f(x) = `' zawartego w stałej o nazwie `tekst1`, wywołana zostaje procedura `Wyswietl`, która na podstawie wszystkich utworzonych elementów edytora równań, wstawia funkcję na formie w sposób matematyczny. Procedura ta również należy do edytora równań, dlatego jej opis został zamieszczony w rozdziale XV.4.

```
procedure Na_ekran;
var iz9:integer;
    x4,x4p:double;
begin
     Form4.Wyczysc; - wywołaj procedurę Wyczysc
     AX:=AXX+Canvas.TextWidth(tekst1); - określ współrzędną poziomą,…
                                 od której rozpocznie się wstawianie znaków wylosowanej
                                 funkcji

     iz9:=0; - zainicjuj wartością zero zmienną iz9
     while znatab[iz9]<>#0 do - wykonaj krok pętli, póki znak wskazany...
                                 zmienną iz9 w sczytywanym łańcuchu...
                                 nie jest pusty
     begin
          Form4.Wstaw_znak(znatab[iz9]); - wywołaj procedurę edytora...
                                 Wstaw_znak, przekazując jej w parametrze...
                                 kolejny sczytany znak z łańcucha

          inc(iz9) - zwiększ wartość w zmiennej iz9 o jeden
     end;
     poch1.blok_glowny(znatab,true); - wywołaj procedurę blok_glowny...
                 przekazując jej w parametrze adres łańcucha zawierającego funkcję...
                 oraz wartość true nakazującą procedurze przygotowanie podpowiedzi

     if not poch1.bl then - jeśli po powrocie z procedury blok_glowny, zmienna...
       błędu klasy pochodna posiada wartość false, co oznacza, że znalezienie pochodnej...
       odbyło się bez błędu, to znajdź wartość dla zmiennej 'x'
```

```
        begin
              iz9:=1;
              x4:=0.0;
              repeat
                    x4p:=Random(100)/100;
                    x4p:=x4p+Random(20)-10;
                    if (iz9 mod 7)=0.0 then x4p:=x4p+50;
                    inc(iz9);
                    if iz9>100 then break;
                    x4:=wart1.wynik(znatab,x4p)
              until not wart1.bl;
              if wart1.bl then
              begin
                   Form9.Wynik9(24);
                   Form9.ShowModal
              end else
              begin
                   Form5.iks5:=x4p;
                   Form5.waf5:=x4
              end;
        end;
        iz9:=StrLen(znatab);
        Form5.czas:=trunc((iz9*20)/sqrt(iz9))-10;
        Refresh; - wyczyść górną formę Form4 z treści graficznych
        Canvas.TextOut(AXX,AY,tekst1); - wstaw przedrostek 'f(x) = ' zawarty...
                    w stałej tekst1, w miejscu wskazanym zmiennymi AXX i AY
        Form4.Wyswietl - wywołaj procedurę edytora Wyswietl wstawiającej...
                         na formie wylosowaną funkcję
end;
```

VII. 6. Zamknięcie formy

Formy nie można zamknąć w sposób bezpośredni, gdyż nie ma ona dostępnego menu, w którym istniałaby opcja zamknięcia aplikacji. Takie menu posiada forma edukacji usytuowana na dole ekranu, w którym można wybrać opcję zamknięcia aplikacji. Wybranie tej pozycji z menu w formie edukacji spowoduje wywołanie procedury `Zamknij`, która w pierwszej kolejności wywołuje procedurę `Wyczysc` uwalniającą pamięć zajmowaną przez wykorzystane elementy edytora równań i podpowiedzi. Następnie uwalniane są pozostałe obiekty uprzednio utworzone. Na koniec wywołana zostaje procedura `Destroy`, która uwalnia pamięć zajmowaną przez klasę `TForm4`, po czym zmiennej klasy przypisany zostaje adres pusty by podczas kolejnej próby utworzenia obiektu jednoznacznie wskazywała na jego brak.

```
procedure TForm4.Zamknij;
begin
     Form4.Wyczysc; - wywołaj procedurę Wyczysc
     if Form5<>nil then - czy wskaźnik Form5 nie posiada adresu pustego?
     begin - jeśli nie, to...
          Form5.Destroy; - usuń z pamięci obiekt wskazywany przez wskaźnik Form5
          Form5:=nil - przypisz wskaźnikowi Form5 identyfikator adresu pustego
     end;
     if poch1<>nil then - czy wskaźnik poch1 nie posiada adresu pustego?
     begin - jeśli nie, to...
          poch1.Free; - usuń z pamięci klasę pochodna wskazywaną przez ten
                        wskaźnik
          poch1:=nil; - przypisz wskaźnikowi poch1 identyfikator adresu pustego
```

```
      end;
      if wart1<>nil then - czy wskaźnik wart1 nie posiada adresu pustego?
      begin - jeśli nie, to...
            wart1.Free; - usuń z pamięci klasę wartosc wskazywaną przez ten wskaźnik
            wart1:=nil - przypisz wskaźnikowi wart1 identyfikator adresu pustego
      end;
      Form4.Destroy; - usuń z pamięci obiekt wskazywany przez wskaźnik Form4
      Form4:=nil - przypisz wskaźnikowi Form4 identyfikator adresu pustego
end;
```

VIII. Moduł podpowiedzi lub wyjściowy – Unit5

Jeśli z menu głównego aplikacji użytkownik wybierze opcje ćwiczeń, forma będąca częścią tego modułu, będzie pracowała jako podpowiedź dla systemu edukacji. Wybór pochodnych bez ćwiczeń lub wartości funkcji spowoduje przygotowanie formy do pracy jako roboczej usytuowanej w lewym, dolnym rogu ekranu. Wobec podwójnej roli jaką może pełnić forma, większość ustawień jej parametrów dokonywane jest dopiero po ustaleniu charakteru pracy formy. Ustawieniem parametrów formy zajmuje się procedura `Poczatek` z odpowiednią opcją logiczną, będącą wynikiem wyboru dokonanego przez użytkownika. Niezależnie od tego, w jakim charakterze forma będzie pracowała, jej właściwość `Enable` ustawiona została w pozycji `false`, co sprawia, że wszelkie reakcje od klawiatury lub myszki komputerowej są przez nią ignorowane.

VIII. 1. Praca formy w charakterze podpowiedzi

Wywołanie procedury `Poczatek` z opcją `true` ustawia formę pod kątem podpowiedzi. W pierwszych instrukcjach procedury utworzony zostaje komponent `Image1` z klasy `Timage`, do którego będą wstawiane podpowiedzi z odpowiedniego pliku graficznego. Komponent ten będzie dostosowywał się do rozmiaru obrazka, przez co jego rozmiary pozostaną niezmienione. W tym trybie pracy, parametry czcionki oraz kolor tła formy, są niezależne od indywidualnych ustawień dokonywanych przez użytkownika. Ma to na celu dostosowanie formy do kolorów i treści zawartych w plikach graficznych, przez co poszczególne części podpowiedzi: pochodzące z pliku graficznego oraz wstawiane przez edytor równań, mają takie same parametry graficzne, co pozytywnie wpływa na estetykę samej podpowiedzi. Zmienne kolorów: czcionki i tła formy (`co5` i `to5`), jak również rozmiaru czcionki (`ws5`), zostały zadeklarowane w części prywatnej klasy `TForm5`. Pozwala to na wielokrotne użycie tych zmiennych, np. podczas przerysowywania zawartości formy czy odświeżania parametrów czcionki.

```
procedure TForm5.Poczatek(czy_nau:boolean);
. . .
begin
     . . .
     if czy_nau then – czy parametr czy_nau posiada wartość true?
     begin – jeśli tak, to...
          Image1:=TImage.Create(self); – utwórz w pamięci komponent Image1
          Image1.Parent:=Form5; – określ rodzica dla nowego komponentu
          Image1.Left:=10; – ustal współrzędną poziomą komponentu Image1
          Image1.AutoSize:=true; – automatycznie dopasuj rozmiar komponentu...
                                   do rozmiaru obrazka

          co5:=clBlack; – zapamiętaj w zmiennej co5 kolor czcionki – czarny
          to5:=clInfoBk; zapamiętaj w zmiennej to5 kolor tła formy – piaskowy
          ws5:=22; – zapamiętaj w zmiennej ws5 rozmiar czcionki
          Canvas.Font.Name:='Arial' – ustaw krój czcionki dla znaków...
                                      wstawianych na formie
```

```
        end else
        . . .
        end;
        Canvas.Font.Color:=co5; - na podstawie zmiennej co5 ustal kolor czcionki
        Canvas.Brush.Color:=to5; - na podstawie zmiennej to5 ustal kolor...
                                   wypełnienia czcionki
        Canvas.Font.Height:=ws5; - na podstawie zmiennej ws5 ustal wysokość czcionki
        Canvas.Brush.Style:=bsSolid; - ustal styl wypełnienia rysowanych linii...
                                       pełnym kolorem

        Canvas.Font.Style:=[]; - określi styl czcionki bez pogrubiania, podkreślania...
                                 czy pochylania tekstu
        (właściwość ta jest typu zbiorowego, dlatego brak dodatkowego formatowania tekstu...
         musi być zaznaczone przez pustą parę nawiasów kwadratowych)
        Form5.Color:=to5 - na podstawie zmiennej to5 ustaw kolor tła formy
end;
```

Dla systemu edukacji przygotowana została procedura `Z_Edukacji`, która zarządza formą podpowiedzi zgodnie z wymogami systemu edukacji. Procedura otrzymuje w parametrach: funkcję elementarną, pochodną funkcji elementarnej, ścieżkę do pliku podpowiedzi oraz numer bieżącego trybu edukacji. Pracę swą rozpoczyna od sprawdzenia, czy dostarczona została ścieżka do pliku podpowiedzi – jeśli tak, w kolejnej instrukcji `FileExists` sprawdzana jest obecność pliku we wskazanej lokalizacji – jeśli plik istnieje, zostaje on przypisany komponentowi wskazywanemu zmienną `Image1`. Określony jest także górny punkt wstawienia obrazka na formie, którego wartość uzależniona jest od wartości parametru `tryb`, istnieją dwie możliwości:

- wartość parametru `tryb` równa jest zero – współrzędna pionowa komponentu równa jest wartości zawartej w stałej procedury `czop5`. Jest to odległość od górnej krawędzi formy równej 10 pikseli. W tym przypadku, plik graficzny jest jedynym elementem podpowiedzi, dlatego można go umieścić w górnej części formy;
- wartość parametru `tryb` jest inna niż zero – współrzędna pionowa komponentu jest obniżona od górnej krawędzi formy o iloraz wysokości obszaru roboczego formy oraz liczby 2,5. Obniżenie komponentu z załadowanym obrazkiem pozwala na umieszczenie nad nim funkcji, której dotyczy podpowiedź zawarta w obrazku.

Jeśli ścieżka do pliku podpowiedzi jest pusta, komponent `Image1` zostaje ukryty poprzez wywołanie procedury `Hide`, należącej do klasy `TImage`. Wstawianie pozostałych elementów podpowiedzi na formie uzależnione jest od niezerowej wartości parametru `tryb`. Właściwe miejsce ich wstawienia wymaga wyliczenia dla nich współrzędnych startowych, zawartych w zmiennych modułu `AX` i `AY`. Współrzędna pozioma `AX` jest sumą niewielkiego odstępu od lewej krawędzi formy oraz długości przedrostka funkcji ‘`f(x)=`‘ lub pochodnej ‘`f'(x)=`‘, zawartych w stałych `tekst2` i `tekst3`. Z kolei wyznaczenie współrzędnej pionowej `AY` wymaga dodatkowego jej skorygowania o ponadnormatywną wysokość wstawianej funkcji czy jej pochodnej, dlatego współrzędna ta jest ustalana dwuetapowo. Wstępna współrzędna pionowa dla funkcji jest równa sumie podwojonej wartości zawartej w stałej `czop5` i wysokości czcionki zapamiętanej w zmiennej `ws5`. Dla tak ustalonych współrzędnych, w pętli `while..do` wywoływana jest procedura `Wstaw_znak`, która przygotowuje funkcję do wstawienia na formie w sposób matematyczny. W jej parametrze przekazywany jest kolejny znak funkcji zawarty w łańcuchu `fun5`. Ostateczną wartość współrzędnej pionowej dla funkcji ustala procedura `Korekcja_Y`. Na podstawie ostatecznych współrzędnych startowych, wstawiany jest przedrostek funkcji, a za nim sama funkcja.

Jeśli parametr `tryb` posiada wartość dwa, wstępna współrzędna pionowa jest powtórnie ustalana dla pochodnej, tym razem na podstawie wartości w zmiennej `y5max`, zawierającej największą współrzędną pionową funkcji, znajdującej się już na formie. Również w tym przypadku, ostateczną wartość współrzędnej pionowej dla pochodnej wyliczy procedura `Korekcja_Y`. Same umieszczenie na formie przedrostka oraz pochodnej przebiega w sposób analogiczny do sposobu wstawiania funkcji. Dodatkowo, między funkcją a pochodną dołączona została zielona kreska będąca elementem ozdobnym podpowiedzi.

```
procedure Z_Edukacji(const fun5,poch5:PChar;podp5:string;
                     const tryb:byte);
const tekst1='     Spróbuj znaleźć pochodną poniższej funkcji:';
      tekst2='f(x) = ';
      tekst3='f`(x) = ';
      czop5=10;
var en5,enp5:integer;
    ta5:char;
    y5min,y5max:integer;

    procedure Korekcja_Y;

begin
     AX:=10; - przypisz zmiennej modułu AX początkową wartość współrzędnej poziomej
     Canvas.Font.Height:=ws5; - na podstawie zmiennej ws5 ustaw wysokość...
                                czcionki
     if podp5<>'' then - czy dostarczony w parametrze łańcuch nie jest pusty?
        if FileExists(podp5) then - jeśli nie jest pusty, to czy plik...
                                    podany w parametrze istnieje?
        begin - jeśli plik istnieje, to...
             if tryb=0 then Image1.Top:=czop5 - jeśli parametr tryb...
                            posiada wartość zero to ustaw współrzędna...
                            pionową komponentu Image1, przypisując...
                            wartość tej współrzędnej ze stałej czop5
             else Image1.Top:=trunc(Form5.ClientHeight/2.5);
                  jeśli parametr tryb nie posiada wartości zero,...
                  wylicz współrzędną pionową komponentu...
                  Image1 równą ilorazowi wysokości...
                  obszaru roboczego formy i liczby 2,5
             if not Image1.Visible then Image1.Show; - jeśli...
                    komponent Image1 jest ukryty, odkryj go
             Image1.Picture.LoadFromFile(podp5) -załaduj obrazek....
              do komponentu Image1 z miejsca wskazanego w łańcuchu podp5
        end else - jeśli plik nie istnieje, to...
        begin
             Form9.Wynik9(25); - wywołaj procedurę obsługi komunikatów
             Form9.ShowModal - wyświetl modalnie formę komunikatów
        end
     else Image1.Hide; - jeśli łańcuch podp5 jest pusty, ukryj komponent Image1
     Refresh; - wyczyść formę z treści graficznych, wstawianych metodami Canvas
     if tryb>0 then - czy parametr tryb ma wartość większą od zera?
     begin - jeśli tak, to...
          Form5.Wyczysc; - uwolnij pamięć zajętą przez elementy edytora równań
          Canvas.Font.Height:=ws5; - na podstawie zmiennej ws5 ustaw...
                                     wysokość czcionki
          if tryb=1 then - czy parametr tryb ma wpisaną wartość jeden?
```

```
begin - jeśli tak, to...
     Canvas.Font.Color:=clGreen; - ustaw kolor czcionki na zielony

     Canvas.TextOut(5,czop5,tekst1) - wstaw na formie tekst...
                    zawarty w stałej tekst1, w miejscu wskazanym...
                    współrzędną poziomą równą 5 oraz pionową...
                    zapisaną w stałej czop5
end;
Canvas.Font.Color:=clBlack; - ustaw kolor czcionki na czarny
en5:=Canvas.TextWidth(tekst2); - zapamiętaj w zmiennej en5...
                         długość tekstu zawartego w stałej tekst2

AY:=2*czop5+ws5; - wylicz w zmiennej AY wstępną współrzędna pionową
AX:=5+en5; - wylicz w zmiennej AX współrzędną poziomą startową
en5:=0; - wyzeruj licznik iteracji pętli
enp5:=StrLen(fun5); - zlicz w zmiennej enp5 ilość znaków...
                      zawartych w łańcuchu fun5

while en5<enp5 do - wykonaj krok pętli, póki wartość w zmiennej...
                    en5 jest mniejsza od wartości w zmiennej enp5
begin
     ta5:=fun5[en5]; - wczytaj do zmiennej ta5 kolejny znak...
                       z łańcucha fun5

     Form5.Wstaw_znak(ta5); - wywołaj procedurę Wstaw_znak...
                 podając jej w parametrze znak sczytany w zmiennej ta5

     inc(en5) - zwiększ wartość w zmiennej en5 o jeden
end;
Korekcja_Y; - skoryguj pola współrzędnych pionowych elementów...
              edytora równań oraz wartość w zmiennej AY

Canvas.TextOut(5,AY,tekst2); - wstaw na formie przedrostek funkcji...
             zawarty w stałej tekst2 w miejscu o współrzędnej poziomej...
             równej 5 oraz pionowej zawartej w zmiennej AY

Form5.Wyswietl; - wstaw na formie funkcję w sposób matematyczny
if tryb=2 then - czy parametr tryb ma wartość równą dwa?
begin - jeśli tak, to...
     AY:=y5max+czop5; - wylicz wstępną współrzędną pionową...
             równą sumie maksymalnej współrzędnej położonej na...
             formie funkcji oraz odstępu zawartego w stałej czop5

     Canvas.Pen.Color:=clGreen; - ustaw kolor rysowanej linii...
                                  na zielony

     Canvas.Pen.Width:=2; - ustaw grubość linii na 2 piksele
     Canvas.MoveTo(20,AY); - ustaw początek rysowanej linii...
            przesunięty względem lewej krawędzi formy o 20 pikseli

     Canvas.LineTo(Form5.ClientWidth-20,AY); narysuj...
     linię poziomą od ustalonego początku do końca pomniejszonego o 20 pikseli...
     od prawej krawędzi formy

     AY:=AY+czop5; - skoryguj wstępną współrzędną pionową dla pochodnej
     Form5.Wyczysc; - uwolnij pamięć zajętą przez elementy edytora..
                      równań
     Canvas.Font.Height:=ws5; - na podstawie zmiennej ws5...
                                ustaw wysokość czcionki

     en5:=Canvas.TextWidth(tekst3); - zapamiętaj w…
      zmiennej en5 długość tekstu zawartego w stałej tekst3

     AX:=5+en5; - wylicz ponownie współrzędną poziomą startową
     en5:=0; - wyzeruj licznik iteracji pętli
```

```
                    enp5:=StrLen(poch5); - zlicz w zmiennej enp5 ilość znaków...
                                           pochodnej zawartej w łańcuchu poch5
                    while en5<enp5 do - wykonaj krok pętli, póki wartość w zmiennej...
                                        en5 jest mniejsza od wartości w zmiennej enp5
                    begin
                         ta5:=poch5[en5]; - wczytaj do zmiennej ta5 kolejny znak...
                                            z łańcucha poch5
                         Form5.Wstaw_znak(ta5); - wywołaj procedurę...
                                  Wstaw_znak podając w parametrze znak...
                                  sczytany w zmiennej ta5
                         inc(en5) - zwiększ o jeden wartość w zmiennej en5
                    end;
                    Korekcja_Y; - skoryguj pola współrzędnych pionowych elementów...
                                  edytora równań oraz wartość w zmiennej AY
                    Canvas.TextOut(5,AY,tekst3); - wstaw na formie przedrostek...
                                                   pochodnej, zawarty w stałej tekst3,...
                                                   w miejscu o współrzędnej poziomej...
                                                   równej 5 oraz pionowej zawartej w...
                                                   zmiennej AY
                    Form5.Wyswietl; - wstaw na formie pochodną w sposób matematyczny
               end
          end;
end;
```

VIII. 1.1. Ułożenie podpowiedzi na formie – procedura `Korekcja_Y`

Zadaniem procedury jest takie zmodyfikowanie wartości w polach współrzędnych pionowych należących do elementów edytora równań, by wstawiane na formie znaki funkcji czy pochodnej nie przysłaniały pozostałych elementów podpowiedzi. Ważne jest także, by funkcja czy pochodna zmieściły się na formie w całości. W pierwszej pętli `while..do` szukana jest najmniejsza i największa wartość współrzędnej pionowej spośród wszystkich elementów edytora. Szukanie minimum polega na porównywaniu wartości z każdego pola `y1` elementu edytora z poprzednio zapamiętaną wartością w zmiennej `y5min`. Jeśli jej wartość jest większa od obecnie sczytanej z pola `y1`, zmiennej tej przypisana zostaje nowa, mniejsza wartość współrzędnej z tego pola. Podobnie przebiega szukanie maksimum, z tą różnicą, że zmienna `y5max` porównywana jest z wartościami z pola `y2`.

Jeśli po opuszczeniu pętli, różnica między dotychczasową wartością współrzędnej startowej w zmiennej `AY` a wartością w zmiennej `y5min` jest różna od zera, uruchamiana jest kolejna pętla `while..do`, w której wszystkie pola współrzędnych pionowych elementów edytora równań zostają zmodyfikowane na podstawie wyliczonej różnicy. Również wartość w zmiennej `AY` zostaje poprawiona o tę różnicę sprawiając, że przedrostek funkcji czy pochodnej znajdzie się na formie, na właściwej wysokości.

```
procedure Korekcja_Y;
var as5:lisc;
    ky5:integer;
begin
     y5min:=maxint; - wpisz zmiennej y5min wartość startową równą...
                      maksymalnej wartości dla typu integer
```

```
        y5max:=-maxint; - wpisz zmiennej y5max wartość startową równą...
                          minimalnej wartości dla typu integer
        as5:=wsp; - do zmiennej as5 wpisz adres pierwszego elementu edytora równań
        while as5<>nil do - wykonaj krok pętli, póki wskaźnik as5...
                            nie ma adresu pustego
        begin
             if y5min>as5^.y1 then y5min:=as5^.y1; - jeśli wartość...
                        w zmiennej y5min jest większa od wartości w polu y1,...
                        wpisz do tej zmiennej wartość z tego pola
             if y5max<as5^.y2 then y5max:=as5^.y2; jeśli wartość ...
                        w zmiennej y5max jest mniejsza od wartości w polu y2,...
                        wpisz do tej zmiennej wartość z tego pola
             as5:=as5^.prawa - pozyskaj adres następnego elementu edytora
        end;
        ky5:=AY-y5min; - wylicz różnicę między wartością dotychczasową...
                współrzędnej pionowej a minimalną, pochodząca z pól edytora
        if ky5<>0 then - czy wyliczona różnica w zmiennej ky5 jest różna od zera?
        begin - jeśli tak, to...
             AY:=AY+ky5; - zmodyfikuj wartość w zmiennej AY o wyliczoną różnicę
             as5:=wsp; - do zmiennej as5 wpisz adres pierwszego elementu edytora
             while as5<>nil do - wykonaj krok pętli, póki wskaźnik as5...
                                 nie ma adresu pustego
             begin
                  with as5^ do - instrukcja wiążąca
                  begin
                       y1:=y1+ky5;
                       y2:=y2+ky5;   } skoryguj współrzędne pionowe
                       ly:=ly+ky5;   } elementów edytora o wyliczoną różnicę
                       py:=py+ky5
                  end;
                  as5:=as5^.prawa - pozyskaj adres następnego elementu edytora
             end;
             y5max:=y5max+ky5; - zmodyfikuj wartość w zmiennej y5max...
                                 o wyliczoną różnicę
        end;
end;
```

VIII. 2. Praca formy jako wyjściowej

Wywołanie procedury `Poczatek` z opcją `false` przygotowuje formę do pracy jako formę roboczą usytuowaną w lewym, dolnym rogu ekranu. Jej wygląd jest taki sam jak formy edukacji, z tą różnicą, że nie ma ona własnego menu oraz, jak było wcześniej zaznaczone, jest ona nieaktywna. By forma miała charakterystyczne obrzeża przy górnej i dolnej jej krawędzi, w procedurze `Poczatek` zostały zadeklarowane dwie zmienne typu `TStaticText`. Za ich pośrednictwem utworzone zostały dwa obiekty. Nadając im wysokość `Height` na poziomie 20 pikseli, właściwości `Align` pozycję odpowiednio `alTop` dla górnego obrzeża i `alBottom` dla dolnego i właściwości `BorderStyle` pozycję `sbsSunken`, uzyskujemy dwa obrzeża rozciągające się w poprzek formy. Kolor obrzeży jest oczywiście zgodny z kolorem formy i jest ustawiony na podstawie zmiennej globalnej `F3D` należącej do modułu regulacji. Parametry czcionki również ustawiane są na podstawie wartości ze zmiennych globalnych modułu regulacji. W pozyskaniu wartości regulacyjnych pośredniczą zmienne prywatne klasy `TForm5`.

```
procedure TForm5.Poczatek(czy_nau:boolean);
var belka1,belka2:TStaticText;
begin
     SZER5:=Form5.ClientWidth; - przypisz zmiennej prywatnej klasy TForm5...
                                 wartość szerokości roboczej formy

     if czy_nau then - czy parametr czy_nau posiada wartość true?
     begin
          . . .
     end else - jeśli nie, to...
         begin
              co5:=Form6.F2D; - przypisz zmiennej co5 wartość...
                                odpowiadająca kolorowi czcionki

              to5:=Form6.F3D; - przypisz zmiennej to5 wartość...
                                odpowiadająca kolorowi formy

              ws5:=Form6.RZ; - przypisz zmiennej ws5 wysokość czcionki w pikselach
              Canvas.Font.Name:=Form6.F1; - ustaw krój czcionki
              belka1:=TStaticText.Create(Form5); - utwórz w pamięci ...
                                                   element klasy TStacicText

              belka1.Parent:=Form5; - określ rodzica dla utworzonego elementu
              belka1.Color:=to5; - na podstawie zmiennej to5 ustaw kolor…
                                   elementu
              belka1.Height:=20; - określ wysokość elementu na poziomie…
                                   20 pikseli
              belka1.BorderStyle:=sbsSunken; - ustaw właściwość...
                                   BorderStyle w pozycji sbsSunken

              belka1.Align:=alTop; - ustaw właściwość Align w pozycji
                                     alTop...
                                ustawiając komponent w poprzek górnej krawędzi formy

              belka2:=TStaticText.Create(Form5); - utwórz w pamięci...
                                                   drugi element z klasy TStaticText

              belka2.Parent:=Form5; - określ rodzica dla utworzonego komponentu
              belka2.Color:=to5; - na podstawie zmiennej to5 ustaw kolor…
                                   elementu
              belka2.Height:=20; - określ wysokość elementu na poziomie…
                                   20 pikseli
              belka2.BorderStyle:=sbsSunken; - ustaw właściwość ...
                                   BorderStyle w pozycji sbsSunken

              belka2.Align:=alBottom - ustaw właściwość Align w pozycji ...
                alBottom ustawiając komponent w poprzek dolnej krawędzi formy
         end;
         . . . (ustawienie parametrów czcionki i koloru formy)
end;
```

Do współpracy z górną formą wejściową przygotowana została procedura `Tylko_wynik`, której zadaniem jest umieszczenie na formie pochodnej znalezionej przez program w sposób matematyczny. Procedura ta jest zatem wykorzystywana tylko po wybraniu z głównego menu formy powitalnej pozycji „pochodne”. Każdorazowo, po wywołaniu procedury, uwalniana jest pamięć zajmowana przez elementy edytora równań, zaś sama forma przygotowywana jest do wstawienia na niej kolejnej pochodnej, dostarczonej procedurze w parametrze w postaci łańcucha typu `PChar`. Przygotowanie formy polega na usunięciu z niej poprzedniej pochodnej, odświeżeniu parametrów czcionki oraz umieszczeniu na formie przedrostka ‘`f'(x)=`’ zawartego w stałej `tekst1`. Współrzędna pionowa zawarta w zmiennej `AY` jest wynikiem różnicy połowy wysokości obszaru roboczego formy i połowy wysoko-

ści czcionki, w wyniku czego zarówno przedrostek pochodnej, jak i sama pochodna umieszczone są pośrodku między górną a dolną krawędzią formy. Gdy forma jest już gotowa, w pętli `while..do` wywoływana jest procedura `Wstaw_znak`, której w parametrze podawany jest kolejny znak z łańcucha zawierającego pochodną. Podczas układania pochodnej przez edytor równań, po każdym wprowadzonym znaku modyfikowana jest zmienna prywatna `mx5` należąca do klasy `TForm5`, która zawiera maksymalną wartość pochodzącą z pól `px` współrzędnych poziomych elementów edytora. Jeżeli po opuszczeniu pętli, wartość ta przekracza początkową szerokość roboczą formy, jej szerokość zostaje zwiększona względem tej wartości, dzięki czemu długie pochodne wykraczające poza prawą krawędź formy mają szansę zmieścić się na formie w całości. Gdy edytor oraz forma są już przygotowane na wyświetlenie pochodnej, wywołana zostaje procedura `Wyswietl`, która umieszcza pochodną na formie.

```
procedure Tylko_wynik(const fun5:PChar);
const tekst1='f`(x) = ';
var x5:integer;
begin
     Form5.Wyczysc; - uwolnij pamięć zajmowaną przez elementy edytora
     Refresh; - wyczyść formę z treści graficznych, czyli z poprzedniej pochodnej
     Form5.Color:=to5; - ustaw kolor formy
     Canvas.Font.Height:=ws5; - ustaw wysokość czcionki
     Canvas.Font.Name:=Form6.F1; - ustaw krój czcionki
     Canvas.Font.Color:=co5; - ustaw kolor czcionki
     Canvas.Brush.Color:=to5; - ustaw kolor wypełnienia czcionki równy...
                                kolorowi tła formy
     Canvas.Brush.Style:=bsSolid; - ustaw styl czcionki bez: pogrubienia,...
                                    podkreślenia czy pochylenia
     AY:=trunc(Form5.ClientHeight/2)-trunc(Form6.RZ/2); - wylicz...
                                 współrzędną pionową dla wstawianej pochodnej
     Canvas.TextOut(AX,AY,tekst1); - umieść na formie przedrostek pochodnej...
       zawarty w stałej tekst1, w miejscu określonym wartościami w zmiennych AX i AY
     AX:=AX+Canvas.TextWidth(tekst1); - zwiększ wartość współrzędnej...
              poziomej w zmiennej AX o długość tekstu zawartego w stałej tekst1
     x5:=0; - wyzeruj licznik iteracji pętli
     while fun5[x5]<>#0 do - wykonaj krok pętli, póki znak w łańcuchu fun5,...
                             wskazany przez zmienną x5, nie jest pusty
     begin
          Form5.Wstaw_znak(fun5[x5]); - wywołaj procedurę Wstaw_znak...
                        podając jej w parametrze kolejny znak z łańcucha fun5
          inc(x5) - zwiększ o jeden wartość w zmiennej x5
     end;
     if (mx5+20)>SZER5 then Form5.Width:=mx5+20; - jeśli suma liczby 20...
                          i wartości w zmiennej mx5 jest większa od zapamiętanej...
                          w zmiennej SZER5 (szerokość podstawowa formy),...
                          zwiększ szerokość formy względem tej sumy
     Form5.Wyswietl - wstaw pochodną na formie
end;
```

IX. Moduł ustawień – Unit6

Moduł ten, wraz z formą regulacji należącą do klasy TForm6, umożliwiają indywidualne ustawienia tych wartości, które użytkownik uzna za konieczne. Są to ustawienia kolorów tła form roboczych, kroju czcionki, jej wielkości oraz koloru a także ustawień parametrów edytorów równań. Przejrzysty interfejs powinien ułatwić te regulacje. Otwarcie formy regulacji możliwe jest tylko z menu formy powitalnej, dlatego by móc wstępnie ocenić efekt ustawień, w lewym, górnym rogu formy regulacji znajduje się panel należący do klasy TPanel, a na nim komponent należący do klasy TStaticText z wpisanym tekstem statycznym „probka". Zarówno panel jak i napis zmieniają swoje właściwości w czasie regulacji, umożliwiając podgląd dokonanych regulacji jeszcze przed wyborem rodzaju ćwiczeń. Efektu ustawień dotyczących edytorów równań nie można w tym panelu zaobserwować, będzie on widoczny dopiero podczas wprowadzania funkcji czy pochodnej na formie roboczej. Dla pozostałej części programu, moduł ustawień jest swego rodzaju magazynem wartości, bowiem wszystkie ustawiane parametry są zapamiętane w zmiennych globalnych klasy TForm6, które są udostępnione dla pozostałych modułów.

Spośród wszystkich zdarzeń wykorzystanych w module od komponentów, jeden z nich jest najważniejszy, dlatego zostanie dokładniej opisany.

Procedura Optymalnie uruchamiana jest zaraz po utworzeniu formy regulacji w głównej części programu lub po użyciu przycisku osadzonego na formie regulacji z napisem „ustawienia optymalne". Uruchomienie procedury przypisuje wszystkim zmiennym globalnym klasy TForm6 wartości optymalne, pochodzące ze stałych tej procedury.

```
public
     { Public declarations }
     F1:string;
     F2,F2D,F3,F3D:TColor;
     KR,RZ,RZ1,RM,MK:integer;
     QR,QP:extended;
end;
. . .
procedure TForm6.Optymalnie;
const
      F1A='Arial'; - domyślny krój czcionki
      F2A=clBlack; - domyślny kolor czcionki
      F3A=clBtnFace; - domyślny kolor formy
      F4A=3; - domyślny odstęp znaków od kreski ułamkowej lub pierwiastkowej
      F5A=25; - domyślna wysokość czcionki w pikselach
      F6A=12; - minimalna wysokość czcionki, jaką mogą osiągnąć znaki należącej do...
                funkcji potęgującej
      F7A=1.7; - domyślny stopień zmniejszenia wysokości czcionki należącej do...
                 funkcji potęgującej lub podstawy logarytmu
      F8A=1.7; - domyślny stopień przesunięcia znaków w górę dla funkcji potęgującej...
                 względem pozycji ostatniego znaku funkcji potęgowanej...
                 lub w dół dla podstawy logarytmu, względem pozycji rdzenia funkcji log
      F9A=8; - domyślna, maksymalna odległość współrzędnej znaku od miejsca kliknięcia,...
               by przy tym znaku mógł pojawić się kursor
```

zmienne globalne należące do klasy TForm6

```
var a:byte;
    b:string;
begin
     ComboBox5.Items.AddStrings(Screen.Fonts); - załaduj do komponentu...
                                                 ComboBox5 listę czcionek zainstalowanych...
                                                 w systemie operacyjnym
     a:=ComboBox5.Items.IndexOf(F1A); - z załadowanej listy czcionek, zwróć do...
                                        zmiennej a pozycję czcionki wskazanej przez stałą F1A
     ComboBox5.Text:=ComboBox5.Items[a]; - w oknie komponentu ComboBox5...
           uwidocznij nazwę czcionki, znajdującej się na pozycji, wskazanej przez zmienną a
     b:=IntToStr(F4A); - dokonaj konwersji liczby zawartej w stałej F4A...
                         do łańcucha b typu string
     a:=ComboBox1.Items.IndexOf(b); - do zmiennej a zwróć pozycję z listy...
                      komponentu ComboBox1, wskazaną przez zawartość łańcucha b
     ComboBox1.Text:=ComboBox1.Items[a]; - w oknie komponentu ComboBox1...
                                           uwidocznij pozycję z listy komponentu,...
                                           wskazaną przez zmienną a
     b:=IntToStr(F6A);
     a:=ComboBox2.Items.IndexOf(b);
     ComboBox2.Text:=ComboBox2.Items[a];
     b:=FloatToStr(F7A);
     a:=ComboBox3.Items.IndexOf(b);
     ComboBox3.Text:=ComboBox3.Items[a];
     b:=FloatToStr(F8A);
     a:=ComboBox4.Items.IndexOf(b);
     ComboBox4.Text:=ComboBox4.Items[a];
     b:=IntToStr(F9A);
     a:=ComboBox6.Items.IndexOf(b);
     ComboBox6.Text:=ComboBox6.Items[a];
                                                 (jak wyżej, w
                                                 stosunku do
                                                 pozostałych
                                                 komponentów)
     ScrollBar1.Position:=F5A; - ustaw suwak komponentu ScrollBar1...
                                w pozycji...
                                odpowiedniej do wartości wskazanej w stałej F5A
     F1:=F1A;
     F2:=F2A;
     F2D:=F2A;
     F3:=F3A;
     F3D:=F3A;
     KR:=F4A;
     RZ:=F5A;
     RM:=F6A;
     QR:=F7A;
     QP:=F8A;
     MK:=F9A;
                                                 przypisz zmiennym glo-
                                                 balnym wartości do-
                                                 myślne, na podstawie
                                                 wartości stałych
     StaticText3.Font.Name:=F1A; - ustaw krój czcionki dla tekstu „probka”
     StaticText3.Font.Color:=F2A; - ustaw kolor czcionki dla tekst „probka”
     Panel1.Color:=F3A; - ustaw kolor panelu kontrolnego względem stałej F3A
     CheckBox1.Checked:=true - ustaw właściwość Checked elementu CheckBox1...
                               w pozycji true, uaktywniając ponaglanie użytkownika po...
                               upływie wyznaczonego czasu oczekiwania na odpowiedź
end;
```

Komponenty `ComboBox1..5`, wyszczególniane w kodzie powyższej procedury, są jedynie wskaźnikami na obiekty z klasy `TcomboBox`, które osadzone zostały na formie regu-

lacji podczas jej projektowania. Możliwość ukrycia pełnej listy wyboru w czasie, gdy wybór ten nie jest wykonywany przez użytkownika sprawiło, że właśnie te komponenty zostały wykorzystane jako elementy regulacyjne. Sam wybór wartości, polegający na rozwinięciu listy komponentu przez kliknięcie i wyborze pozycji z listy, również przez kliknięcie, przedstawia procedura zdarzenia kliknięcia w komponent podczas wyboru kroju czcionki.

```
procedure TForm6.ComboBox5Click(Sender: TObject);
var t1:string;
begin
     t1:=ComboBox5.Text; - w zmiennej t1 zapamiętaj wybraną wartość z listy
     StaticText3.Font.Name:=t1; - ustaw krój czcionki w komponencie...
                                  StaticText3 z wpisanym tekstem „probka”...
                                  na podstawie wartości w zmiennej t1
     F1:=t1 - przypisz zmiennej globalnej F1 nazwę wybranego kroju czcionki,...
              zapamiętanej w zmiennej t1
end;
```

W przypadku wybranych wartości reprezentujących liczby, przed przypisaniem jej zmiennej globalnej, należy poddać ją konwersji do wartości liczbowej, wykorzystując funkcje `StrToInt` lub `StrToFloat`.

Z kolei wykorzystanie suwaka jako elementu regulacyjnego wysokości czcionki, okazało się najbardziej trafne, z uwagi na wygodę jego użycia. Wybór wartości, za jego pośrednictwem, dokonywany jest w zakresie od 10 do 50 i nie wymaga konwersji, ponieważ komponent zwraca wartość liczbową. Minimalny krok zmiany wartości o jeden, uzyskany w wyniku przesuwania suwaka, uzyskany został poprzez przypisanie polu `LargeChange` opisywanego komponentu liczby jeden.

```
procedure TForm6.ScrollBar1Change(Sender: TObject);
begin
     RZ:=ScrollBar1.Position; - zmiennej globalnej przypisz wartość proporcjonalną...
                                do pozycji suwaka
     StaticText3.Font.Height:=RZ - ustaw wysokość czcionki w komponencie ...
                                   StaticText3 z wpisanym tekstem „probka”,...
                                   na podstawie wartości w zmiennej RZ
end;
```

Zmiany koloru formy lub czcionki dokonuje użytkownik, wykorzystując jeden z czterech przycisków osadzonych na formie regulacji – użycie jednego z nich uruchamia okno dialogowe `ColorDialog1`, które umożliwia wybór koloru. Zmianę koloru czcionki dla górnej formy roboczej przedstawia poniższa procedura zdarzenia, uruchamiana po kliknięciu w przycisk `Button1`.

```
procedure TForm6.Button1Click(Sender: TObject);
begin
     StaticText3.Font.Color:=F2; - ustaw kolor czcionki w komponencie...
                                   StaticText3, z wpisanym tekstem „probka”,...
                                   na podstawie dotychczasowej wartości w zmiennej F2
     if ColorDialog1.Execute then - czy okno dialogowe ColorDialog1...
                                    zostało otwarte?
     begin - jeśli tak, to...
          F2:=ColorDialog1.Color; - przypisz zmiennej globalnej F2 wartość koloru...
                                    wybranego z komponentu ColorDialog1
```

```
                StaticText3.Font.Color:=F2 - ustaw kolor czcionki w komponencie...
                                             StaticText3, na podstawie nowej wartości w zmiennej F2
      end;
end;
```

Zmiana koloru czcionki dla dolnej formy roboczej lub koloru samej formy odbywa się analogicznie do tej, zaprezentowanej w powyższej procedurze.

IX. 1. Obsługa pliku pomocy `pomoc.chm`

Plik pomocy stworzony został za pomocą darmowego programu HTMLHelp Workshop ver. 4.71.1015.0 firmy Microsoft Corporation. Wsad dla tego programu stanowią pliki w formacie HTML, które utworzone zostały w programie Microsoft Word.

Otwarcie pliku pomocy wymaga uruchomienia funkcji znajdującej się w bibliotece `hhctrl.ocx`, będącej częścią systemu operacyjnego Windows. Biblioteka ta zwykle znajduje się w katalogu `\Windows\System32\`. Poniżej znajdują się dwie wybrane wartości stałe, wykorzystywane do otwarcia pliku pomocy – są to liczby szesnastkowe, o czym informuje znak dolara przed liczbą.

```
const
  HH_DISPLAY_TOPIC=$0000; - podstawowa wartość otwarcia pliku pomocy
  HH_DISPLAY_TOC=$0001; - wartość otwarcia pliku pomocy z otwartą zakładką...
                          spisu treści
```

Wykorzystanie funkcji bibliotecznej wymagało zadeklarowania nagłówka funkcji, której parametry muszą być zgodne z parametrami funkcji `HtmlHelpA` zawartej we wspomnianej bibliotece. Dyrektywa `external` udostępnia wskazaną funkcję do wykorzystania i umożliwia wywołanie jej poprzez wyżej zadeklarowany nagłówek funkcji. Dyrektywa `stdcall`, to konwersja używana przez interfejs `API`, zapewniająca zgodność z innymi programami napisanymi pod system operacyjny Windows. Dyrektywa sprawia, że parametry funkcji odkładane na stosie, są zdejmowane ze stosu jeszcze przed zwróceniem wartości przez funkcję.

```
function HTMLHelp(hwndCaller:THandle; - przekazanie funkcji uchwytu aplikacji
                  pszFile:PChar; - ścieżka do pliku pomocy
                  uCommand:Cardinal; - polecenie do wykonania na pliku…
                                       pomocy
                  dwData:Longint):THandle;stdcall; - dodatkowe dane,...
                                                     zależne od polecenia

external 'hhctrl.ocx' name 'HtmlHelpA';
deklaracja nagłówka funkcji bibliotecznej obsługującej plik pomocy
```

Funkcja `HH` jest wywoływana w momencie wybrania z dostępnego menu polecenia wyświetlenia pliku pomocy. Została ona przypisana do aplikacji jako obsługa zdarzenia `OnHelp`. By funkcja mogła być wykorzystana do obsługi tego zdarzenia, musi być zgodna z typem `THelpEvent` zadeklarowanym w module `Classes`.

```
function TForm6.HH(Command:Word;
                   Data:Integer;
                   var CallHelp:Boolean):Boolean;;
begin
     if (Command=0) and (Data=0) then
         HTMLHelp(Application.Handle, - przekazanie funkcji uchwytu aplikacji
                  PChar(Application.HelpFile), - ponieważ
                           przekazywana...
                           ścieżka do pliku pomocy powinna być typu PChar..
                           została dokonana konwersja polegająca na przekazaniu..
                           do funkcji adresu komórki pamięci, w której rezyduje...
                           początek łańcucha typu string
                  HH_DISPLAY_TOC, - polecenie otwarcia pliku pomocy z otwartą...
                                    zakładką spisu treści
                  0); - brak dodatkowych danych
     CallHelp:=false;
     Result:=false
end;
```

X. Blok pochodnej funkcji – Unit7

Podstawowym zadaniem tego bloku jest znalezienie pochodnej i zastąpienie nią funkcji, którą główna procedura bloku otrzymuje w parametrze. Dodatkowym zadaniem bloku jest przygotowywanie podpowiedzi i kontrola błędów. W bloku zadeklarowano klasę o nazwie pochodna i jedną procedurę klasy blok_glowny, która jest jednocześnie główną procedurą bloku. Dostarczany procedurze parametr jest zmienną łańcuchową typu PChar zawierającą adres początku łańcucha znakowego dysponującego funkcją matematyczną. Deklarację klasy przedstawiono poniżej:

```
unit Unit7;

interface

uses SysUtils;
type
    . . .
    pochodna = class
    procedure blok_glowny(var cialo:PChar;wpis:boolean);

    private

    public
          . . .
          bl:boolean;
          blw:byte;
end;
```

Zmienną klasy jest poch1.

```
var poch1:pochodna;
```

W celu realizacji reguły łańcuchowej, polegającej na wyodrębnieniu z całej funkcji wszystkich funkcji elementarnych, zadeklarowano strukturę danych, której kopie alokowane są w pamięci komputera w sposób dynamiczny, a więc w trakcie wykonywania się programu. Jej budowa jest następująca:

```
type bloki=^kawalki;
    kawalki=record
                  t1:PChar;
                  t2:PChar;
                  t3:PChar;
                  t4:PChar;
                  jj:Integer;
                  fw:Boolean;
                  fn:Boolean;
                  i1:Integer;
                  poprz:bloki;
                  nast:bloki
    end;
```

Pola t1, t2, t3 i t4 są łańcuchami znakowymi typu PChar, a więc pamięć dla nich jest dodatkowo przydzielana dynamicznie o wielkości zależnej od przechowywanych w nich zawartości. Przeznaczenie tych pól jest ściśle określone w procesie rozkładu funkcji.

Pole t1 zawiera tą część funkcji, którą należy przeanalizować w całości na danym poziomie rozkładu. Nowy poziom rozkładu tworzony jest wtedy, gdy znalezienie pochodnej wymaga zaangażowania więcej niż tylko jednego poziomu rozkładu. Dotyczy to wyrazów łączonych znakami mnożenia, dzielenia czy potęgowania, ponieważ w każdym z tych trzech przypadków, na pochodną wynikową składają się zarówno wyrazy łączone tymi znakami akcji, jak i ich pochodne – bez dodatkowego poziomu analizy byłoby to niemożliwe. Podobnie jest z funkcjami trygonometrycznymi, których pochodna składa się zarówno z argumentu funkcji oraz jej pochodnej. Znakiem rozpoznawalnym do tworzenia nowego poziomu jest wtedy nawias otwierający.

Pole t2 jest podstawowym elementem rozkładu funkcji w ramach jednego poziomu. Na podstawie zawartości tego pola znajdowane są pochodne na danym poziomie rozkładu.

Pole t3 przewidziane jest na pochodne cząstkowe, znalezione w ramach jednego poziomu.

Pole t4 powinno zawierać pochodną ostateczną na danym poziomie rozkładu.

Praca głównej procedury bloku polega na analizie: wszystkich znaków zawartych w łańcuchu znakowym, który dostępny jest procedurze za pośrednictwem parametru cialo: uruchomienia wszystkich potrzebnych poziomów do analizy oraz powrocie do poziomu podstawowego i zwrócenia z pola t4 pochodnej pełnej funkcji.

Do obsługi struktury bloki wystarczą dwa wskaźniki:

```
var biez1,pom1:bloki;
```

Pierwszą czynnością głównej procedury bloku jest nadanie zmiennym wartości początkowych i utworzenie w pamięci pierwszego elementu bloki, zawierającego w swym polu t1 funkcję matematyczną, otrzymaną za pośrednictwem parametru cialo.

```
bl:=false; - zainicjuj zmienną błędu wartością false
luka:=false; - zmienna pomocnicza użyta przy odrzucaniu znaku spacji
dot_wpis:=false; } ustawienie zmiennych pomocniczych stosowanych...
zero:=false;     } w procesie tworzenia podpowiedzi
blw:=0; - wyzeruj zmienną kodu błędu
lp7:=0; - wyzeruj licznik podpowiedzi
p1:=0;  - wyzeruj licznik numeru poziomu zagłębienia
k4:=0;  - wyzeruj licznik nawiasów
Z:=nil; - dodatkową zmienną łańcuchową Z zainicjuj adresem pustym
if cialo<>nil then k1:=StrLen(cialo) else exit; - jeśli parametr cialo...
                          nie posiada adresu pustego, wylicz ilość niepustych znaków...
                          zawartych w łańcuchu, na który wskazuje ten parametr
new(biez1); - utwórz w pamięci pierwszy element bloki
GetMem(biez1^.t1,k1*sizeof(PChar)); - przydziel pamięć dla łańcucha t1..
           o wielkości odpowiedniej do liczby znaków zapamiętanych w zmiennej k1.
StrCopy(biez1^.t1,cialo); - skopiuj zawartość parametru cialo do łańcucha t1[1]
with biez1^ do - w instrukcji wiążącej uzupełnij pozostałe pola wartościami startowymi
begin
     t2:=nil;
```

[1] Pola t1, t2, t3 i t4 to wskaźniki na łańcuchy znaków typu PChar. Ponieważ łańcuchy lokowane w pamięci w sposób dynamiczny nie mają nazwy dlatego, by odnieść się do nich w tekście, posłużono się nazwami wskaźników wskazujących na konkretne łańcuchy znaków.

```
        t3:=nil;
        t4:=nil;
        jj:=0;
        fw:=false;
        fn:=false;
        i1:=0;
        poprz:=nil;   } jest to pierwszy element, zatem pola przechowujące adresy...
        nast:=nil     } poprzednika i następnika muszą być zainicjowane adresami pustymi
    end;
    k:=0; - wyzeruj zmienną przyjmującą wartości w kodzie ASCII odczytywanych znaków
    k1:=0; - wyzeruj zmienną wskazującą na miejsce w łańcuchu, z którego odczytywany będzie znak
```

Proces rozkładu funkcji odbywa się w pętli `while..do`, która jest główną pętlą procedury. Z każdym jej krokiem odczytywane są pojedyncze znaki z pola `t1`.

```
while biez1<>nil do - wykonaj krok pętli, póki zmienna biez1 nie ma adresu pustego
begin
     try - spodziewane wystąpienie błędu
        k:=ord(biez1^.t1[k1]); - odczytaj wartość znaku w kodzie ASCII z łańcucha t1...
                                 z miejsca wskazanego zmienną k1
     except - gdy wystąpi błąd...
           bl:=true;   }
           blw:=10;    } zbiór instrukcji po wystąpieniu błędu
           break       }
     end;
```

Może się zdarzyć, że pole `t1` będzie zainicjowane wartością pustą, co podczas próby odczytania znaku z nieistniejącego łańcucha spowoduje błąd systemu operacyjnego. By komunikat systemu się nie pojawił, polecenie odczytania znaku z tego pola zostało zamknięte w instrukcji `try..except..end`. Działa ona w ten sposób, że gdyby odczytanie znaku nie powiodło się, wówczas to ta instrukcja przejmie obsłużenie błędu, a nie system operacyjny. W części `except` znajdują się instrukcje, które mają się wykonać w sytuacji zaistnienia błędu, w tym przypadku jest to ustawienie zmiennej błędu w pozycji `true` oraz bezwzględne opuszczenie pętli instrukcją `break`, by nie dopuścić do kolejnych błędów i w konsekwencji zwieszenia się programu.

Kolejną instrukcją jest odrzucenie ewentualnych znaków pustych mogących pojawić się na początku analizowanego łańcucha.

```
if k=0 then - czy odczytany znak jest pusty?
   if not luka then - jeśli tak, to czy odczytane dotąd znaki są wyłącznie puste?
   begin - jeśli tak, to…
        inc(k1); - zwiększ wartość w zmiennej k1, by wskazywała na następny znak…
                   w łańcuchu
        continue - pomiń wszystkie następne instrukcje zawarte w pętli...
                   i rozpocznij pętlę od początku
   end;
luka:=true; - odczytany znak jest inny niż pusty. Od tej chwili odczytanie kolejnego znaku...
              pustego będzie oznaczało koniec łańcucha
```

Odczytany znak z pola `t1` musi zostać przepisany lub dopisany do pola `t2` – służy do tego procedura `wpisz`, która zaalokuje odpowiednią ilość pamięci dla niego, uwzględniając przy tym jego dotychczasową zawartość i nowy znak. Przepisywanie znaków z pola `t1` do `t2` jest podstawowym elementem rozkładu funkcji, gdyż to właśnie zawartość pola `t2` jest analizowana przez wewnętrzne procedury znajdujące pochodne funkcji. Procedura `wpisz`

otrzymuje w parametrze, poprzez zmienną `biez1`, element `bloki` oraz w zmiennej `k` kod ASCII odczytanego znaku.

```
wpisz(biez1,k); - wywołanie procedury wpisz
```

Po dopisaniu znaku do pola `t2`, następuje sprawdzenie, czy odczytany został znak potęgowania oraz czy potęgowanie to dotyczy funkcji trygonometrycznej z potęgą umieszczoną między rdzeniem funkcji a jej argumentem – jeśli tak, to znak potęgowania wraz z funkcją potęgującą muszą zostać przeniesione na koniec argumentu, gdyż pozostawienie potęgi przed argumentem funkcji spowoduje pojawienie się błędu polegającego na nierozpoznaniu funkcji.

```
if k=94 then - czy odczytany znak jest znakiem potęgowania?
   if czy_trygon(biez1,k1) then - jeśli tak, to czy jest to potęgowanie funkcji...
                  trygonometrycznej z potęgą umieszczoną przed argumentem funkcji?
   begin - jeśli tak, to...
        trygon(biez1,k1); - przenieś potęgę na koniec argumentu w polu t1
        k:=Pred(StrLen(biez1^.t2)); - odczytaj pozycję znaku poprzedzającego znak...
                                      potęgowania w polu t2
        biez1^.t2[k]:='('; - wpisz znak nawiasu w miejsce znaku potęgowania
        k:=ord(biez1^.t1[k1]) - teraz bieżącym, odczytanym znakiem z pola t1...
   end;                         jest znak nawiasu otwierającego
```

Kolejny zbiór instrukcji zawarty w pętli `while..do` wykonuje się wtedy, gdy odczytanym znakiem z pola `t1` jest jeden ze znaków działania, czyli: mnożenia, dodawania, odejmowania, dzielenia lub potęgowania, przy dodatkowym warunku, że znak ten nie jest pierwszym znakiem w tym polu. Po spełnieniu tych warunków następuje sprawdzenie wartości logicznej pola `fw` należącego do bieżącego elementu `bloki` i jeśli ma ono wartość `false`, co oznacza, że tymczasowa pochodna nie została jeszcze znaleziona, wywołana zostaje procedura `jeden`, znajdująca pochodne z takich przypadków jak: liczba lub znak e, π lub x. Jeżeli procedura po skończonej pracy nie zwróci błędu, w polu `t3` powinna znaleźć się pochodna ze wspomnianych przypadków funkcji. Ponieważ procedura `jeden` może być wywoływana wielokrotnie w ramach jednego poziomu, pole `t3` będzie zawierało pochodne również z poprzednich wywołań tejże procedury, a nie tylko z ostatniego, co oznacza, że tego pola nie można wykorzystać do przygotowywania podpowiedzi. W związku z tym, do procedury `jeden` zostaje dołączony dodatkowo łańcuch `Z`, do którego procedura zwróci pochodną tylko z bieżącego jej wywołania. Po wykorzystaniu łańcucha pomocniczego, pamięć przez niego zajmowana zostaje uwolniona.

```
if ((k=42) or (k=43) or (k=45) or (k=47) or (k=94)) and (k1>0) then
begin - jeśli odczytany jest jeden ze znaków: '*', '+', '– ', '/', '^' i nie jest to pierwszy znak, to...
     if not biez1^.fw then - czy pochodna tymczasowa nie została znaleziona?
     begin - jeśli nie została znaleziona, to...
          jeden(biez1^.t3,biez1^.t2,Z,tym); - znajdź pochodną i wpisz ją...
                                              do pola t3
          if not bl then - czy brak błędu?
          begin - jeśli nie ma błędu, to...
               . . .
               k3:=StrLen(Z); - z ilu znaków składa się łańcuch Z?
               FreeMem(Z,k3*sizeof(PChar)); - uwolnij pamięć zajmowaną...
                                              przez łańcuch Z
               Z:=nil - przypisz zmiennej łańcuchowej Z identyfikator adresu pustego
          end;
```

```
        end;
        . . .
```

Wykrycie znaku dodawania lub odejmowania uruchamia kolejny blok instrukcji, którego zadaniem jest przepisanie pochodnej tymczasowej z pola `t3` do `t4` przeznaczonego na pochodne ostateczne na danym poziomie analizy funkcji. Przepisywaniem zawartości wspomnianych pól zajmuje się procedura `przepisz`, która po przepisaniu uwalnia pamięć zajmowaną przez pole `t3` a po opuszczeniu procedury również i pole `t2`, pozostawiając w nim jedynie odczytany znak działania. Uwolnienie pamięci zajmowanej przez wspomniane pola – łańcuchy znaków, tuż po wykryciu znaku dodawania lub odejmowania świadczy o tym, że pola te nie będą już potrzebne do dalszej analizy, gdyż części funkcji znajdujących się za wykrytym znakiem działania będzie analizowana osobno, co jest zgodne z zasadami różniczkowania funkcji połączonych znakami dodawania lub odejmowania.

```
if ((k=42) or (k=43) or (k=45) or (k=47) or (k=94)) and (k1>0) then
begin
     . . .
    if not bl then
       if (k=43) or (k=45) then – czy odczytanym znakiem jest dodawanie...
                                   lub odejmowanie?
       begin – jeśli tak, to...
            przepisz(biez1); – przepisz zawartość pola t3 do t4
            k3:=StrLen(biez1^.t2); – ile znaków znajduje się w polu t2?
            FreeMem(biez1^.t2,k3*sizeof(PChar)); – uwolnij pamięć...
                                                   zajętą przez pole t2
            GetMem(Biez1^.t2,sizeof(PChar)); – przydziel pamięć dla pola...
                                               t2 o długości jednego znaku
            biez1^.t2[0]:=chr(k); – wpisz do pola t2 znak, którego wartość…
                                    znajduje się w zmiennej k
           biez1^.t2[1]:=#0; – zamknij pole– łańcuch dopisując do niego znak pusty
            . . .
            biez1^.fw:=false – zaznacz, że pochodna została znaleziona i przepisana...
       end else                do pola t4 jako ostateczna w bieżącym poziomie
    else break; – opuść pętli while..do, jeśli wystąpił błąd
end;
```

Gdy z pola `t1` zostanie odczytany znak pusty o kodzie ASCII równym zero, wykonywane są instrukcje, które uwalniają pamięć zajętą przez bieżący element `bloki` i jego pola łańcuchowe. Przedtem znalezioną pochodną trzeba zachować w dodatkowym łańcuchu, by można ją było wykorzystać w dalszych krokach programu. Jest to o tyle ważne, że element, który zawiera znalezioną pochodną na danym poziomie analizy będzie za chwilę usunięty z pamięci. Jeśli jego pole `fw` posiada wartość `true`, następuje przepisanie zawartości pól `t3` do `t4`, w przeciwnym przypadku wywoływana zostaje procedura `jeden`, która znalezioną pochodną od razu umieści w polu `t4` przeznaczonym na pochodne ostateczne na danym poziomie analizy. Ważną sprawą jest ustalenie pozycji, od której po zmianie elementu bieżącego `bloki` na poprzedni, rozpocznie się odczytywanie znaków z pola `t1`. Pozycja ta została zapamiętana w polu `i1` składnika listy, który za chwilę stanie się bieżącym.

```
if k=0 then – czy z pola t1 został odczytany znak pusty?
begin – jeśli tak, to...
     if biez1^.fw then przepisz(biez1) – jeśli pochodna tymczasowa została...
                                         znaleziona, przepisz ją z pola t3 do t4
```

```
else - w przeciwnym przypadku, gdy pochodna tymczasowa nie została znaleziona, to...
begin
     jeden(biez1^.t4,biez1^.t2,Z,tym); - znajdź pochodną i wpisz ją...
                                         do pola t4
     if not bl then - czy nie ma błędu?
     begin - jeśli nie, to…
          . . .
          k3:=StrLen(Z); - z ilu znaków składa się łańcuch tymczasowy Z?
          FreeMem(Z,k3*sizeof(PChar)); - zwolnij pamięć zajmowaną przez
Z
          Z:=nil - zainicjuj zmienną łańcuchową Z wartością pustą
     end;
end;
dec(p1); - zmniejsz o jeden numer poziomu analizy
biez1^.fw:=false;
if biez1^.t4<>nil then - czy pole t4 nie jest zainicjowane adresem pustym?
begin - jeśli nie, to...
     k3:=StrLen(biez1^.t4); - z ilu znaków składa się pole t4?
     GetMem(Z,k3*sizeof(PChar)); - przydziel pamięć dla łańcucha Z ...
                          o wielkości odpowiedniej do długości łańcucha t4
     StrCopy(Z,biez1^.t4); - skopiuj zawartość pola t4 do łańcucha Z
     FreeMem(biez1^.t4,k3*sizeof(PChar)) - zwolnij pamięć zajmowaną...
end;                                       przez pole t4
if biez1^.t3<>nil then - czy pole t3 zostało zainicjowane adresem pustym?
begin - jeśli nie, to…
     k3:=StrLen(biez1^.t3); - z ilu znaków składa się pole t3?
     FreeMem(biez1^.t3,k3*sizeof(PChar)) - uwolnij pamięć zajętą przez t3
end;
if biez1^.t2<>nil then - czy pole t2 zostało zainicjowane adresem pustym?
begin - jeśli nie, to…
     k3:=StrLen(biez1^.t2); - (jak wyżej)
     FreeMem(biez1^.t2,k3*sizeof(PChar)) - (jak wyżej)
end;
if biez1^.t1<>nil then - czy pole t1 zostało zainicjowane adresem pustym?
begin - jeśli nie, to…
     k3:=StrLen(biez1^.t1); - (jak wyżej)
     FreeMem(biez1^.t1,k3*sizeof(PChar)) - (jak wyżej)
end;
pom1:=biez1^.poprz; - pobierz adres poprzedniego elementu bloki
if pom1<>nil then - czy poprzedni element bloki istnieje?
begin - jeśli tak, to…
     pom1^.nast:=nil; - zaznacz, że adres następnego elementu bloki…
      nie istnieje – wpisz w pole nast. poprzedniego elementu bloki adres pusty
     k1:=pom1^.i1 - przypisz zmiennej k1 nr pozycji, od której pole t1 ...
                    należące do poprzednika, będzie odczytywane
end;
dispose(biez1); - usuń z pamięci bieżący element bloki
biez1:=pom1; - przypisz wskaźnikowi biez1 adres swojego poprzednika
```

Jeśli do zmiennej `pom1`, zamiast adresu poprzednika zostanie wczytany adres pusty, nastąpi opuszczenie pętli `while..do`. Oznacza to, że pochodna całej funkcji została znaleziona i zapamiętana w zmiennej tymczasowej `Z`.

Zidentyfikowanie w polu `t1` jednego ze znaków akcji, czyli mnożenia, dzielenia, potęgowania lub nawiasu otwierającego, spowoduje uruchomienie mechanizmu przygotowujące-

go następny – niższy poziom analizy funkcji. Ponieważ mechanizm ten jest inny dla nawiasu i inny dla znaków akcji, przypadki te zostaną opisane oddzielnie.

Dla nawiasu otwierającego, poza utworzeniem następnika `bloki` i uczynieniem go elementem bieżącym, w pętli `repeat..until` wydzielana jest ta część funkcji, która została zamknięta w parze nawiasów – szukany jest zatem nawias zamykający. Z każdym krokiem pętli zliczane są nawiasy otwierające i zamykające w ten sposób, że zidentyfikowanie nawiasu otwierającego zwiększa wartość w zmiennej `k4` o jeden, natomiast dla nawiasu zamykającego zmniejsza ją o jeden. W rezultacie, gdy zmienna ta osiągnie wartość zero (pierwotnie jest ona zainicjowana wartością jeden) pętla zostaje opuszczona. We wspomnianej pętli, z każdym jej krokiem wywoływana jest procedura `wpisz`, która oprócz adresu dotychczasowego elementu `bloki` i zmiennej `k3`, przechowującej odczytany w pętli znak przeznaczony do skopiowania, otrzymuje także adres następnika, który po opuszczeniu pętli stanie się elementem bieżącym. Nim to nastąpi, do pola `i1` dotychczasowego elementu `bloki` wpisana zostaje wartość ze zmiennej `k1`, która posłuży do wyznaczenia pozycji, od której będzie sczytywane pole `t1` po powrocie z nowego poziomu zagłębienia, tzn. gdy element ten stanie się ponownie bieżącym.

```
if k=40 then – czy odczytanym znakiem jest nawias otwierający?
begin – jeśli tak, to...
     pom1:=biez1^.nast; – odczytaj ewentualny adres następnika
     if pom1=nil then – czy następny poziom zagłębienia istnieje?
     begin – jeśli nie, to…
          new(pom1); – utwórz w pamięci nowy element bloki
          with pom1^ do
          begin
               t1:=nil;
               t2:=nil;      } zainicjowanie pól– zmiennych łańcuchowych…
               t3:=nil;      } wartością pustą jest obowiązkowe
               t4:=nil;
               jj:=0;
               fw:=false;
               fn:=false;
               i1:=0;
               poprz:=biez1; – związanie nowego elementu z dotychczasowym
               nast:=nil
          end;
          biez1^.nast:=pom1; – związanie dotychczasowego elementu z nowym
     end else
         begin
              bl:=true;
              blw:=1;      } istnienie następnego elementu przed utworzeniem...
              break        } nowego jest błędem
         end;
     k4:=1; – zainicjuj licznik nawiasów wartością początkową równą jeden
     inc(k1); – zwiększ o jeden wartość w zmiennej k1, by wskazywała na następny…
                znak w polu t1 za wykrytym nawiasem otwierającym
     inc(p1); – zwiększ o jeden numeru poziomu
     repeat
          k3:=ord(biez1^.t1[k1]); – odczytaj i zapamiętaj w zmiennej k3…
                wartość znaku z pola t1, z miejsca wskazanego zmienną k1
          if k3=40 then inc(k4); – dla nawiasu otwierającego zwiększ wartość...
                                   icznika k4 o jeden
          if k3=41 then dec(k4); – dla nawiasu zamykającego zmniejsz wartość...
                                   licznika k4 o jeden
```

```
                if (k4>0) and (k3<>0) then wpisz(biez1,k3,pom1)
                else break; - jeśli wartość w liczniku nawiasów jest wciąż większa od zera...
                  a odczytany znak nie jest pusty, dopisz go do pola t2 bieżącego…
                  i do t1 nowego elementu bloki, w przeciwnym razie opuść pętlę

                inc(k1); - zwiększ wartość w zmiennej sterującej pętlą o jeden
                if (k4<>0) and (k3=0) then - czy odczytano znak pusty…
                                                przy niezgodności nawiasów?
                begin - jeśli tak, to…
                      bl:=true; } znak pusty przy nierównej liczbie nawiasów jest błędem
                      blw:=3    }
                end
          until (k4=0) or (bl); - opuść pętlę przy równości nawiasów lub błędzie
          biez1^.i1:=k1; - zapamiętaj w polu i1 bieżącego elementu bloki, pozycję znaku...
                           w polu t1, poprzedzającego znak nawiasu zamykającego

          biez1:=biez1^.nast; - nowy element bloki jest teraz bieżącym
          k1:=-1 - ponieważ przed rozpoczęciem cyklu głównej pętli while..do jest zwiększenie...
                   tej zmiennej o jeden, trzeba ją zainicjować wartością początkową –1, by praca...
                   pętli rozpoczęła się od zerowej wartości tej zmiennej
end;
```

Po zidentyfikowaniu znaku akcji, czyli mnożenia, dzielenia lub potęgowania, uruchamiany jest blok instrukcji, który przygotowuje następny – niższy poziom rozkładu funkcji. Polega to, tak jak dla znaku nawiasu otwierającego, na utworzeniu w pamięci nowego składnika `bloki` oraz na prawidłowym rozkładzie funkcji, stosownym do wykrytego znaku i zgodnym z zasadami matematyki. Tuż przed rozkładem funkcji na nowym poziomie rozkładu a po utworzeniu w pamięci nowego elementu `bloki`, w polu `jj` dotychczasowego elementu, zostaje zapamiętana pozycja wykrytego znaku akcji znajdującego się w polu `t2`. Informacja ta wykorzystana zostanie przez procedurę `trzy` do rozdziału funkcji na tę przed i za wykrytym znakiem akcji. O konieczności wywołania procedury `trzy` będzie informować wartość `true` kolejnego pola `fn` elementu `bloki`. Rozkład funkcji odbywa się w pętli `while..do` – warunkiem jej opuszczenia jest odczytanie z pola `t1` znaku pustego. Również i w tej pętli zliczane są nawiasy w ten sposób, że odczytanie znaku nawiasu otwierającego zwiększa wartość w zmiennej `k4` o jeden, natomiast dla znaku nawiasu zamykającego zmniejsza ją o jeden. Zidentyfikowanie w pętli znaku mnożenia lub dzielenia przy równości nawiasów, czyli przy wartości zmiennej `k4` równej zero, pętla zostaje opuszczona instrukcją `break`. W tym przypadku zostaje zachowana hierarchia działań, polegająca na tym, że potęgowanie ma najwyższy priorytet jako działanie, dlatego znak ten nie został ujęty w warunku opuszczenia pętli. Z kolei odczytanie znaku dodawania lub odejmowania, również przy równości nawiasów oraz braku znaku mnożenia, dzielenia lub potęgowania bezpośrednio przed wykrytym znakiem akcji, skutkuje opuszczeniem pętli. Warunek ten pozwala na opuszczenie nawiasów zamykających np. funkcję potęgującą, rozpoczynającą się od znaku dodawania lub odejmowania. W opisywanej pętli, z chwilą odczytania znaku potęgowania zostaje wywołana funkcja `czy_trygon`, która sprawdza, czy potęgowanie dotyczy funkcji trygonometrycznej oraz czy funkcja potęgująca znajduje się przed argumentem funkcji – jeśli tak, to za pomocą procedury `trygon` funkcja potęgująca zostaje przeniesiona na koniec argumentu. Zaraz po opuszczeniu pętli `while..do`, bieżącym elementem `bloki` staje się nowy element utworzony w tym bloku instrukcji, co jest równoznaczne z uruchomieniem kolejnego poziomu rozkładu funkcji.

```
if (k=94) or (k=42) or (k=47) then
begin - gdy odczytanym znakiem jest potęgowanie, mnożenie lub dzielenie
      . . . (instrukcje utworzenia w pamięci nowego elementu bloki)
      k4:=0; - wyzerowanie licznika nawiasów
      biez1^.jj:=Pred(StrLen(biez1^.t2)); - zapamiętaj w polu jj pozycję znaku...
      akcji w łańcuchu t2, która równa jest liczbie znaków w łańcuchu pomniejszonej o jeden,...
      gdyż pozycje w łańcuchach liczone są od zera
      biez1^.fn:=true; - umieść informację w polu fn o potrzebie wywołania...
                         procedury trzy
      k1:=inc(k1); - zwiększ wartość w zmiennej k1 o jeden, by w pętli odczytać…
                     następny znak stojący za rozpoznanym znakiem akcji
      k3:=ord(biez1^.t1[k1]); - odczytaj wartość tego znaku
      while k3<>0 do - opuść pętlę po odczytaniu z pola t1 znaku pustego
      begin
           if k3=40 then inc(k4); - doliczanie nawiasów otwierających
           if k3=41 then dec(k4); - odliczanie nawiasów zamykających
           if k3=94 then - czy odczytanym znakiem jest potęgowanie?
              if czy_trygon(biez1,k1) then - czy jest to potęgowanie funkcji...
               trygonometrycznej, w której znak potęgowania znajduje się przed argumentem?
              begin - jeśli tak, to…
                   trygon(biez1,k1); - przenieś funkcję potęgującą na koniec...
                                       argumentu
                   k3:=ord(biez1^.t1[k1]); - odczytaj wartości pierwszego znaku...
                                             argumentu funkcji trygonometrycznej

                   if k3=40 then inc(k4) - jeśli znak jest nawiasem otwierającym,...
                                           zostaje on zliczony w liczniku nawiasów
              end;
              if k4=0 then - czy liczba nawiasów jest równa zero?
              begin - jeśli tak, to …
                   if (k3=42) or (k3=47) then break; - przerwij pętlę...
                                                       po odczytaniu znaku mnożenia lub dzielenia
                   if (k3=43) or (k3=45) then - czy odczytano znak dodawania...
                                                lub odejmowania?
                   begin - jeśli tak, to...
                        k4:=ord(biez1^.t1[k1-1]); - odczytaj wartość...
                                                    poprzedniego znaku
                        if (k4<>42) and (k4<>47) and (k4<>94) then
                    czy poprzedni znak jest różny od mnożeniem, dzieleniem czy potęgowaniem?
                        begin - jeśli tak, to...
                             k4:=0; - przywróć zerową wartość licznikowi nawiasów
                             break - opuść pętlę
                        end else k4:=0; - przywróć zerową wartość licznikowi...
                   end;                   nawiasów, by kontynuować pętlę
              end;
              wpisz(biez1,k3,pom1); - dopisz odczytany znak do pola t2...
              dotychczasowego elementu (biez1) i do pola t1 nowego elementu (pom1)
              if (k3=0) and (k4<>0) then - czy odczytano znak pusty…
                                           przy niezgodności nawiasów?
```

```
                begin - jeśli tak, to jest to błąd
                      bl:=true;
                      blw:=3;      } ustaw zmienną błędu bl w pozycji true i opuść pętlę
                      break
                end;
                inc(k1); - zwiększ wartość w zmiennej k1 o jeden
                k3:=ord(biez1^.t1[k1]) - odczytaj wartość następnego znaku
        end;
        biez1^.i1:=Pred(k1); - w polu i1 bieżącego elementu zapamiętaj wartość...
                             zmiennej k1, pomniejszoną o jeden, by wskazywała na pozycję...
                             ostatniego znaku funkcji znajdującej się za rozpoznanym znakiem akcji
        biez1:=biez1^.nast; - teraz nowy element jest bieżącym
        k1:=-1 - ponieważ przed rozpoczęciem cyklu głównej pętli while..do jest zwiększenie...
                 tej zmiennej o jeden, trzeba ją zainicjować wartością początkową –1,...
                 by praca pętli rozpoczęła się od zerowej wartości tej zmiennej
end;
```

Dalsza część kodu należącego do głównej pętli `while..do` wykona się tylko wtedy, gdy dotychczasowa praca pętli przebiegła bezbłędnie oraz gdy bieżący element `bloki` istnieje – jego brak mógłby zaistnieć podczas odczytania znaku pustego z pola `t1` należącego do pierwszego elementu `bloki`, który nie posiada swego poprzednika. Jeśli zatem element `bloki` istnieje a dotychczasowa praca głównej pętli przebiegła bezbłędnie, sprawdzona zostaje wartość logiczna pola `fn` bieżącego elementu – jeśli posiada ono wartość `true`, wywołana zostaje procedura `trzy`, która znajduje pochodną dwóch funkcji połączonych jednym ze znaków akcji. Po powrocie z procedury, polu `fw` przypisana zostaje wartość `true`, która wykryta w kolejnych instrukcjach pętli głównej spowoduje przepisanie pochodnej tymczasowej, znajdującej się w polu `t3`, do pola `t4` przeznaczonego na pochodne ostateczne na danym poziomie analizy. Bardzo ważną sprawą jest utrzymanie właściwej pozycji na sczytywanej funkcji z pola `t1`, dlatego kolejną instrukcją po opuszczeniu procedury jest zwiększenie wartości zmiennej `k1` o jeden, by po rozpoczęciu kolejnego cyklu pętli wskazywała na następny znak. Dla lepszego zrozumienia powyższej czynności, dobrze jest posłużyć się przykładem:

$X^{18}+$ – przed wywołaniem procedury `trzy`, zmienna `k1` posiada wartość 2, która wskazuje na ostatni znak funkcji potęgującej, czyli 8

$X^{18}+$ – po opuszczeniu procedury `trzy`, wartość w zmiennej `k1` zostaje zwiększona o jeden, więc wskazuje na znak + stojący za funkcją potęgującą

```
inc(k1); - zwiększ wartość w zmiennej sterującej pętlą o jeden
if biez1<>nil then - czy bieżący element istnieje?
   if not bl then - jeśli tak, to czy nie ma błędu?
   begin - jeśli warunki zostały spełnione, to…
         if biez1^.fn then - czy pole fn posiada wartość true?
         begin - jeśli tak, to...
               trzy(biez1,Z,tym); - wywołaj procedurę trzy, która znajduje…
               pochodną funkcji połączonych znakami mnożenia, dzielenia lub potęgowania
               . . .
               biez1^.fn:=false; - odznacz potrzebę wywołania procedury trzy
               biez1^.fw:=true; - zaznacz, że pochodna tymczasowa została...
                                  znaleziona
               continue - opuść następne instrukcje w pętli...
                          i rozpocznij nowy jej cykl
         end;
```

```
      . . .
end;
```

Jeśli w powyższym bloku instrukcji, pole `fn` posiada wartość `false`, sprawdzana jest wartość sczytanego w pętli znaku – jeśli jest nią liczba 41, co odpowiada znakowi nawiasu zamykającego, uruchamiany jest blok instrukcji związanych ze znalezieniem pochodnej funkcji trygonometrycznej. Ważnym elementem w tym bloku jest procedura `dwa` – jej wywołanie uzależnione jest od zerowej wartości licznika nawiasów (`k4`) i od obecności pochodnej argumentu funkcji znajdującej się w łańcuchu pomocniczym `Z` – adres pusty tego łańcucha jest błędem, który może wyniknąć np. z wprowadzenia przez użytkownika funkcji trygonometrycznej z pustymi nawiasami, czyli bez argumentu. Zaraz po opuszczeniu procedury `dwa`, pamięć zajmowana przez łańcuch `Z` zostaje uwolniona.

```
if biez1^.fn then
begin
     . . .
end;
if k=41 then - czy odczytanym znakiem jest nawias zamykający?
   if k4<>0 then - jeśli tak, to czy liczba nawiasów otwierających i zamykających...
                   nie jest równa?

   begin -jeśli nie jest równa, to...
        bl:=true;
        blw:=3;      } ustaw zmienną błędu bl w pozycji true i opuść pętlę główną
        break
   end else - gdy liczba nawiasów otwierających i zamykających jest zgodna, to…
       if Z<>nil then - czy łańcuch Z nie jest pusty?
       begin - jeśli łańcuch Z nie jest pusty, to...
            k3:=StrLen(Z); - z ilu znaków składa się łańcuch Z?
            (zawartość łańcucha po opuszczeniu procedury dwa może ulec zmianie...
            dlatego zliczenie znaków przeprowadza się przed jej wywołaniem)

            dwa(biez1,Z,tym); - wywołaj procedurę dwa
            FreeMem(Z,k3*sizeof(PChar)); - uwolnij pamięć zajmowaną przez...
                                           łańcuch Z o długości takiej, jaka była określona...
                                           przy przydzieleniu pamięci dla tego łańcucha

            Z:=nil; - przypisz zmiennej łańcuchowej Z adres pusty
            . . .
            biez1^.fw:=true; - zaznacz, że pochodna tymczasowa została znaleziona
       end else
           begin
                bl:=true;    } pusty łańcuch tymczasowy przed wywołaniem...
                blw:=3       } procedury dwa jest błędem
           end;
```

Ostatnią instrukcją w głównej pętli `while..do` jest sprawdzenie wartości zmiennej błędu `bl` – jeśli posiada ona wartość `true`, program opuszcza pętlę instrukcją `break`. Opuszczenie pętli z błędem uruchamia blok instrukcji, której celem jest uwolnienie pamięci zajmowanej zarówno przez wszystkie rezydujące w pamięci elementy `bloki`, jak i przez wszystkie te łańcuchy, które w dalszym ciągu zajmują pamięć komputera. W sytuacji błędu, nie można mieć pewności, czy bieżący element `bloki` jest ostatnim, dlatego w pierwszej kolejności należy uzyskać pewny adres ostatniego elementu – odbywa się to w pętli `while..do`.

```
pom1:=nil;
while biez1<>nil do - wykonaj krok pętli, póki wskaźnik biez1 nie jest pusty
begin
     pom1:=biez1; - zapamiętaj niepusty adres elementu bloki
     biez1:=biez1^.nast - pozyskaj adres następnika z jego własnego pola
end;
```

Po opuszczeniu pętli, w zmiennej `pom1`, znajduje się pewny adres ostatniego elementu. Teraz już bez obaw, że któryś z elementów zostanie pominięty, uruchamiana jest kolejna pętla `while..do`, w której pamięć zajmowana przez kolejne elementy `bloki` zostaje uwolniona, począwszy od ostatniego elementu a skończywszy na pierwszym, wraz z ich łańcuchami.

```
while pom1<>nil do - wykonaj krok pętli, póki wskaźnik pom1 nie jest pusty
begin
     if pom1^.t4<>nil then - czy pole t4 nie jest zainicjowane adresem pustym?
     begin - jeśli nie, to...
          k3:=StrLen(pom1^.t4); - z ilu znaków składa się pole t4?
          FreeMem(pom1^.t4,k3*sizeof(PChar)) - zwolnij pamięć zajmowaną...
     end;                                        przez pole t4
     if pom1^.t3<>nil then - czy pole t3 nie jest zainicjowane adresem pustym?
     begin - jeśli nie, to…
          k3:=StrLen(pom1^.t3);                  } (jak wyżej)
          FreeMem(pom1^.t3,k3*sizeof(PChar))
     end;
     if pom1^.t2<>nil then - czy pole t2 nie jest zainicjowane adresem pustym?
     begin - jeśli nie, to…
          k3:=StrLen(pom1^.t2);                  } (jak wyżej)
          FreeMem(pom1^.t2,k3*sizeof(PChar))
     end;
     if pom1^.t1<>nil then - czy pole t1 nie jest zainicjowane adresem pustym?
     begin - jeśli nie, to…
          k3:=StrLen(pom1^.t1);                  } (jak wyżej)
          FreeMem(pom1^.t1,k3*sizeof(PChar))
     end;
     biez1:=pom1^.poprz; - zapamiętaj adres poprzednika
     dispose(pom1); - uwolnij pamięć zajmowaną przez dotychczasowy element bloki
     pom1:=biez1 - wskaźnikowi pom1 przypisz adres poprzednika
end;
if Z<>nil then - czy łańcuch pomocniczy nie jest pusty?
begin - jeśli nie jest pusty, to...
     k3:=StrLen(Z);                             } (jak wyżej)
     FreeMem(Z,k3*sizeof(PChar))
end;
```

Gdy po opuszczeniu głównej pętli `while..do`, zmienna błędu ma wartość `false`, pamięć zajmowana przez wszystkie elementy `bloki` została już uwolniona (odbywało się to sukcesywnie po wykrywaniu w głównej pętli znaku pustego), dlatego w tej sytuacji pozostaje tylko odczytanie liczby znaków, z których składa się dostarczony parametr `cialo` zawierający pełną funkcję, zwolnienie pamięci zajmowanej przez ten łańcuch oraz przypisanie mu adresu łańcucha pomocniczego Z, który powinien zawierać znalezioną, pełną pochodną.

```
k1:=StrLen(cialo); - z ilu znaków składa się łańcuch wyjściowy cialo?
FreeMem(cialo,k1*sizeof(PChar)); - zwolnij pamięci zajmowaną przez ten łańcuch
cialo:=Z; - przypisz łańcuchowi wyjściowemu cialo adresu łańcucha pomocniczego Z
```

Opisany sposób rozkładu funkcji na mniejsze jej części i analiza tych części na poszczególnych poziomach rozkładu ma tę cenną zaletę, że umożliwia pracę z funkcjami o teoretycznie nieograniczonym stopniu skomplikowania – takie było założenie zastosowane podczas projektowania aplikacji. W tym przypadku, jedynym ograniczeniem jest ilość wolnej przestrzeni adresowej pamięci komputera. Gdyby na pewnym etapie pracy aplikacji zabrakło wymaganej ilości pamięci, odpowiedniemu wskaźnikowi zostałby przypisany identyfikator adresu pustego, który wykryty w instrukcji warunkowej, spowodowałby przerwanie pracy programu – jest to mało prawdopodobne, ale możliwe. Nie zmienia to faktu, że mimo wielu prób z różnymi funkcjami, praca modułu przebiega bezbłędnie a pochodne znajdowane w ten sposób są poprawne.

X. 1. Czy funkcja jest trygonometryczna? Funkcja `czy_trygon`

Jest to dość prosta funkcja, której zadaniem jest zbadanie, czy przed znakiem potęgowania znajduje się funkcja trygonometryczna. Opisywana funkcja otrzymuje w parametrze adres bieżącego elementu `bloki` oraz zmienną, której wartość wskazuje na pozycję znaku potęgowania w łańcuchu. W pierwszej kolejności sprawdzana jest wartość owej zmiennej, czy jest większa od jedności (rdzeń funkcji trygonometrycznej składa się przynajmniej z dwóch znaków) – jeśli warunek został spełniony, w kolejnych instrukcjach wyboru `case` sprawdzane są dwa poprzednie znaki stojące przed znakiem potęgowania:

- pierwszy człon instrukcji identyfikuje funkcje kończące się na ‘in’, którymi mogą być: sin, arcsin – wyklucza się tym samym funkcję ‘ln’;
- drugi człon identyfikuje funkcje kończące się na ‘tg’, którymi mogą być: tg, ctg, arctg, arcctg – wyklucza się tym samym funkcję ‘log’;
- trzeci człon sprawdza tylko ostatni znak ‘s’, identyfikując przez to wszystkie funkcje kończące się na tą literę, czyli: cos, arccos.

Spełnienie jednego z powyższych warunków sprowadza się do zwrócenia przez funkcję wartości `true` – w przeciwnym przypadku zwraca ona wartość `false`. W tym przypadku nie jest potrzebne dokładne badanie funkcji, gdyż to zadanie zostało powierzone procedurze `dwa`, ma ona tylko wykluczyć potrzebę przeniesienia funkcji potęgującej, gdy przed znakiem potęgowania znajduje się np. nawias zamykający czy zmienna x.

```
function czy_trygon(const la1:bloki;const mi:Integer):boolean;
var k1:Integer;
begin
     Result:=false; - domyślne zwrócenie przez funkcję wartości false
     if mi>1 then - czy liczba znaków w łańcuchu jest większa niż jeden?
     begin - jeśli tak, to...
          k1:=ord(la1^.t1[mi-1]); - odczytaj wartość znaku stojącego przed...
                                    znakiem potęgowania
          case k1 of
               110: - jeśli poprzedni znak ma wartość litery ‘n’ to...
               begin
                    k1:=Ord(la1^.t1[mi-2]); - odczytaj znak przed literą ‘n’
```

```
                    if k1=105 then Result:=true - jeśli jest to wartość...
                                                  znaku 'i', to jest to funkcja trygonometryczna
                end;
                103: - jeśli poprzedni znak ma wartość litery 'g', to...
                begin
                    k1:=Ord(la1^.t1[mi-2]); - odczytaj znak przed literą 'g'
                    if k1=116 then Result:=true - jeśli jest to wartość...
                                                  znaku 't', to jest to funkcja trygonometryczna
                end;
                115:Result:=true; - jeśli poprzedni znak ma wartość litery 's',...
            end;                                to jest to funkcja trygonometryczna

        end;
    end;
```

X. 2. Przesunięcie potęgi na koniec argumentu – procedura `trygon`

Zadaniem tej procedury jest przeniesienie funkcji potęgującej na koniec argumentu funkcji trygonometrycznej. Wywołanie jej jest uzależnione od pozytywnego testu pochodzącego od opisanej wyżej funkcji `czy_trygon`. Pierwsza instrukcja procedury to przypisanie lokalnej zmiennej adresu łańcucha zawartego w polu `t1`, ale nie od jego początku lecz od znaku potęgowania, którego pozycję wskazuje parametr `mi`. Kolejna instrukcja `StrScan` wyszukuje w lokalnym łańcuchu pierwszego wystąpienia znaku nawiasu otwierającego i jeśli go znajdzie, kolejnej zmiennej lokalnej przypisze adres łańcucha rozpoczynającego się od tego nawiasu. Gdyby wyszukanie nawiasu się nie powiodło, wówczas lokalnej zmiennej zostanie przypisany adres pusty. Gdy nawias został znaleziony, wywołana zostaje lokalna funkcja `pozycja`, której zadaniem jest zwrócenie liczby znaków, z których składa się argument funkcji trygonometrycznej. Licząc nawias otwierający, zamykający i przynajmniej jeden znak argumentu, funkcja powinna zwrócić wartość przynajmniej równą trzy – mniejsza wartość będzie błędem. Kolejne trzy instrukcje mają za zadanie określić liczbę znaków, z których składa się funkcja potęgująca, którą należy przenieść. Liczba ta równa jest różnicy ilości znaków w łańcuchu lokalnym `w2` rozpoczynającym się od znaku potęgowania i ilości znaków w łańcuchu lokalnym `w3`, rozpoczynającym się od nawiasu otwierającego. W tym przypadku ilość ta musi być większa od jedności, ponieważ funkcja potęgująca nie może składać się tylko ze znaku potęgowania. Gdy warunek został spełniony, kolejnej zmiennej lokalnej, będącej jednocześnie łańcuchem wynikowym procedury, zostaje przydzielona pamięć o rozmiarze równym łańcuchowi zawartemu w polu `t1`, należącemu do elementu bieżącego `bloki`. Dzięki następnej funkcji `StrLCopy`, z pola `t1` do łańcucha wynikowego `w4` zostaje skopiowana liczba znaków wskazana zmienną `mi`, dzięki czemu, skopiowany zostaje rdzeń funkcji trygonometrycznej i ewentualnie wcześniejsze znaki. Zmienna `mi` dostarczona w parametrze wskazuje co prawda na znak potęgowania w polu `t1`, ale z uwagi, że pozycję w łańcuchach liczy się od zera, wartość ta jest równa liczbie znaków, licząc od początku łańcucha do znaku stojącego przed znakiem potęgowania, dlatego został skopiowany rdzeń funkcji bez znaku potęgowania. Kolejna instrukcja `StrLCat` to dopisanie argumentu funkcji znajdującego się w łańcuchu lokalnym `w3` do łańcucha wynikowego. By instrukcja dopisała pożądaną ilość znaków, w trzecim jej parametrze należy podać sumaryczną ilość znaków po skopiowaniu, czyli należy zsumować ilość znaków już znajdującą się w łańcuchu wynikowym i ilość znaków przeznaczonych do skopiowania. W podobny sposób, do łańcucha wynikowego dopisana zostaje funkcja potęgująca. Łańcuch wynikowy `w4` ma już komplet potęgowanej funkcji try-

gonometrycznej, ale za tą funkcją mogą znajdować się jeszcze dalsze znaki, dlatego by to sprawdzić, do zmiennej lokalnej `w3` przypisany zostaje adres łańcucha rozpoczynającego się od znaku spoza tej funkcji trygonometrycznej – jeśli łańcuch nie jest pusty, jego zawartość zostaje dopisana do łańcucha wynikowego. Zwolnienie pamięci zajmowanej przez pole `t1`, należące do otrzymanego w parametrze elementu `bloki` i przypisanie mu adresu łańcucha wynikowego `w4`, kończy pracę procedury.

```
procedure trygon(const lan2:bloki;const mi:Integer);
var w2,w3,w4:PChar;
    kx1,kx2,kx3,kp1,kpm:integer;

    function pozycja(const lancuch:PChar):integer;

begin
     w2:=@lan2^.t1[mi]; - przypisz łańcuchowi w2 adres łańcucha zawartego w polu...
                          t1 rozpoczynającego się od znaku potęgowania

     w3:=StrScan(w2,'('); - przypisz łańcuchowi w3 adres łańcucha w2…
                            rozpoczynającego się od nawiasu otwierającego

     if w3<>nil then - czy łańcuch w3 nie posiada adresu pustego?
     begin - jeśli nie, to...
          kx1:=pozycja(w3); - z ilu znaków składa się argument funkcji?
          if kx1>2 then - czy ilość znaków jest większa niż dwa?
          begin - jeśli tak, to...
               kx2:=StrLen(w2); - z ilu znaków składa się łańcuch w2?
               kx3:=StrLen(w3); - z ilu znaków składa się łańcuch w3?
               kx3:=kx2-kx3; - różnica dwóch powyższych wartości jest ilością...
                               znaków z których składa się funkcja potęgująca

               if kx3>1 then - czy ilość znaków funkcji potęgującej...
                             jest większa od 1?

               begin - jeśli tak, to...
                    kp1:=StrLen(lan2^.t1); - z ilu znaków składa się..
                                             pole t1?

                    GetMem(w4,kp1*sizeof(PChar)); - przydziel pamięć dla...
                      łańcucha lokalnego o identycznej długości, co łańcuch w polu t1...
                      elementu bloki otrzymanego w parametrze

                    StrLCopy(w4,lan2^.t1,mi); - skopiuj zawartość pola...
                                                t1 do łańcucha w4, od jego początku...
                                                do ostatniego znaku rdzenia funkcji

                    kpm:=StrLen(w4); - z ilu znaków składa się łańcuch
                                       lokalny w4?

                    kpm:=kpm+kx1; - z ilu znaków będzie składał się łańcuch po...
                                    dodaniu do niego znaków argumentu funkcji?

                    StrLCat(w4,w3,kpm); - dopisz do łańcucha w4 znaki ...
                                          argumentu funkcji, z łańcucha w3

                    kpm:=StrLen(w4); - z ilu znaków składa się teraz łańcuch w4?
                    kpm:=kpm+kx3; - z ilu znaków będzie się składał łańcuch w4,..
                                    po dodaniu do niego znaków funkcji potęgującej?

                    StrLCat(w4,w2,kpm); - dopisz do łańcucha w4 znaki…
                                          funkcji potęgującej, z łańcucha w2
```

```
                    w3:=@lan2^.t1[kpm]; - przypisz łańcuchowi w3 adres...
                                          zawarty w polu t1, rozpoczynającego się od...
                                          znaku za funkcją trygonometryczną
                    if w3<>'' then StrCat(w4,w3); - czy zostało coś...
                                                    jeszcze do skopiowania z pola t1?
                                                    Jeśli tak, to dopisz do łańcucha w4…
                                                    zawartość łańcucha w3
                    FreeMem(lan2^.t1,kp1*sizeof(PChar)); - zwolnij…
                                                           pamięć zajętą przez pole t1 i...
                    lan2^.t1:=w4 - przypisz mu adres łańcucha lokalnego w4
                end;
            end else - gdy ilość znaków argumentu funkcji jest mniejsza niż 3 – jest to błąd
                begin
                    bl:=true;
                    blw:=10
                end
        end else - gdy nie znaleziono nawiasu otwierającego – jest to błąd
            begin
                bl:=true;
                blw:=3
            end;
end;
```

X. 2.1. Z ilu znaków składa się argument funkcji? Funkcja `pozycja`

Funkcja ta ma za zadanie zwrócenie wartości odpowiadającej ilości znaków zawartych w dostarczonym łańcuchu, składającym się na argument funkcji trygonometrycznej. Zliczanie znaków odbywa się w pętli `while..do`, w której odbywa się identyfikacja znaków, poczynając od znaku stojącego za znakiem nawiasu otwierającego, rozpoczynającego argument funkcji. Licznik nawiasów, czyli zmienna `n3`, zainicjowana została wartością jeden, uwzględniając pierwszy nawias otwierający argumentu. Z każdym krokiem pętli identyfikowany jest kolejny znak jako wartość w kodzie ASCII w ten sposób, że dla znaku nawiasu otwierającego licznik zwiększa swoją wartość o jeden, natomiast dla nawiasu zamykającego zmniejsza ją o jeden. Opuszczenie pętli nastąpi z chwilą zidentyfikowania znaku pustego o wartości w kodzie ASCII równej zero lub gdy licznik nawiasów osiągnie wartość zero. Jeśli po opuszczeniu pętli, wartość ostatnio odczytanego znaku będzie różna od zera, funkcja zwróci ilość iteracji pętli powiększoną o jeden, co odpowiada ilości znaków argumentu, w przeciwnym przypadku funkcja zwróci wartość zero, która wykryta w procedurze `trygon` uznana zostanie za błąd.

```
function pozycja(const lancuch:PChar):integer;
var n3,i3,k3:integer;
begin
     i3:=1; - ustaw wartość początkową licznika iteracji pętli
     k3:=ord(lancuch[i3]); - odczytaj wartość następnego znaku
     n3:=1; - ustaw wartość początkową licznika nawiasów
     while k3<>0 do - wykonaj krok pętli, póki wartość znaku jest różna od zera
     begin
          if k3=40 then inc(n3);- dla nawiasu otwierającego zwiększ wartość n3
          if k3=41 then dec(n3);- dla nawiasu zamykającego zmniejsz…
                                  wartość n3
          if n3=0 then break; - opuść pętlę, gdy licznik nawiasów będzie…
                                równy 0
```

```
            inc(i3); - zwiększ wartość o jeden licznika iteracji
            k3:=ord(lancuch[i3]) - odczytaj wartość następnego znaku
        end;
        if k3<>0 then Result:=Succ(i3) else Result:=0
        gdy wartość ostatnio odczytanego znaku jest różna od zera, zwróć...
        powiększoną o jeden ilość iteracji, w przeciwnym przypadku zwróć wartość zero
end;
```

X. 3. Pochodna z liczby lub zmiennej – procedura `jeden`

Procedura otrzymuje w parametrach trzy łańcuchy typu `PChar` i zmienną `slownik`, która wykorzystana będzie podczas przygotowywania podpowiedzi. Łańcuch wejściowy `la2`, przeznaczony jest tylko do odczytu, dlatego przed jego nazwą znajduje się literał `const`. Gdy w łańcuchu tym zostanie bezbłędnie rozpoznany jeden z następujących przypadków funkcji:

- liczba całkowita lub w postaci ułamka dziesiętnego;
- stałe typu ‘e’ lub ‘π’;
- zmienna jako znak ‘x’;

do łańcuchów wynikowych `la1` i `la3` trafią znaki odpowiadające pochodnej rozpoznanego przypadku – literały `var` przed nazwami łańcuchów zezwalają na ich modyfikację. Parametr `slownik` jest typu znakowego, dlatego wstępnie został zainicjowany znakiem pustym. Procedura posiada dwa dwuznakowe łańcuchy lokalne typu `Char` – pierwszy z nich (`tym1`) przeznaczony jest do zapamiętania znaku działania poprzedzającego rozpoznawaną funkcję, zaś drugi (`tym2`) do wstępnego zapamiętania pochodnej wynikowej. Oba łańcuchy przed użyciem zostają „wyczyszczone” z przypadkowych znaków – odbywa się to w pętli `for`. Rozpoznawanie znaków funkcji odbywa się w pętli `repeat..until`, której opuszczenie nastąpi po rozpoznaniu znaku pustego. Pierwsza instrukcja pętli to odczytanie wartości znaku z łańcucha `la2`, z miejsca wskazanego zmienną sterującą `i`. Kolejne dwa człony instrukcji warunkowych mają za zadanie zapamiętanie w łańcuchu lokalnym znaku działania oraz zareagowanie błędem, gdyby następnym znakiem za wcześniej rozpoznanym znakiem działania okazał się kolejny znak działania. Pomocną okazała się tu zmienna logiczna `s`, która przyjmuje wartość logiczną `true` po rozpoznaniu pierwszego znaku działania. W kolejnym członie instrukcji rozpoznawane są znaki cyfr oraz kropki i przecinka będące separatorem liczby dziesiętnej. W tym miejscu trzeba zaznaczyć, że póki program nie będzie liczył wartości z wprowadzonej liczby dziesiętnej, obydwa separatory są poprawne. Poza pierwszym rozpoznanym znakiem cyfry, kolejne znaki cyfr sprawdzane są już w wewnętrznej pętli członu `while..do`, porównując je z tablicą stałych o nazwie `cyfry`. Pętla ta wykorzystuje tę samą zmienną sterującą, co pętla `repeat..until`, dzięki czemu, po opuszczeniu pętli `while..do` zachowana zostaje właściwa pozycja w analizowanym łańcuchu. Kolejne trzy człony instrukcji warunkowych rozpoznają odpowiednio: znak litery ‘π’, ‘e’ oraz ‘x’. Po rozpoznaniu tych znaków (jak też znaków cyfr), do łańcucha `tym2` wpisany zostaje znak cyfry zero, z wyjątkiem znaku ‘x’, po rozpoznaniu którego wpisany zostaje znak cyfry jeden. Poza uzupełnieniem łańcucha znakami pochodnej oraz parametru `słownik` stosownym znakiem, będącym kodem rozpoznanej funkcji, kolejnej zmiennej logicznej `st` wpisana zostaje wartość `true`, oznaczającej poprawne rozpoznanie funkcji. Poza zwiększeniem o jeden wartości w zmiennej sterującej pętlą, znajduje się jeszcze instrukcja zabezpieczająca polegająca na opuszczeniu pętli w sytuacji, gdyby wartość odczytanego znaku okazała się nieodpowiednia.

Jest to o tyle ważne, że program nie dopuści do analizy następnego znaku, mimo że procedura mogłaby znaleźć jego pochodną.

```
procedure jeden(var la1:PChar;const la2:PChar;var la3:PChar;
var slownik:char);
const cyfry=[44,46,48,49,50,51,52,53,54,55,56,57]; - tablica stałych...
                                zawierająca wartości znaków cyfr, kropki i przecinka
var i,k:integer;
    s,st:boolean;
    tym1,tym2:array[0..1] of Char;
    pom1:PChar;
begin
     slownik:=#0; - wstępne zainicjuj parametr slownik wartością pustą
     for i:=0 to 1 do
     begin
          tym1[i]:=#0; } wyczyść łańcuchy lokalne z tzw. „śmieci”
          tym2[i]:=#0
     end;
     i:=0;
     s:=false;
     st:=false;
     repeat
           k:=ord(la2[i]); - odczytaj wartość znaku wskazanego zmienną i
           if ((k=42) or (k=43) or (k=45) or (k=47) or (k=94)) then
       czy odczytany znak to mnożenie, dodawanie, odejmowanie, dzielenie czy potęgowanie?
              if s then jeśli tak, to czy znak działania jest powtórzony?
              begin
                   blw:=12;
                   bl:=true;   } jeśli tak, to opuść pętlę z błędem
                   break
              end;
           if ((k=42) or (k=43) or (k=45) or (k=47) or (k=94))
           and (i=0) then - czy odczytany, pierwszy znak to mnożenie,...
                     dodawanie, odejmowanie, dzielenie czy potęgowanie?
           begin jeśli tak, to...
                tym1[0]:=Chr(k); - zapamiętaj ten znak w łańcuchu tym1
                s:=true - ustaw zmienną s, by zareagować na ewentualne…
                          powtórzenie znaku
           end;
           if k in cyfry then - czy odczytany znak należy do cyfr?
              if not st then - czy pochodna nie została jeszcze znaleziona?
              begin - jeśli nie, to...
                   inc(i); - zwiększ wartość w zmiennej sterującej pętlą o jeden
                   k:=ord(la2[i]); - odczytaj wartość następnego znaku
                   while k in cyfry do - wykonaj krok pętli, póki znak jest cyfrą
                   begin
                        inc(i); - zwiększ wartość w zmiennej sterującej o jeden
                        k:=ord(la2[i]); - odczytaj wartość następnego znaku
                   end;
                   tym2[0]:='0'; - wpisz znak pochodnej do łańcucha tym2
                   slownik:='0'; - wpisz kod rozpoznanej funkcji do slownik
                   s:=false; - za liczbą może znajdować się znak działania
                   st:=true; - pochodna została znaleziona
                   inc(i); - zwiększ wartość zmiennej sterującej pętlą o jeden
                   continue - przerwij następne instrukcje i rozpocznij nowy cykl
              end;
```

```
        if k=112 then - czy wartość znaku odpowiada literze 'π'?
           if not st then - jeśli tak, to czy pochodna nie została znaleziona?
           begin - jeśli nie, to...
                 tym2[0]:='0'; - wpisz znak pochodnej do łańcucha tym2
                 slownik:='0'; - wpisz kod rozpoznanej funkcji do slownik
                 s:=false; - za liczbą może znajdować się znak działania
                 st:=true; - pochodna została znaleziona
                 inc(i); - zwiększ wartość zmiennej sterującej pętlą o jeden
                 continue - przerwij następne instrukcje i rozpocznij nowy cykl pętli
           end;
        if k=101 then - czy wartość znaku odpowiada literze 'e'?
           if not st then - jeśli tak, to...
           begin
                 tym2[0]:='0';
                 slownik:='0';
                 s:=false;        } (jak wyżej)
                 st:=true;
                 inc(i);
                 continue
           end;
        if k=120 then - czy wartość znaku odpowiada literze 'x'?
           if not st then - jeśli tak, to...
           begin
                 tym2[0]:='1';
                 slownik:='1';
                 s:=false;        } (jak wyżej)
                 st:=true;
                 inc(i);
                 continue
           end;
        if (k>47) and (k<>94) then - czy znak jest niewłaściwy?
        begin - wartość jest różna od znaków działania, więc...
              bl:=true;
              blw:=10;    } znak nie został rozpoznany w poprzednich...
              break       } instrukcjach, więc przerwij pętlę z błędem
        end;
        inc(i) - zwiększ wartość zmiennej sterującej pętlą o jeden
until k=0; - opuść pętlę, gdy odczytany znak jest pusty
. . .
```

Jeżeli pętla `repeat..until` została opuszczona bez błędu a zmienna `st` posiada wartość `false`, procedura kończy swoją pracę z ustawioną zmienną błędu, w przeciwnym przypadku uruchamiany jest blok instrukcji, w którym łańcuchy zewnętrzne są uzupełniane znalezioną pochodną. Gdy łańcuch `la1` rezyduje już w pamięci komputera, lokalnej zmiennej pomocniczej `pom1` przydzielona zostaje pamięć, której rozmiar będzie równy sumie długości łańcuchów `la1`, `tym2` oraz `tym1`, pod warunkiem, że został on wykorzystany w procesie szukania pochodnej. Po uzupełnieniu łańcucha `pom1` zawartością wymienionych łańcuchów, zwolnieniem pamięci dotychczas zajmowanej przez łańcuch zewnętrzny `la1` i przypisaniem mu adresu łańcucha lokalnego `pom1`, procedura kończy swoją pracę dla tego przypadku.

Gdy dla łańcucha `la1` nie została przydzielona pamięć, praca tej części bloku sprowadza się do przydzielenia pamięci temu łańcuchowi o rozmiarze jednego lub dwóch znaków, zależnie od tego, czy łańcuch `tym1` ma wpisany znak odejmowania czy nie (w tym przypadku odrzuca się znak dodawania, by pochodna nie rozpoczynała się od tego znaku) oraz uzupełnienia łańcucha znakami z łańcuchów lokalnych. W przypadku łańcucha `la3`, w pierwszej kolejności, gdy łańcuch nie ma adresu pustego, uwalniana jest pamięć przez niego zajmowa-

na. Kolejne instrukcje są identyczne jak dla łańcucha `la1`. Tak więc uzupełnienie łańcuchów zewnętrznych znakami znalezionej pochodnej kończy pracę procedury.

```
if not bl then - czy nie ma błędu?
   if st then - jeśli nie, to czy pochodna została znaleziona?
   begin - jeśli tak, to...
        if la1<>nil then - czy łańcuch la1 istnieje w pamięci?
        begin - jeśli tak, to...
             i:=StrLen(la1); - z ilu znaków składa się łańcuch la1?
             if tym1[0]<>#0 then k:=1 else k:=0; - ustaw zmienną k...
                                  jeśli łańcuch tym1 zawiera znak inny niż pusty
             GetMem(pom1,(i+k)*sizeof(PChar)); - przydziel pamięć...
                             o rozmiarze łańcucha tym1 i wartości w zmiennej k
             StrCopy(pom1,la1);
             StrCat(pom1,tym1);  } skopiuj zawartość łańcuchów lokalnych do pom1
             StrCat(pom1,tym2);
             FreeMem(la1,i*sizeof(PChar)); - zwolnij pamięć zajmowaną...
                                                przez łańcuch la1
             la1:=pom1 - przypisz łańcuchowi la1 adres łańcucha lokalnego pom1
        end else - w przeciwnym przypadku łańcuch la1 nie istnieje, więc...
            begin
                 if tym1[0]='-' then k:=2 else k:=1; - ustaw zmienną...
                       k2 w zależności od istnienia lub nie znaku '– ' w łańcuchu tym1
                 GetMem(la1,k*sizeof(PChar)); - przydziel pamięć dla la1
                 StrCopy(la1,tym1);  } skopiuj zawartość łańcuchów lokalnych...
                 StrCat(la1,tym2)    } do łańcucha zewnętrznego la1
            end;
        if la3<>nil then - czy łańcuch la3 zajmuje pamięć?
        begin - jeśli tak, to...
             i:=StrLen(la3); - z ilu znaków składa się łańcuch?
             FreeMem(la3,i*sizeof(PChar)) - zwolnij pamięć…
                                  zajmowaną przez łańcuch la3
        end;
        if tym1[0]='-' then k:=2 else k:=1; - ustaw zmienną k2...
                   w zależności od istnienia lub nie znaku '– ' w łańcuchu tym1
        GetMem(la3,k*sizeof(PChar)); - przydziel pamięć dla łańcucha la3
        StrCopy(la3,tym1);  } skopiuj zawartość łańcuchów lokalnych...
        StrCat(la3,tym2)    } do łańcucha zewnętrznego la3
   end else - gdy pochodna nie została znaleziona, to...
       begin
            bl:=true;  } ustaw zmienną błędu i kod błędu
            blw:=12    }
       end;
```

X. 4. Przepisanie znaków pochodnej do łańcucha wynikowego – procedura `przepisz`

Zadaniem tej procedury jest prawidłowe przygotowanie pola `t4` należącego do bieżącego elementu `bloki` i przepisania do niego zawartości pola `t3`. Skoro operacja ta ma przebiegać w ramach tego samego obiektu, korzystne okazało się przekazanie procedurze adresu tego elementu. Przekazany parametr jest poprzedzony literałem `const`, który co prawda nie pozwala na zmianę adresu samego elementu `bloki`, ale można modyfikować jego pola. Na początku pracy procedury sprawdzana jest zawartość łańcucha, czyli pola `t3` – zawarty w nim adres pusty oznacza błąd powodujący ustawienie zmiennej błędu `bl` w pozycji `true` oraz bezwzględne opuszczenie procedury instrukcją `exit` . Jeśli jednak łańcuch istnieje w pamięci, w lokalnej zmiennej zapamiętana zostaje jego długość. Podobnie sprawdzana jest zawartość łańcucha, czyli pola `t4` – jeśli istnieje w pamięci, kolejna zmienna lokalna zapamiętuje jego długość. Sposób skopiowania zawartości pól `t3` do `t4` jest uzależniony od wartości w zmiennej lokalnej `v2`:

- gdy jest większa od zera, co jest równoznaczne z wcześniejszym ulokowaniem łańcucha dostępnego przez pole `t4` w pamięci, łańcuchowi `pom1`, przydzielona zostaje pamięć o rozmiarze równym sumie długości łańcuchów dostępnych przez pola: `t4` i `t3`, po czym do łańcucha `pom1` zostaje skopiowana zawartość tych łańcuchów. Po skopiowaniu, pamięć zajmowana przez pole `t4` zostaje uwolniona;
- gdy jest równa zero, sprawdzany jest pierwszy znak w łańcuchu dostępnym przez pole `t3` – jeśli jest nim znak dodawania, znak ten zostaje pominięty a łańcuchowi `pom1` zostaje przydzielona pamięć o rozmiarze pomniejszonym o ten znak – dla każdego innego znaku początkowego, pamięć o rozmiarze równym długości tego łańcucha zostaje przydzielona łańcuchowi `pom1`. Również w tym przypadku, zawartość pola `t3` zostaje skopiowana do łańcucha `pom1`.

Po zwolnieniu pamięci zajmowanej przez pole `t3` oraz przypisaniu polu `t4` adresu łańcucha lokalnego `pom1`, procedura kończy swoją pracę.

```
procedure przepisz(const lancuch:bloki);
var pom1,tym1:PChar;
    v1,v2:Integer;
begin
     v2:=0;
     if lancuch^.t3<>nil then v1:=StrLen(lancuch^.t3) - jeśli pole t3. nie
..
                                   jest zainicjowane adresem pustym, zapamiętaj w zmiennej v1…
                                   liczbę znaków zawartą w łańcuchu dostępnemu przez to pole
     else - gdy jest zainicjowane adresem pustym, jest to błąd
     begin
          bl:=true;
          blw:=10;
          exit
     end;
     if lancuch^.t4<>nil then v2:=StrLen(lancuch^.t4); - jeśli pole t4…
                        nie jest zainicjowane adresem pustym, zapamiętaj w zmiennej v2…
                        liczbę znaków zawartą w łańcuchu dostępnym przez to pole
     pom1:=nil; - łańcuch pomocniczy pom1 zainicjuj adresem pustym
     if v2>0 then - czy w polu t4 jest przynajmniej jeden znak?
```

```
        begin - jeśli tak, to...
             GetMem(pom1,(v1+v2)*sizeof(PChar)); - przydziel pamięć dla...
                                  łańcucha lokalnego pom1 o rozmiarze równym…
                                  sumie długości łańcuchów z pól t3 i t4

             StrCopy(pom1,lancuch^.t4); - skopiuj zawartość pola t4 do pom1
             FreeMem(lancuch^.t4,v2*sizeof(PChar)); - zwolnij pamięć zajętą ...
                                                      przez łańcuch z pola t4

             StrCat(pom1,lancuch^.t3) - dopisz do pom1 zawartość łańcucha…
                                        z pola t3
        end else - gdy łańcuch z pola t4 jest pusty, to...
            if lancuch^.t3[0]='+' then - czy łańcuch z pola t3 rozpoczyna się…
                                         od znaku '+'?
            begin - jeśli tak, to...
                 tym1:=@lancuch^.t3[1]; - zapamiętaj adres łańcucha z pola t3...
                                          rozpoczynając się od następnego znaku

                 if tym1[0]<>#0 then - czy łańcuch tym1 nie rozpoczyna się...
                                       od znaku pustego?
                 begin - jeśli nie, to...
                      v1:=StrLen(tym1); - z ilu znaków składa się ten łańcuch?
                      GetMem(pom1,v1*sizeof(PChar)); - przydziel pamięć...
                       łańcuchowi pom1 o rozmiarze równym długości łańcucha tym1

                      StrCopy(pom1,tym1) - skopiuj zawartość łańcucha tym1...
                                           do pom1

                 end else - pierwszy znak w łańcuchu tym1 jest pusty – jest to błąd
                     begin
                          bl:=true;
                          blw:=10
                     end
            end else - łańcuch z pola t3 nie rozpoczyna się od znaku '+', więc...
                begin
                     GetMem(pom1,v1*sizeof(PChar)); - przydziel pamięć...
                    dla łańcucha pom1 o rozmiarze równym długości łańcucha z pola t3

                     StrCopy(pom1,lancuch^.t3) - skopiuj zawartość łańcucha...
                                                 z pola t3 do łańcucha pom1
                end;
        FreeMem(lancuch^.t3,v1*sizeof(PChar)); - zwolnij pamięć zajmowaną…
                                                 przez łańcuch z pola t3

        lancuch^.t3:=nil; - przypisz polu t3 identyfikator adresu pustego
        lancuch^.t4:=pom1 - przypisz polu t4 adres łańcucha pom1
end;
```

X. 5. Czy przenieść znak +/– na początek? Funkcja `czy_przeniesc_znak`

Zadaniem funkcji jest zbadanie, czy pierwszym znakiem w dostarczonym łańcuchu jest znak dodawania lub odejmowania oraz czy znak ten można przenieść w inne miejsce układanej pochodnej. Rozważmy trzy przypadki funkcji:

a) $-x+1$
b) $-(x+1)$
c) $-(x+1)+\pi$

Przykład pierwszy (a) przedstawia funkcję, w której przeniesienie znaku odejmowania zmieni sens matematyczny tej funkcji, więc tego znaku nie wolno przenosić. W drugim przykładzie (b), przeniesienie znaku minus nie wpłynie na funkcję zamkniętą w nawiasie, dlatego znak ten można wstawić w inne miejsce. W trzecim przykładzie (c), za nawiasem znajduje się stała π poprzedzona znakiem dodawania – również w tym przypadku usunięcie początkowego znaku odejmowania zmieni sens matematyczny tej funkcji, dlatego znaku nie wolno przenosić w inne miejsce (gdyby znak π poprzedzony był np. znakiem mnożenia, wówczas przeniesienie znaku minus byłoby dozwolone, gdyż znak mnożenia ma wyższy priorytet względem znaku dodawania czy odejmowania).

Po zidentyfikowaniu na początku łańcucha znaku dodawania lub odejmowania, kolejne znaki sprawdzane są już w pętli `while..do` – gdyby pojawił się kolejny znak dodawania lub odejmowania i znak ten nie byłby zamknięty w nawiasie, następuje opuszczenie pętli z ustawioną zmienną logiczną `naw4` w pozycji `false`, dając tym samym odpowiedź, że znaku z początku łańcucha nie wolno przenosić. Gdyby jednak po sprawdzeniu pełnego łańcucha zmienna `naw4` miała wartość `true`, początkowy znak dodawania lub odejmowania zostaje pominięty przez przekazanie łańcuchowi lokalnemu `pom2` adresu łańcucha zewnętrznego `lan_A`, ale od następnego znaku, a kolejno przeniesienie zawartości łańcucha `pom2` z powrotem do łańcucha zewnętrznego `lan_A` już bez początkowego znaku dodawania czy odejmowania. W tym przypadku, gdy usuniętym znakiem okazało się odejmowanie, funkcja zwraca wartość `true`, w przeciwnym przypadku – `false`. Zwracana wartość wykorzystana będzie w celu zliczenia pozostałych znaków odejmowania zawartych w innych funkcjach i wyłonienia znaku wypadkowego – mechanizm ten będzie dokładnie wyjaśniony podczas opisywania procedur `dwa` oraz `trzy`.

```
function czy_przeniesc_znak(var lan_A:PChar):boolean;
const zn_pm=[43,45];- tablica stałych zawierająca wartości znaków dodawania...
                       i odejmowania
var i2,v2,v2p,n2:integer;
    pom2:PChar;
    naw4:boolean;
begin
     Result:=false; - domyślne zwrócenie przez funkcję wartości false
     i2:=0;
     v2p:=ord(lan_A[i2]); - odczytaj wartość pierwszego znaku w łańcuchu...
                            zewnętrznym
     if v2p in zn_pm then - czy pierwszym znakiem jest dodawanie lub odejmowanie?
     begin - jeśli tak, to...
          n2:=0; - wyzeruj licznik nawiasów
          i2:=1;
          naw4:=true; - domyślnie, pierwszy znak +/– wolno usunąć
          v2:=ord(lan_A[i2]); - odczytaj wartość następnego znaku w łańcuchu
          while v2<>0 do - wykonaj krok pętli, póki wartość znaku nie jest pusta
          begin
               if v2=40 then inc(n2); - dla nawiasu otwierającego zwiększ n2
               if v2=41 then dec(n2); - dla nawiasu zamykającego zmniejsz n2
               if v2 in zn_pm then - czy znak znajduje się w tablicy zn_pm?
                  if n2<=0 then - jeśli tak, to czy znak nie jest zamknięty...
                                  w nawiasie?
                  begin jeśli nie, to...
                       naw4:=false; - pierwszego znaku +/– nie wolno usunąć
                       break - dalsze sprawdzanie jest bezcelowe, opuść pętlę
                  end;
               inc(i2); - zwiększ wartość zmiennej sterujące pętlą o jeden
```

```
                v2:=ord(lan_A[i2]) - odczytaj wartość następnego znaku
            end;
            if naw4 then - czy po opuszczeniu pętli można usunąć pierwszy znak +/– ?
            begin - jeśli tak, to...
                pom2:=@lan_A[1]; - zapamiętaj adres łańcucha od następnego znaku
                StrCopy(lan_A,pom2); - skopiuj zawartość łańcucha pom2...
                                       do lan_A
                if v2p=45 then Result:=true; - gdy usunięto znak…
                                               odejmowania...
                                               funkcja zwraca wartość true
            end;
        end;
    end;
```

X. 6. Czy brak pary nawiasów? Funkcja `czy_brak_nawiasu`

Zbadanie dostarczonego łańcucha znakowego pod względem ewentualnego braku wymaganej pary nawiasów oraz zwrócenie informacji w postaci wartości logicznej – to podstawowe zadanie niniejszej funkcji. Poza łańcuchem znaków do zbadania, funkcja otrzymuje jeszcze drugi parametr, tym razem liczbowy, który określa kryterium badania łańcucha. Badanie odbywa się w pętli `while..do`, w której znaki nawiasów są zliczane w ten sposób, że dla nawiasu otwierającego zmienna zliczająca je zwiększa swoją wartość o jeden, natomiast dla nawiasu zamykającego – zmniejsza ją o jeden. Gdy wartość tej zmiennej będzie wynosiła zero oraz gdy sprawdzany znak nie jest pierwszym w łańcuchu, w instrukcji wyboru `case` porównywane są ten i następne znaki z jednym z trzech możliwych, stałych zestawów znaków, których wybór podyktowany został przez drugi parametr dostarczony funkcji:

- dla parametru równego zero, identyfikowane są te znaki, które zawierają się w tablicy `zestaw_A`, składającej się z wartości znaków dodawania i odejmowania. Przypadek dotyczy łańcucha znaków, który będzie wstawiony za znakiem dodawania, odejmowania lub mnożenia. Gdy identyfikacja wypadnie pomyślnie, funkcja zwróci wartość `true` i opuści pętlę `while..do`;
- dla parametru równego jeden, identyfikowane są znaki, zawierające się w tablicy `zestaw_C`, składającej się z wartości znaków dodawania, odejmowania, mnożenia, dzielenia oraz potęgowania. Gdyby identyfikacja wypadła niepomyślnie, dodatkowo dla tego przypadku wywołana zostaje funkcja `czy_pierwiastek`, która zwróci wartość `true`, gdyby w dostarczonym łańcuchu znajdował się znak pierwiastkowania. Ponieważ znak pierwiastkowania może nie być pierwszym znakiem w łańcuchu, jego zidentyfikowanie w omawianej pętli byłoby niepewne, dlatego dodatkowa funkcja dokładnie sprawdzi obecność tego znaku w łańcuchu. W obu przypadkach badanie dotyczy tego łańcucha znaków, za którym ma znaleźć się znak potęgowania, a więc dla funkcji potęgowanej. Pozytywna identyfikacja spowoduje zwrócenie przez funkcję wartości `true` i opuszczenie pętli;
- dla parametru równego dwa, identyfikowane są znaki, zawierające się w tablicy `zestaw_B` składającej się z wartości znaków dodawania, odejmowania, mnożenia i dzielenia (bez znaku potęgowania i pierwiastkowania). Ten przypadek dotyczy łańcucha znaków, który będzie wstawiony za znakiem potęgowania. Tak jak w powyższych przypadkach, pozytywna identyfikacja spowoduje zwrócenie przez funkcję wartości `true` i opuszczenie pętli.

```
function czy_brak_nawiasu(const lancuch:PChar;przel:byte):boolean;
const zestaw_A=[43,45]; - tablica stałych z wartościami znaków dodawania…
                           i odejmowania
      zestaw_B=[42,43,45,47]; - tablica stałych z wartościami znaków mnożenia...
                                dodawania, odejmowania i dzielenia
      zestaw_C=[42,43,45,47,94]; - tablica stałych składająca się z wartości ...
                                   wszystkich znaków działania
var i,k1,n7:integer;

    function czy_pierwiastek(const lan_p:PChar):boolean;

begin
     i:=0;
     n7:=0;
     Result:=false; - domyślne zwrócenie wartości false
     if lancuch=nil then exit; - gdyby łańcuch był pusty,...
                                 bezwzględnie opuść funkcję
     k1:=ord(lancuch[i]); - odczytaj wartość pierwszego znaku z łańcucha
     while k1<>0 do - wykonaj krok pętli, póki wartość odczytanego znaku jest…
                      różna od zera
     begin
          if k1=40 then inc(n7); - dla znaku nawiasu otwierającego zwiększ n7
          if k1=41 then dec(n7); - dla znaku nawiasu zamykającego zmniejsz n7
          if i>0 then - pierwszy znak nie jest analizowany
             if n7=0 then - jeśli nie jest to pierwszy znak to czy licznik nawiasów...
                            równy jest zero?
                case przel of - gdy licznik nawiasów jest równy zero, to...
                     0: - dla pierwszego kryterium badania, czyli dla przel=0
                     if k1 in zestaw_A then - czy odczytany znak...
                                              zawiera się w tablicy zestaw_A?
                     begin - jeśli tak, to...
                          Result:=true; - funkcja zwraca wartość true
                          break - opuść pętlę
                     end;
                     1: - dla drugiego kryterium badania, czyli dla przel=1
                     begin
                          if k1 in zestaw_C then - czy odczytany znak...
                                                   zawiera się w tablicy zestaw_B?
                          begin - jeśli tak, to...
                               Result:=true; - zwróć wartość true
                               break - opuść pętlę
                          end;
                          if czy_pierwiastek(lancuch) then
                          begin - jeśli pierwiastek nie jest zamknięty…
                                  w nawiasie, to...
                               Result:=true; - zwróć wartość true
                               break - opuść pętlę
                          end;
                     end;
                     2: - dla trzeciego kryterium badania, czyli dla przel=2
                     if k1 in zestaw_B then - czy odczytany znak...
                                              zawiera się w tablicy zestaw_B?
                     begin - jeśli tak, to...
                          Result:=true; - funkcja zwraca wartość true
                          break - opuść pętlę
```

```
                    end;
                end;
            inc(i); - zwiększ zmienną sterującą o jeden
            k1:=ord(lancuch[i]) - odczytaj wartość następnego znaku
        end;
    end;
```

X. 6.1. Czy pochodna zawiera pierwiastek? Funkcja `czy_pierwiastek`

Funkcja ta, poza wyszukiwaniem w dostarczonym łańcuchu znaku pierwiastka, sprawdza jeszcze, czy pierwiastek nie jest już zamknięty w nawiasie, dlatego w pętli `while..do` zliczane są nawiasy – znalezienie znaku pierwiastkowania poprzedzonego znakiem nawiasu otwierającego powoduje opuszczenie pętli a funkcja zwraca wartość `false`, w przeciwnym przypadku, gdy znaleziony znak pierwiastka nie jest poprzedzony znakiem nawiasu otwierającego, funkcja zwraca wartość `true`.

```
function czy_pierwiastek(const lan_p:PChar):boolean;
var ia1,na1,ka1:integer;
begin
     na1:=0; - wyzeruj licznik nawiasów
     ia1:=0; - wyzeruj zmienną sterującą pętlą
     Result:=false; - domyślne zwrócenie przez funkcję wartości false
     ka1:=ord(lan_p[ia1]); - odczytaj wartość pierwszego znaku
     while ka1<>0 do - wykonaj krok pętli, póki wartość odczytanego znaku...
                       jest różna od zera
     begin
          if ka1=40 then inc(na1); - dla nawiasu otwierającego…
                                     zwiększ na1
          if ka1=41 then dec(na1); - dla nawiasu zamykającego..
                                     zmniejsz na1
          if ka1=92 then - czy obecnie odczytanym znakiem jest pierwiastek?
             if na1>0 then break - jeśli tak, to gdy licznik nawiasów jest...
                                   większy od zera – opuść pętlę
             else - gdy licznik nawiasów nie jest większy od zera, to...
             begin
                  Result:=true; - zwróć wartość true
                  beak - opuść pętlę
             end;
          inc(ia1); - zwiększ wartość zmiennej sterującej pętlą o jeden
          ka1:=ord(lan_p[ia1]) - odczytaj wartość następnego znaku
     end;
end;
```

X. 7. Usunięcie z części funkcji pary nawiasów – procedura `z_nawiasu`

Procedura ma za zadanie sprawdzenie, czy z dostarczonego łańcucha znaków można usunąć skrajną parę nawiasów – jeśli tak, to nawiasy zostają z łańcucha usunięte. Podstawowym warunkiem usunięcia nawiasów jest obecność nawiasu otwierającego na początku łańcucha i nawiasu zamykającego na jego końcu. Gdy warunek zostanie spełniony, wewnętrzna

funkcja `czy_znaki` sprawdzi jeszcze, czy w dostarczonym jej łańcuchu znaków nie znajduje się znak działania usytuowany poza nawiasem. Spróbujmy przeanalizować następującą funkcję:

$$(x+1)\cdot(x-1)$$

Zaprezentowana funkcja, co prawda, posiada nawias otwierający na początku i zamykający na końcu, ale obecność znaku mnożenia między nawiasami uniemożliwia usunięcie tych nawiasów, dlatego funkcja `czy_znaki` musi wykryć taki lub podobne przypadki i nie dopuścić do powstania błędu.

Jeśli wszystkie powyższe warunki zostaną spełnione, do dostarczonego w parametrze łańcucha `tablica` zostanie skopiowany łańcuch znaków już bez skrajnych nawiasów.

```
procedure z_nawiasu(var tablica:PChar);
var tym1:PChar;
    b2:integer;

    function czy_znaki(const pom4:PChar):boolean; - funkcja wewnętrzna

    const znaki=[42,43,45,47,94]; - tablica stałych, składająca się z wartości...
                 znaków mnożenia, dodawania, odejmowania, dzielenia i potęgowania
    var b1,bw1,bn1:integer;
        pzn1:boolean;
    begin
         b1:=ord(pom4[0]); - odczytaj wartość pierwszego znaku z łańcucha pom4
         bw1:=0; - wyzeruj zmienną sterującą pętlą
         bn1:=0; - wyzeruj licznik nawiasów
         pzn1:=false; - przypisz zmiennej pomocniczej początkową wartość false
         while b1<>0 do - wykonaj krok pętli, póki wartość znaku jest różna od zera
         begin
              if b1=40 then inc(bn1); - dla nawiasu otwierającego zwiększ bn1
              if b1=41 then dec(bn1); - dla nawiasu zamykającego zmniejsz bn1
              if b1 in znaki then - czy odczytany znak jest znakiem działania?
                 if bn1=0 then - jeśli tak, to czy licznik nawiasów jest równy zero?
                 begin - jeśli tak, to...
                      pzn1:=true; - nawiasów nie wolno usuwać
                      break - opuść pętlę – dalsze sprawdzanie jest bezcelowe
                 end;
              inc(bw1); - zwiększ zmienną sterującą o jeden
              b1:=ord(pom4[bw1]) - odczytaj wartość następnego znaku
         end;
         Result:=pzn1 - zwróć wartość zawartą w zmiennej pomocniczej pzn1
    end;

begin
     if tablica=nil then exit; - gdy dostarczony łańcuch jest pusty...
                               – opuść procedurę
     b2:=StrLen(tablica); - z ilu znaków składa się łańcuch?
     if tablica[b2-1]<>')' then exit; - jeśli ostatni, niepusty znak nie jest ...
                                        nawiasem zamykającym – opuść procedurę
     if tablica[0]<>'(' then exit; - jeśli pierwszym znakiem nie jest nawias ...
                                     otwierający – opuść procedurę
     tym1:=@tablica[1]; - zapamiętaj adres łańcucha tablica od następnego znaku
```

```
        dec(b2,2); - zmniejsz o dwa ilość znaków zawartych w łańcuchu tablica
        if b2>0 then - czy po zmniejszeniu liczby znaków, liczba jest większa od zera?
           if not czy_znaki(tablica) then - czy w dostarczonym łańcuchu...
                                            nie ma znaków działania, które nie są zamknięte w nawiasie?

              StrLCopy(tablica,tym1,b2); - jeśli nie, to skopiuj z łańcucha tym1...
                                           do łańcucha tablica ilość znaków,...
                                           zawartą w zmiennej b2- ...
                                           pomniejszoną o dwa skrajne nawiasy
           end;
end;
```

Skopiowanie do zewnętrznego łańcucha `tablica` pomniejszonej liczby znaków tworzy pewną niebezpieczną sytuację związaną z wyciekiem pamięci. Wynika to z tego, że zbadanie rzeczywistej zajętości pamięci przez łańcuch o pomniejszonej zawartości jest już niemożliwe. By jednak nie dopuścić do wycieku pamięci podczas jej uwalniania, bezwzględnie należy zachować wartość rozmiaru łańcucha podczas przydziału dla niego pamięci i taką samą wartość wykorzystać podczas uwalniania pamięci. Niezachowanie tej zasady skutkuje, w dłuższej perspektywie pracy programu, sukcesywnym kurczeniem się wolnej przestrzeni adresowej pamięci komputera.

X. 8. Pochodna z funkcji trygonometrycznej – procedura `dwa`

Procedura ta znajduje pochodną z funkcji trygonometrycznych, cyklometrycznych, pierwiastków i logarytmów. Pierwszym elementem dostarczonym procedurze w parametrze jest adres bieżącego elementu `bloki` – jest on poprzedzony przedrostkiem `const`, który nie pozwala na zmianę jego adresu, ale zezwala na modyfikację pól. W łańcuchu dostępnym przez pole `t2` znajduje się funkcja, z której procedura ma znaleźć pochodną i umieścić w polu `t3`. Drugim elementem dostarczonym procedurze jest łańcuch znaków zawierający pochodną argumentu funkcji, znalezioną w poprzednich krokach programu. Trzecim elementem jest zmienna znakowa, która po bezbłędnym opuszczeniu procedury będzie zawierała kod rozpoznanej funkcji. Pierwszą czynnością procedury jest sprawdzenie, czy w łańcuchu dostępnym przez pole `t2` znajduje się nawias otwierający – najbardziej odpowiednią instrukcją do tego celu okazała się funkcja biblioteczna `StrScan`, z zawartym w jej drugim parametrze znakiem nawiasu otwierającego. Funkcja ta szuka pierwszego wystąpienia tego znaku w dostarczonym łańcuchu – jeśli nawias zostanie znaleziony, zwraca ona adres łańcucha rozpoczynającego się od znalezionego nawiasu – gdyby nawias nie został znaleziony, wówczas funkcja ta zwróci adres pusty, który wykryty w instrukcji warunkowej spowoduje natychmiastowe opuszczenie procedury z ustawioną zmienną błędu w pozycji `true`. Znalezienie nawiasu pozwala na dalszą pracę procedury – jej kolejnym krokiem jest zapamiętanie w zmiennej `kd1` liczby znaków znajdujących się w łańcuchu `tym1` (póki nie jest rozpoznana funkcja, może się w nim znajdować funkcja logarytmowana, podpierwiastkowa lub argument funkcji trygonometrycznej). Wydzielony łańcuch `tym1` jest częścią łańcucha dostępnego przez pole `t2`, dlatego nie należy go modyfikować. W tej sytuacji utworzony został w pamięci kolejny łańcuch `pom2` – jego rozmiar odpowiada liczbie znaków zawartych w łańcuchu `tym1`. Po skopiowaniu zawartości łańcucha `tym1` do `pom2`, usunięciu z niego skrajnych nawiasów i wyliczeniu ilości znaków po usunięciu nawiasów, odbywa się rozpoznanie funkcji i znalezienie jej pochodnej, pod warunkiem, że różnica ilości znaków w łańcuchach: `pom2` oraz łańcucha do-

stępnego przez pole t2, będzie większa od jedności lub gdy pierwszym znakiem w łańcuchu z pola t2 będzie znak pierwiastkowania. Może się zdarzyć, że w łańcuchu z pola t2 będzie znajdowała się funkcja zamknięta w nawiasie z dołączonym na początku łańcucha znakiem dodawania lub odejmowania. By podczas uzupełniania łańcucha z pola t3 znakami pochodnej nie pominąć tego znaku, jego możliwa obecność w łańcuchu z pola t2 została zaznaczona przez ustawienie zmiennej logicznej zn2 w pozycji true. Na podstawie różnicy znaków między łańcuchami: z pola t2 i pom2, utworzony zostaje w pamięci kolejny łańcuch – tym2, do którego skopiowana zostaje, wyliczona ilość znaków z łańcucha z pola t2, stanowiąca rdzeń funkcji. Owy rdzeń może być poprzedzony znakiem dodawania lub odejmowania (pozostałe znaki działania zostały odrzucone podczas rozkładu funkcji), dlatego po wykryciu tych znaków w instrukcji warunkowej, do zmiennej tym1 zostaje wczytany adres łańcucha tym2 rozpoczynającego się od następnego znaku, a następnie skopiowana zostaje zawartości łańcucha tym1 z powrotem do tym2. W tym przypadku, wykorzystanie zmiennej łańcuchowej tym1 w pozbyciu się znaku jest jedyną możliwością, ponieważ nie wolno przypisywać adresu łańcucha z innego miejsca niż z jego początku tej samej zmiennej łańcuchowej, gdyż doprowadziłoby to do pozostawienia w pamięci znaku bez dostępu do niego oraz do sztucznego ograniczenia przestrzeni adresowej dla łańcucha, czego konsekwencją byłoby pojawienie się błędu podczas uwalniania pamięci zajmowanej przez łańcuch.

```
tym2:=@tym2[1]; – błędna instrukcja!!!
```

Z kolei rdzeń funkcji może składać się ze znaków cyfr, dlatego zostają one przeniesione do lokalnego łańcucha typu Char o nazwie ww, zaś w samym łańcuchu tym2, znaki te zostają zastąpione znakami pustymi. Cyfry znajdujące się między rdzeniem funkcji a nawiasem otwierającym mogą należeć do podstawy logarytmu lub stanowić stopień pierwiastka. Gdy łańcuch ww jest pusty, uruchomiony zostaje blok rozpoznawania funkcji trygonometrycznych i logarytmu naturalnego. Przed uruchomieniem bloku, zmienna logiczna sd zostaje zainicjowana wartością false – będzie stanowiła informację dla końcowych instrukcji procedury, czy funkcja została rozpoznana czy też nie. Blok składa się z ciągu instrukcji warunkowych, dostosowanych do wszystkich możliwych funkcji, które nie będą wykorzystywały łańcucha ww. Każda z tych instrukcji składa się z funkcji bibliotecznej StrComp, która porównuje ze sobą łańcuch tym2 zawierający rdzeń funkcji z nazwą jednej z dziewięciu funkcji – jeśli porównanie wypadnie pomyślnie, funkcja zwraca wartość zero, uruchamiając jedną ze wskazanych procedur wewnętrznych – każda z tych procedur otrzymuje w parametrach łańcuch wynikowy za pośrednictwem pola t3 należącego do elementu bloki, łańcuch pom2 zawierający znaki argumentu funkcji i łańcuch lancuch2 w którym powinna znajdować się pochodna argumentu. Po powrocie z procedury wewnętrznej, zmienna slownik przyjmuje kod znakowy odpowiedni do rozpoznanej funkcji, a zmiennej logicznej sd przypisana zostaje wartość true, co oznacza, że funkcja została rozpoznana. Pozostałe dwie części bloku dotyczą rozpoznawania pierwiastka i logarytmu o dowolnej podstawie, których uruchomienie zależne jest od niepustego, pierwszego znaku w łańcuchu ww. Po opuszczeniu powyższego bloku instrukcji, ustalany jest znak pochodnej. Odbywa się to przez porównanie pierwszych znaków zawartych w łańcuchach dostępnych przez pola t2 i t3 – gdy oba łańcuchy rozpoczynają się znakiem odejmowania, znak w łańcuchu wynikowym zostaje zamieniony na dodawanie, gdy znak odejmowania znajduje się tylko w jednym z łańcuchów, łańcuch w polu t3 będzie się rozpoczynał od znaku odejmowania, co jest oczywiście zgodne z zasadami matematyki. Bez względu na to, czy pochodna została rozpoznana czy też nie, pamięć zajmowana przez

łańcuchy tym2 i pom2 zostaje uwolniona w całości, dzięki zachowaniu pierwotnej wielkości tych łańcuchów w zmiennych kd4 i kd1.

Gdy rdzeń funkcji nie istnieje, to znaczy, że cała dostarczona funkcja zamknięta jest w nawiasie, procedura musi uwolnić pamięć zajmowaną przez łańcuch pom2. W zmiennej kd1 zostaje zapamiętana ilość znaków znajdujących się w łańcuchu lancuch2 (pochodna funkcji zamkniętej w nawiasie). W kolejnej zmiennej kd2 znajduje się dodatkowa ilość znaków, które są wynikiem obecności znaku dodawania lub odejmowania na początku łańcucha z pola t2, o czym informuje wcześniej ustawiona zmienna logiczna zn2 w pozycji true. Jeśli znak ten istnieje, musi się znaleźć w łańcuchu wynikowym z pola t3, a w przypadku znaku odejmowania także i dodatkowa para nawiasów, by zachować zgodność znakową pochodnej względem znaku minus. Po ustaleniu ostatecznej liczby znaków, które mają znaleźć się w łańcuchu z pola t3, przydzielana jest dla niego pamięć, której rozmiar zależny jest od sumy wartości zmiennych kd1 i kd2. Zależnie od wartości zmiennej kd2, do łańcucha z pola t3 dołączone zostaną:

- dla zerowej wartości – tylko zawartość łańcucha lancuch2;
- dla wartości równej jeden – znak dodawania i zawartość łańcucha lancuch2;
- dla wartości równej trzy – znak odejmowania i zawartość łańcucha lancuch2; która zamknięta została w dodatkowej parze nawiasów.

```
procedure dwa(const lancuch1:bloki;
              var lancuch2:PChar;var slownik:char);
const liczby=[44,46,48,49,50,51,52,53,54,55,56,57,101,112];
                tablica stałych, składająca się z wartości cyfr i separatorów dziesiętnych
var kd1,kd2,kd3,kd4:integer;
    jd:byte;
    sd:boolean;
    tym1,tym2,pom2:PChar;
    ww:array[0..5] of char;

begin
     slownik:=#0; - zainicjuj zmienną slownik znakiem pustym,...
                    gdyby funkcja nie została rozpoznana,
     tym1:=StrScan(lancuch1^.t2,'('); - zwróć adres łańcucha zmiennej tym1...
                                rozpoczynającego się od znaku nawiasu otwierającego
     if tym1=nil then - czy łańcuch tym1 jest pusty, czyli nawias nie został znaleziony?
     begin - jeśli tak, to...
          bl:=true; - ustaw zmienną błędu w pozycji true
          blw:=3; - zmiennej kodu błędu przypisz stosowny numer
          exit - opuść procedurę
     end;
     kd1:=StrLen(tym1); - z ilu znaków składa się łańcuch tym1 (argument funkcji)?
     GetMem(pom2,kd1*sizeof(PChar)); - przydziel pamięć dla łańcucha pom2...
                          o wielkości odpowiadającej ilości znaków łańcucha tym1
     StrCopy(pom2,tym1); - skopiuj zawartość łańcucha tym1 do pom2
     z_nawiasu(pom2); - usuń z łańcucha pom2 skrajną parę nawiasów
     kd2:=StrLen(lancuch1^.t2); - z ilu znaków składa się łańcuch z pola t2?
     kd4:=kd2-kd1; - wylicz różnicę wielkości łańcuchów tym1 i t2 (rdzeń funkcji)
     zn2:=false; - zainicjuj zmienna logiczną zn2 w pozycji false
     if kd4=1 then zn2:=true; - jeśli przed nawiasem znajduje się jeden znak,...
                                ustaw zmienną zn2 w pozycji true
     if (kd4>1) or (lancuch1^.t2[0]='\') then - czy rdzeń funkcji istnieje?
     begin - jeśli tak, to...
```

```
GetMem(tym2,kd4*sizeof(PChar)); - przydziel pamięć...
                                dla łańcucha tym2  o wielkości odpowiedniej...
                                do wyliczonej ilości znaków rdzenia funkcji

StrLCopy(tym2,lancuch1^.t2,kd4); - skopiuj z łańcucha z pola t2…
     do tym2 wyliczoną dla rdzenia ilość znaków, zapamiętaną w zmiennej kd4

if (tym2[0]='+') or (tym2[0]='-') then - czy łańcuch tym2 ...
                                       rozpoczyna się od znaku '+' lub '– '?
begin - jeśli tak, to...
     tym1:=@tym2[1]; - przypisz łańcuchowi tym1 adres łańcucha tym2,...
                       ale od następnego znaku, z pominięciem +/–

     StrCopy(tym2,tym1) - skopiuj zawartość łańcucha tym1 do tym2
end;
for jd:=0 to 5 do ww[jd]:=#0; - „wyczyść" łańcuch lokalny ww
kd2:=0;
jd:=0;
kd3:=ord(tym2[kd2]); - odczytaj wartość pierwszego znaku…
                       z łańcucha tym2
while kd3<>0 do - wykonaj krok pętli, póki wartość odczytanego znaku...
                  nie jest równa zero
begin
     if kd3 in liczby then - czy odczytany znak znajduje się…
                             w tablicy...
                           liczby, czyli czy znak jest cyfrą?
        if jd<6 then - jeśli tak, to czy ilość odczytanych…
                       cyfr jest mniejsza niż 6?
        begin - jeśli warunki zostały spełnione, to...
             ww[jd]:=chr(kd3); - przekonwertuj z wartości na znak...
               i wpisz go do tablicy ww w miejscu wskazanym zmienną jd

             tym2[kd2]:=#0; - odczytany znak w tablicy tym2 zastąp...
                              znakiem pustym

             inc(jd) - zwiększ wartość jd o jeden dla następnego wpisu
        end else - jeśli ilość cyfr przekracza 5, to...
            begin
                 bl:=true; - ustaw zmienną błędu w pozycji true
                 blw:=5; - przypisz zmiennej stosowny kod błędu

                 FreeMem(tym2,kd4*sizeof(PChar));
                 FreeMem(pom2,kd1*sizeof(PChar));
                 zwolnij pamięć zajmowaną przez łańcuchy tym2 i pom2

                 exit - bezwzględnie opuść procedurę
            end;
     inc(kd2); - zwiększ wartość kd2 o jeden
     kd3:=ord(tym2[kd2]); - odczytaj wartość następnego znaku ...
                            z łańcucha tym2
end;
sd:=false; - zainicjuj zmienną sd wartością false
if ww[0]=#0 then - czy pierwszy znak w łańcuchu ww jest pusty?
begin - jeśli tak, to...
     if StrComp(tym2,'arcsin')=0 then - czy zawartość łańcucha ...
                                        tym2 i nazwa funkcji 'arcsin' są takie same?

     begin - jeśli tak, to...
          asi(lancuch1^.t3,pom2,lancuch2); - wywołaj...
                                             procedurę asi, która zwróci pochodną...
                                             rozpoznanej funkcji do łańcucha z pola t3
```

```
      slownik:='c'; - wpisz kod tej funkcji do zmiennej slownik
      sd:=true - funkcja została rozpoznana
end;
if not sd then - czy funkcja została już rozpoznana?
                 jeśli nie, to (jak wyżej, dla funkcji 'arccos')
   if StrComp(tym2,'arccos')=0 then
   begin
        aco(lancuch1^.t3,pom2,lancuch2);
        slownik:='a';
        sd:=true
   end;
if not sd then - (jak wyżej, dla funkcji 'arctg')
   if StrComp(tym2,'arctg')=0 then
   begin
        atg(lancuch1^.t3,pom2,lancuch2);
        slownik:='d';
        sd:=true
   end;
if not sd then - (jak wyżej, dla funkcji 'arcctg')
   if StrComp(tym2,'arcctg')=0 then
   begin
        act(lancuch1^.t3,pom2,lancuch2);
        slownik:='b';
        sd:=true
   end;
if not sd then - (jak wyżej, dla funkcji 'sin')
   if StrComp(tym2,'sin')=0 then
   begin
        bes(lancuch1^.t3,pom2,lancuch2);
        slownik:='e';
        sd:=true
   end;
if not sd then - (jak wyżej, dla funkcji 'cos')
   if StrComp(tym2,'cos')=0 then
   begin
        bec(lancuch1^.t3,pom2,lancuch2);
        slownik:='f';
        sd:=true
   end;
if not sd then - (jak wyżej, dla funkcji 'ctg')
   if StrComp(tym2,'ctg')=0 then
   begin
        bet(lancuch1^.t3,pom2,lancuch2);
        slownik:='g';
        sd:=true
   end;
if not sd then - (jak wyżej, dla funkcji 'tg')
   if StrComp(tym2,'tg')=0 then
   begin
        beg(lancuch1^.t3,pom2,lancuch2);
        slownik:='h';
        sd:=true
   end;
if not sd then - (jak wyżej, dla logarytmu naturalnego)
   if StrComp(tym2,'ln')=0 then
   begin
        lina(lancuch1^.t3,pom2,lancuch2);
        slownik:='j';
        sd:=true
   end;
```

```
      end;
```
gdy w łańcuchu ww na pozycji zero nie ma znaku pustego, to... .
```
      if not sd then - (jak wyżej, dla pierwiastka)
         if StrComp(tym2,'\')=0 then
         begin
              pet(lancuch1^.t3,pom2,lancuch2,ww);
              slownik:='k';
              sd:=true
         end;
      if not sd then - (jak wyżej, dla logarytmu o dowolnej podstawie)
         if StrComp(tym2,'log')=0 then
         begin
              loga(lancuch1^.t3,pom2,lancuch2,ww);
              slownik:='i';
              sd:=true
         end;
      FreeMem(tym2,kd4*sizeof(PChar));
      FreeMem(pom2,kd1*sizeof(PChar));
```
zwolnij pamięć zajmowaną przez łańcuchy tym2 i pom2
```
      if sd then - czy funkcja została rozpoznana?
         if lancuch1^.t3[0]='-' then - jeśli tak, to czy pierwszy znak...
                                       pochodnej rozpoczyna się od znaku minus?
            if lancuch1^.t2[0]='-' then lancuch1^.t3[0]:='+'
            jeśli tak, to jeśli pierwszy znak w łańcuchu wejściowym t2 to minus,...
            to zamień pierwszy znak pochodnej w łańcuchu z pola t3 z '– ' na '+'
            else - w przeciwnym przypadku znak '– ' w łańcuchu z pola t3...
                   pozostaje
         else gdy łańcuch z pola t3 nie rozpoczyna się od znaku minus, to...
             if lancuch1^.t2[0]='-' then lancuch1^.t3[0]:='- '
             jeśli łańcuch z pola t2 rozpoczyna się od znaku '– ' to zamień pierwszy...
             znak w łańcuchu z pola t3 z '+' na '– '
             else
      else - gdy funkcja nie została rozpoznana, to...
          begin
               bl:=true; - ustaw zmienną błędu w pozycji true
               blw:=7 - przypisz zmiennej kod błędu
          end;
end else - gdy dostarczona funkcja jest w całości zamknięta w nawiasie, to...
    begin
         FreeMem(pom2,kd1*sizeof(PChar)); - zwolnij pamięć...
                                            zajmowaną przez łańcuch pom2
         kd1:=StrLen(lancuch2); - z ilu znaków składa się łańcuch...
                                  lancuch2?
         kd2:=0; - wyzeruj zmienną kd2
         if zn2 then - czy zmienna zn2 została ustawiona w pozycji true?
         begin - jeśli tak, to...
              if lancuch1^.t2[0]='-' then kd2:=2 –jeśli ...
                 pierwszym znakiem w łańcuchu z pola t2 jest odejmowanie,...
                 wpisz do zmiennej kd2 liczbę dwa oznaczającą potrzebę...
                 dołączenia do łańcucha wynikowego, oprócz znaku...
                 odejmowania także pochodnej zamkniętej w dodatkowej...
                 parze nawiasów.
              else
              if lancuch1^.t2[0]='+' then kd2:=1 –jeśli nie,...
                 to gdy pierwszym znakiem w łańcuchu t2 jest dodawanie,...
                 wpisz do zmiennej kd2 liczbę jeden, oznaczającą potrzebę...
```

```
                        dołączenia do łańcucha wynikowego t3 znaku dodawania...
                        i znalezionej pochodnej
                    else - obecność innego znaku na początku łańcucha t2...
                           przed nawiasem otwierającym jest błędem, więc...
                    begin
                         blw:=7; - wpisz do zmiennej kodu błędu liczbę siedem
                         bl:=true - ustaw zmienną błędu w pozycji true
                    end;
                end;
                GetMem(lancuch1^.t3,(kd1+kd2)*sizeof(PChar));
                    przydziel pamięć dla łańcucha t3 o rozmiarze proporcjonalnym...
                    do wartości w zmiennych kd1 i kd2
                if kd2>0 then - czy są przewidziane dodatkowe znaki do wstawienia?
                   if kd2=1 then - jeśli tak, to czy jest to jeden dodatkowy znak?
                   begin - jeśli tak, to...
                        StrCopy(lancuch1^.t3,'+'); - wstaw na początku...
                                                     łańcucha t3 znak dodawania
                        StrCat(lancuch1^.t3,lancuch2); - dołącz ...
                                                         zawartość łańcucha lancuch2
                   end else - przeciwny przypadek wskazuje na trzy dodatkowe znaki
                       begin
                            StrCopy(lancuch1^.t3,'-'); - wstaw na…
                                                         początku łańcucha t3 znak odejmowania
                            StrCat(lancuch1^.t3,'('); - dołącz znak...
                                                        nawiasu otwierającego
                            StrCat(lancuch1^.t3,lancuch2); dołącz ...
                                                           zawartość łańcucha lancuch2
                            StrCat(lancuch1^.t3,')') dołącz znak nawiasu...
                                                     zamykającego
                       end
                else StrCopy(lancuch1^.t3,lancuch2); - jeśli nie ma ...
                                                       dodatkowych znaków do wstawienia,...
                                                       skopiuj do łańcucha t3 zawartość
                                                       łańcucha lancuch2
            end;
    end;
```

Znajdowaniem pochodnych rozpoznanych funkcji i wpisywaniem ich do łańcucha `t3`, zajmuje się jedenaście wewnętrznych procedur. Ich budowa i działanie jest zasadniczo bardzo do siebie podobne dlatego, by nie powielać opisu tych samych instrukcji, poniżej zostaną zaprezentowane tylko te, znajdujące pochodną funkcji arcsin oraz pierwiastka.

X. 8.1. Jaka jest wartość znaków w łańcuchu? Funkcja `ile`

Zadaniem funkcji jest zsumowanie wartości znaków znajdujących się w dostarczonym łańcuchu znakowym i zwrócenie tej sumy przez funkcję. Zliczane są wyłącznie znaki cyfr oraz liter. Bardzo ważną wartością dla procedur składających pochodną wynikową jest zero, dlatego od wartości każdego sczytanego znaku odejmowana jest liczba 48, która jest kodem ASCII cyfry zero. W ten sposób, dla łańcucha składającego się z dwóch lub więcej zer, funkcja zwróci wartość zero, gdyż suma zer daje w wyniku zero, inaczej niż suma dwóch lub więcej liczb 48. Znaki z łańcucha sczytywane są w pętli `while..do` do końca łańcucha.

```
function ile(const lancuch:PChar):integer;
var m1,m2,m3:integer;
begin
     m1:=0;
     m2:=0;
     m3:=ord(lancuch[m1]); - odczytaj wartość pierwszego znaku z łańcucha
     while m3<>0 do - wykonaj krok pętli, póki wartość znaku jest różna od zera
     begin
          if (m3>=48) and (m3<=57) then m2:=m2+m3-48; - dodaj...
                  wartość odczytanego znaku cyfry do zmiennej m2 i odejmij 48
          if m3>=97 then m2:=m2+m3-48; - jak wyżej, tylko dla znaków liter
          inc(m1); - zwiększ wartość m1 o jeden
          m3:=ord(lancuch[m1]); - odczytaj wartość następnego znaku
     end;
     Result:=m2 - zwróć wartość zsumowanych znaków ze zmiennej m2
end;
```

X. 8.2. Zamknięcie części pochodnej w nawiasie – procedura `w_nawias`

Zadaniem procedury jest skopiowanie zawartości jednego dostarczonego łańcucha do drugiego i dołączenie do niego skrajnej pary nawiasów. Zadanie wydaje się proste, ale wzbudzić może pewne obawy, że w łańcuchu docelowym nie ma miejsca dla dodatkowych dwóch znaków. Obawy są na szczęście bezzasadne, gdyż pamięć dla tego łańcucha została przydzielona z pewnym zapasem. Procedura składa się tylko z trzech instrukcji, które w całości wypełniają zadanie procedury.

```
procedure w_nawias(var lan1:PChar;const lan2:PChar);
begin
     StrCat(lan1,'('); - dopisz do łańcucha lan1 znak nawiasu otwierającego
     StrCat(lan1,lan2); - dopisz do łańcucha lan1 zawartość łańcucha lan2
     StrCat(lan1,')') - dopisz do łańcucha lan1 znak nawiasu zamykającego
end;
```

X. 8.3. Pochodna funkcji arcsin – procedura `asi`

Procedura znajduje pochodną funkcji `arcsin` według poniższego wzoru:

$$f(x)=\arcsin(x)\Rightarrow f'(x)=\frac{1}{\sqrt{1-x^2}}$$

By maksymalnie zmniejszyć liczby instrukcji wstawiania znaków pochodnej do łańcucha wynikowego, w samej procedurze zostały zadeklarowane dwa łańcuchy stałe składające się z niezmiennych elementów pochodnej – pierwszy łańcuch `cz1` posiada w kolejności następujące znaki: dzielenia, nawias otwierający, pierwiastek, ponownie nawias otwierający, cyfrę jeden i znak odejmowania – drugi łańcuch `cz2` posiada: znak potęgowania, cyfrę dwa i dwa znaki nawiasów zamykających.

W pierwszej instrukcji warunkowej sprawdzana jest wartość pochodnej argumentu funkcji, który znajduje się w łańcuchu `la_B` (łańcuch ten został przekazany procedurze jako

wartość, co oznacza, że w samej procedurze funkcjonuje jedynie kopia tego łańcucha, którą można modyfikować) – jeśli wynosi ona zero, dla łańcucha wynikowego `lawin` zostaje przydzielona pamięć dla jednego znaku, którym jest cyfra zero, będąca pochodną wynikową dla tego przypadku – jeśli wartość pochodnej jest większa od zera, uruchamiany jest blok instrukcji, który wypełni łańcuch wynikowy znakami pochodnej, zgodnie ze wzorem dla funkcji, którą procedura się zajmuje. Na początku wyliczana jest wstępna ilość znaków, z których będzie składała się pochodna wynikowa – jest to wyliczenie z zapasem 5 znaków. Na tę ilość składają się: długości łańcuchów wejściowych `la_A` i `la_B` zawierające odpowiednio – argument funkcji i jego pochodną, ponadto wartość pochodząca ze stałej `wst1`, która z drobnym zapasem zawiera sumę długości łańcuchów stałych. Dla tak wyliczonej ilości przydzielana zostaje pamięć dla wewnętrznego łańcucha `tym3`. Jeśli łańcuch `la_B` rozpoczyna się od znaku dodawania lub odejmowania, znak ten zostaje usunięty, natomiast do łańcucha `tym3` zostaje wstawiony znak dodawania lub odejmowania, gdy usuniętym znakiem z łańcucha `la_B` było odejmowanie. Za wstawionym znakiem koniecznie musi znaleźć się znak pusty zamykający łańcuch. Brak znaku pustego zaburzyłby proces dopisywania kolejnych znaków do łańcucha za pośrednictwem funkcji bibliotecznej `StrCat`, gdyż kolejne znaki dopisywane są przez tą funkcję od miejsca wystąpienia pierwszego znaku pustego. W kolejnej instrukcji, do łańcucha `tym3` zostaje skopiowana zawartość łańcucha `la_B`, po uprzednim uzupełnieniu w nim skrajnych nawiasów – dla tego przypadku, brak nawiasów zostaje wykryty w sytuacji występowania w łańcuchu `la_B` znaku dodawania, odejmowania, mnożenia, dzielenia oraz pierwiastka, które nie są zamknięte w nawiasie. Po uzupełnieniu łańcucha `tym3` znakami pochodnej argumentu, do łańcucha zostają dopisane znaki zawarte w łańcuchu stałym `cz1`. W następnej instrukcji, z łańcucha `la_A` zostaje usunięty pierwszy ewentualny znak dodawania lub odejmowania. W tym przypadku funkcja `czy_przeniesc_znak` została wywołana jak procedura – zwracana przez nią wartość została zignorowana. Po uzupełnieniu w łańcuchu `la_A` ewentualnego braku nawiasów (kryteria braku nawiasów są takie same jak dla łańcucha `la_B`), jego zawartość zostaje dopisana do łańcucha `tym3`. Po skopiowaniu zawartości drugiego łańcucha stałego `cz2` do `tym3` oraz wyliczeniu rzeczywistej ilości znaków zawartych w łańcuchu `tym3`, dla łańcucha wyjściowego `lawin` zostaje przydzielona pamięć a sam łańcuch zostaje uzupełniony znakami znalezionej pochodnej znajdującymi się w łańcuchu `tym3`. Ostatnią instrukcją jest zwolnienie pamięci zajmowanej przez łańcuch `tym3`.

```
procedure asi(var lawin,la_A,la_B:PChar);
const wst1=15;
      cz1:array[0..6] of char=('/','(','\','(','1','- ',#0);
      cz2:array[0..4] of char=('^','2',')',')',#0);
var tym3:PChar;
    a1,a2:integer;
begin
     if ile(la_B)=0 then - czy wartość łańcucha la_B jest zerowa?
     begin - jeśli tak, to...
          GetMem(lawin,sizeof(PChar)); - przydziel pamięć dla łańcucha ...
                  zewnętrznego lawin o rozmiarze odpowiednim dla jednego znaku
          lawin[0]:='0'; - wpisz zero – pochodną wynikową
          lawin[1]:=#0 - zamknij łańcuch
     end else - gdy wartość łańcucha la_B nie jest zerowa, to...
         begin
              a1:=wst1+StrLen(la_A)+StrLen(la_B); - wylicz...
              długość łańcucha dla pochodnej z zapasem zawartym w zmiennej wst1
              GetMem(tym3,a1*sizeof(PChar)); - przydziel pamięć...
              dla łańcucha tym3 o wielkości odpowiedniej do wyliczonej długości
```

```
                    if czy_przeniesc_znak(la_B) then tym3[0]:='-'
                    usuń znak +/– z łańcucha la_B i ustal znak rozpoczynający pochodną
                    else tym3[0]:='+';

                    tym3[1]:=#0; - zamknij łańcuch po wpisaniu +/–
                    if czy_brak_nawiasu(la_B,1) then –czy w łańcuchu...
                    la_B znajdują się znaki +/– nie zamknięte w nawiasie?

                        w_nawias(tym3,la_B) –jeśli tak, to skopiuj zawartość...
                        łańcucha la_B do tym3 zamkniętą w dodatkowej parze nawiasów

                    else StrCat(tym3,la_B); - w przeciwnym przypadku…
                                              skopiuj zawartość łańcucha la_B do tym3

                    StrCat(tym3,cz1); - dopisz do łańcucha tym3 zawartość cz1
                    czy_przeniesc_znak(la_A); - usuń znak +/– z łańcucha la_A
                    if czy_brak_nawiasu(la_A,1) then - czy w łańcuchu...
                              la_A znajdują się znaki +/– nie zamknięte w nawiasie?

                        w_nawias(tym3,la_A) –jeśli tak, to skopiuj zawartość...
                        łańcucha la_A do tym3 zamkniętą w dodatkowej parze nawiasów

                    else StrCat(tym3,la_A); - w przeciwnym przypadku
                                              skopiuj zawartość łańcucha la_A do tym3
                    StrCat(tym3,cz2); - dopisz do łańcucha tym3 zawartość cz2
                    a2:=StrLen(tym3); - z ilu znaków składa się łańcuch tym3?
                    GetMem(lawin,a2*sizeof(PChar)); - przydziel pamięć...
                        dla łańcucha zewnętrznego lawin o rozmiarze odpowiednim...
                        do wyliczonej ilości znaków zapamiętanej w zmiennej a2

                    StrCopy(lawin,tym3); - skopiuj zawartość tym3 do lawin

                    FreeMem(tym3,a1*sizeof(PChar)) - zwolnij pamięć...
                end;                                 zajmowaną przez łańcuch tym3
end;
```

X. 8.4. Pochodna pierwiastka – procedura `pet`

Zadaniem procedury jest znalezienie pochodnej pierwiastka stopnia od drugiego do dziewiątego, według następujących wzorów:

$$f(x)=\sqrt{(x)} \Rightarrow f'(x)=\frac{1}{2\cdot\sqrt{(x)}}$$

lub

$$f(x)=\sqrt[n]{(x)} \Rightarrow f'(x)=\frac{1}{n\cdot\sqrt[n]{(x)^{n-1}}}$$

Procedura otrzymuje w parametrach cztery łańcuchy znakowe, które w procedurze widoczne są pod następującymi nazwami:

- `lawin` – łańcuch wynikowy typu `PChar`, poprzedzony przedrostkiem `var`, który pozwala na jego modyfikację;

- `la_A` – łańcuch typu `PChar` zawierający znaki funkcji podpierwiastkowej;
- `la_B` – łańcuch typu `PChar` zawierający znaki pochodnej funkcji podpierwiastkowej. Łańcuch ten oraz poprzedni, są przekazane procedurze poprzez wartość, co oznacza, że w samej procedurze widoczne są jedynie ich kopie, które można modyfikować;
- `la_stopien` – łańcuch typu `Char`, poprzedzony przedrostkiem `const`, a więc tylko do odczytu. Łańcuch ten przekazuje procedurze znak stopnia pierwiastka – gdy pierwiastek jest drugiego stopnia, łańcuch ten jest pusty.

Tak jak w pozostałych procedurach wewnętrznych, zadeklarowane zostały łańcuchy stałe, składające się z niezmiennych elementów pochodnej. Rozwiązanie to pozwala ograniczyć do minimum ilość instrukcji składających pochodną wynikową. W omawianej procedurze znajdują się trzy takie łańcuchy: `cz1`, `cz2` i `cz3`.

Na początku sprawdzana jest wartość łańcucha `la_B` – jeśli funkcja `ile` zwróci wartość zero, dla łańcucha wynikowego `lawin` przydzielana zostaje pamięć dla jednego znaku. Po wpisanie znaku cyfry zero a za nim znaku pustego zamykającego łańcuch, praca procedury zostaje ukończona. Inna jest sytuacja, gdy funkcja `ile` zwróci wartość większą od zera – uruchamiane są wówczas instrukcje, które składają pochodną wynikową zgodnie ze wzorem. Na początku tego bloku instrukcji, procedura musi ustalić z pewnym zapasem, z ilu znaków będzie składała się pochodna. Ilość ta jest sumą długości łańcuchów `la_A`, `la_B` oraz stałej `wst1`, której wartość jest sumą długości łańcuchów stałych oraz naddatku. Po ustaleniu orientacyjnej liczby znaków pochodnej, dla łańcucha lokalnego `tym3` przydzielana zostaje pamięć, której wielkość jest proporcjonalna do wyliczonej długości łańcucha. Do zmiennej `a2` wczytana zostaje wartość pierwszego znaku z łańcucha `la_stopien` – gdy jest ona zerowa, zmiennej tej przypisana zostaje wartość 50, odpowiadająca kodowi ASCII cyfry dwa, oznaczającej drugi stopień pierwiastka. Kolejną instrukcją jest ustalenie znaku pochodnej – polega to na usunięciu z łańcucha `la_B` początkowego znaku dodawania lub odejmowania oraz, gdy usuniętym znakiem było odejmowanie, wpisanie tego znaku do łańcucha `tym3`. W każdym innym przypadku do łańcucha `tym3` zostaje wpisany znak dodawania. Zaraz po wpisaniu pierwszego znaku pochodnej, łańcuch zostaje zamknięty. Kolejną instrukcją jest uzupełnienie ewentualnego braku skrajnych nawiasów w łańcuchu `la_B` i dopisanie go do łańcucha `tym3`. Kryterium ustalenia braku nawiasu przez funkcję `czy_brak_nawiasu` jest obecność przynajmniej jednego ze znaków działania (mnożenie, dzielenie, dodawanie, odejmowanie) lub pierwiastka, które nie są zamknięte w nawiasie. Kryterium to podawane jest funkcji w drugim parametrze – jest nim cyfra jeden. Po wpisaniu do łańcucha `tym3` pochodnej argumentu, pora na pierwszy łańcuch stały – `cz1`. Składa się on ze znaku dzielenia i nawiasu otwierającego. Tuż za nim, do łańcucha `tym3` dopisany zostaje znak stopnia pierwiastka, jednak nie bezpośrednio a za pomocą dwuznakowego łańcucha lokalnego `st1`, do którego trafia znak zapamiętany w zmiennej liczbowej `a2` (po konwersji z postaci liczbowej na znakową) oraz znak pusty. Tak przygotowany łańcuch zostaje dopisany do łańcucha `tym3` za pomocą funkcji bibliotecznej `StrCat`. Kolejnym dopisanym elementem jest drugi łańcuch stały `cz2`, który składa się ze znaku mnożenia i pierwiastka. Od tego momentu, działanie procedury uzależnione jest od stopnia pierwiastka, którego wartość została zapamiętana w zmiennej `a2` – jeżeli pochodna znajdowana jest z pierwiastka stopnia drugiego, do łańcucha `tym3` zostają dodane: znak nawiasu otwierającego (pierwszy element funkcji podpierwiastkowej), zawartość łańcucha `la_A` oraz trzeci łańcuch stały `cz3`, składający się z dwóch znaków nawiasów zamykających. Jeśli stopień pierwiastka jest większy od drugiego, do łańcucha `tym3` zostaje ponownie dopisana zawartość łańcucha `st1` a za nim funkcja podpierwiastkowa, ale w tym przypadku musi ona być zamknięta w nawiasie, by za nią można było umieścić funkcję potęgującą. Zatem, nim łańcuch `la_A` zostanie dopisany do łańcucha `tym3`, zostanie on zba-

dany, czy znajduje się w nim choć jeden z możliwych znaków działania (dodawanie, odejmowanie, mnożenie, dzielenie, pierwiastek), który nie jest zamknięty w nawiasie – jeśli tak, łańcuch ten zostaje uzupełniony skrajną parą nawiasów. Kolejnymi elementami dodanymi do łańcucha `tym3`, są: znak potęgowania, zmodyfikowany łańcuch `st1` zawierający pomniejszony znak potęgowania oraz trzeci łańcuch stały `st3` kończący pochodną wynikową.

Ostatnią czynnością procedury, wspólną dla obydwu przypadków pochodnej, jest zliczenie ilości znaków znajdujących się w łańcuchu `tym3`, przydzielenie pamięci dla łańcucha wynikowego `lawin` o wielkości odpowiedniej do wyliczonej ilości znaków, skopiowanie zawartości łańcucha `tym3` do `lawin` oraz uwolnienie pamięci zajmowanej przez łańcuch lokalny `tym3`.

```
procedure pet(var lawin:PChar;la_A,la_B:PChar;
              const la_stopien:array of char);

const wst1=20;
      cz1:array[0..2] of char=('/','(',#0);
      cz2:array[0..2] of char=('*','\',#0);
      cz3:array[0..2] of char=(')',')',#0);
var tym3:PChar;
    st1:array[0..1] of char;
    a1,a2:integer;
    s:boolean;
begin
     if ile(la_B)=0 then - czy wartość łańcucha la_B jest zerowa?
     begin - jeśli tak, to...
          GetMem(lawin,sizeof(PChar)); - przydziel pamięć dla ...
              łańcucha lawin o rozmiarze odpowiednim dla jednego znaku
          lawin[0]:='0'; - wpisz znak zero – pochodną wynikową
          lawin[1]:=#0 - zamknij łańcuch
     end else - gdy wartość łańcucha la_B nie jest zerowa, to...
         begin
              a2:=ord(la_stopien[0]); - jaka jest wartość pierwszego...
                                        znaku w łańcuchu la_stopien?
              if a2=0 then a2:=50; - jeśli łańcuch jest pusty,
                    przypisz zmiennej a2 wartość w kodzie ASCII cyfry dwa
              a1:=wst1+StrLen(la_A)+StrLen(la_B); - wylicz...
                        przybliżoną ilość znaków pochodnej wynikowej
              GetMem(tym3,a1*sizeof(PChar)); - przydziel pamięć...
              dla łańcucha tym3 o wielkości odpowiedniej do ilości znaków ...
              zapamiętanych w zmiennej a1
              if czy_przeniesc_znak(la_B) then tym3[0]:='-'
              else tym3[0]:= '+';
                       usuń ewentualny znak +/– z łańcucha la_B i ustal znak
                       rozpoczynający pochodną
              tym3[1]:=#0; - zamknij łańcuch tym3
              if czy_brak_nawiasu(la_B,1) then - jeśli w łańcuchu...
                                    la_B znajduje się choć jeden znak działania...
                                    nie zamknięty w nawiasie...
                 w_nawias(tym3,la_B) - skopiuj zawartość łańcucha la_B...
                                    do tym3, uzupełniając go skrajną parą nawiasów
              else StrCat(tym3,la_B); - gdy nawiasy nie są konieczne,...
                                    dopisz zawartość łańcucha la_B do tym3
```

```
                    StrCat(tym3,cz1); - dopisz zawartość łańcucha cz1 do tym3
                    st1[0]:=chr(a2); - wpisz do łańcucha st1 znak stopnia...
                                       pierwiastka
                    st1[1]:=#0; - zamknij łańcuch st1
                    StrCat(tym3,st1); - dopisz łańcuch st1 do tym3
                    StrCat(tym3,cz2); - dopisz łańcuch cz2 do tym3
                    s:=false; - zainicjuj zmienną logiczną s wartością false
                    if a2>50 then - czy stopień pierwiastka jest większy niż dwa?
                    begin - jeśli tak, to...
                          StrCat(tym3,st1); - dopisz łańcuch st1 do tym3
                          s:=true - ustaw zmienną logiczną s w pozycji true
                    end;
                    StrCat(tym3,'('); - dopisz do tym3 znak nawiasu…
                                        otwierającego
                    if s then - czy zmienna s ma wartość true?
                    begin - jeśli tak, to...
                          if czy_brak_nawiasu(la_A,1) then
                             w_nawias(tym3,la_A)
                           zamknij zawartość łańcucha la_A w parze nawiasów i dopisz ...
                           jego zawartość do tym3
                          else StrCat(tym3,la_A); - jeśli łańcuch la_A...
                           posiada wymaganą parę nawiasów, dopisz go do tym3
                          StrCat(tym3,'^'); - do tym3 dopisz znak potęgowania
                          st1[0]:=chr(a2-1); - zmodyfikuj pierwsze pole...
                          łańcucha st1, wpisując do niego znak, którego wartością jest...
                          stopień pierwiastka pomniejszony o jeden
                          st1[1]:=#0; - zamknij łańcuch st1
                          StrCat(tym3,st1) - dopisz zawartość łańcucha st1...
                                             do tym3
                    end else StrCat(tym3,la_A); - gdy zmienna s ma wartość ...
                                  false, dopisz zawartość łańcucha la_A do tym3
                    StrCat(tym3,cz3); - dopisz zawartość łańcucha cz3 do tym3
                    a2:=StrLen(tym3); - z ilu znaków składa się łańcuch tym3?
                    GetMem(lawin,a2*sizeof(PChar)); - przydziel pamięć ...
                     dla łańcucha lawin o wielkości odpowiedniej do ilości znaków...
                     zapamiętanych w zmiennej a2
                    StrCopy(lawin,tym3); - przenieś zawartość łańcucha tym3...
                                                                  do lawin
                    FreeMem(tym3,a1*sizeof(PChar)) - zwolnij pamięć...
                                                     zajmowaną przez łańcuch tym3
              end;
end;
```

X. 9. Pochodna funkcji elementarnych – procedura `trzy`

Działanie tej procedury jest nieco inne niż wcześniej opisywanych procedur `jeden` oraz `dwa`. Różnica polega na tym, że procedura `trzy` znajduje pochodną nie jednej lecz dwóch funkcji połączonych znakami mnożenia, dzielenia lub potęgowania. W związku z tym, inna jest technika znajdowania pochodnej, która nie opiera się już o schemat pochodnych funkcji elementarnych a o fundamentalne zasady różniczkowania dwóch i więcej funkcji po-

łączonych wyżej wymienionymi znakami działania. Często zdarza się, że funkcja składa się z wielu części łączonych znakami mnożenia, dzielenia czy potęgowania – w takiej sytuacji procedura wywoływana jest wielokrotnie, dla każdego znaku działania oddzielnie. Kluczowym przy każdym wywołaniu procedury jest określenie, który znak działania ma być brany pod uwagę oraz która funkcja jest tą przed tym znakiem, a która za nim. Można by przekazać procedurze dwa łańcuchy znaków oddzielnie, jak również kluczowy znak działania, jednak wygodniejszym rozwiązaniem okazało się przekazanie procedurze miejsca osadzenia znaku działania w łańcuchu, dzięki czemu można bez problemu znaleźć ten znak oraz rozdzielić funkcję na dwie części. Owe miejsce zostało zapamiętane w polu `jj` należącym do elementu `bloki`, którego adres został procedurze przekazany w parametrze o nazwie `lancuch`. Poza wspomnianym adresem, procedura otrzymuje także łańcuch o nazwie `lancuch2` zawierający znaki pochodnej funkcji znajdującej się za znakiem rozdzielającym (pochodna funkcji przed znakiem rozdzielającym znajduje się w polu `t3` dostarczonego elementu `bloki`).

Pierwszą czynnością procedury jest sprawdzenie, czy pole `t3` oraz łańcuch `lancuch2` nie posiadają adresów pustych – obecność choć w jednym łańcuchu adresu pustego powoduje opuszczenie procedury z błędem. Jeśli łańcuchy nie są puste, dla trzech łańcuchów lokalnych przydzielona zostaje pamięć o wielkościach odpowiednich do rozmiarów łańcuchów, z których znaki będą przenoszone. Dodatkowo, by umożliwić ich bezpieczną modyfikację, rozmiary te zostały powiększone o niewielki naddatek. Do pierwszego łańcucha lokalnego `tym1` została przeniesiona zawartość łańcucha `t3`, pozostałe dwa są na razie puste. Przygotowanie łańcuchów lokalnych pozwala na rozdzielenie funkcji zawartej w polu `t2` na dwie części. Nim to jednak nastąpi, z pola `jj` do zmiennej `rx1` zostaje przepisana wartość, która wskaże miejsce rozdziału funkcji. Na jej podstawie, z łańcucha `t2` do `tym2` zostaje skopiowanych dokładnie tyle znaków, na ile wskazuje ta wartość. Skopiowanie funkcji za znakiem rozdziału polega na wczytaniu do zmiennej łańcuchowej `pom4` adresu łańcucha `t2`, ale od następnego znaku za znakiem rozdziału i skopiowanie zawartości łańcucha `pom4` do `tym3`. W wyniku opisanych czynności, łańcuch `tym2` będzie posiadał znaki funkcji rozpoczynające się od początku łańcucha `t2` do znaku poprzedzającego znak rozdzielający, natomiast łańcuch `tym3` – funkcję rozpoczynającą się od następnego znaku za znakiem rozdzielającym do końca łańcucha `t2`. Z uzupełnionych znakami łańcuchów lokalnych zostają usunięte skrajne nawiasy, czyli pierwszy znak nawiasu otwierającego i ostatni znak nawiasu zamykającego. Czynności tej dokonuje procedura `z_nawiasu`.

```
procedure trzy(const lancuch:bloki;const lancuch2:PChar;
               var slownik:char);
const zapas=10;
      log3:array[0..2] of Char=('l','n',#0);
var tym1,tym2,tym3,pom3,pom4:PChar;
    tx1,tx2,rx1,rx2,rxp,zp1,vb1:integer;
    limi3:byte;
    przl:boolean;
begin
     slownik:=#0; - zainicjuj zmienną slownik znakiem pustym, na wypadek błędu
     if lancuch^.t3=nil then - czy pole t3 zawiera adres pusty?
     begin - jeśli tak, to...
          bl:=true;  - ustaw zmienną błędu
          blw:=10; - zmiennej kodu błędu przypisz stosowny numer
          exit - opuść procedurę
     end;
     if lancuch2=nil then - czy łańcuch lancuch2 zawiera adres pusty?
     begin - jeśli tak, to...
          bl:=true; - ustaw zmienną błędu w pozycji true
```

```
            blw:=10; - przypisz zmiennej kodu błędu – liczbę 10
            exit - bezwzględnie opuść procedurę
        end;
        tx2:=StrLen(lancuch^.t3); - określ długość łańcucha w polu t3
        tx1:=tx2+zapas; - określ długości łańcucha lokalnego, uwzględniając…
                          długość łańcucha z pola t3 oraz zapas
        GetMem(tym1,tx1*sizeof(PChar)); - przydziel pamięć dla łańcucha...
                    tym1 o wielkości odpowiedniej do wyliczonej dla niego długości
        StrCopy(tym1,lancuch^.t3); - skopiuj zawartość łańcucha x pola t3 do tym1
        FreeMem(lancuch^.t3,tx2*sizeof(PChar)); - zwolnij pamięć ...
                                                  zajmowaną przez łańcuch z pola t3
        tx2:=StrLen(lancuch^.t2)+zapas; - określ długość łańcuchów lokalnych...
                    równej sumie zajętości łańcucha z pola t2 oraz naddatku w stałej zapas
        GetMem(tym2,tx2*sizeof(PChar)); } przydziel pamięć dla łańcuchów lokalnych...
        GetMem(tym3,tx2*sizeof(PChar)); } tym2 i tym3 o wielkościach stosownych...
                                         do wyliczonych dla nich liczby znaków
        rx1:=lancuch^.jj; - wczytaj wartość miejsca rozdziału funkcji z pola jj
        zp1:=ord(lancuch^.t2[rx1]); - zapamiętaj wartość znaku rozdzielającego…
                                      z miejsca w polu t2 wskazanego zmienną rx1
        StrLCopy(tym2,lancuch^.t2,rx1); - skopiuj z łańcucha z pola t2...
                do łańcucha tym2 dokładną ilość znaków wskazaną w zmiennej rx1
        inc(rx1); - zwiększ wartość w zmiennej rx1 o jeden
        pom4:=@lancuch^.t2[rx1]; - przypisz zmiennej pom4 adres pola t2 ...
                                   od miejsca wskazanego zmienną rx1
        StrCopy(tym3,pom4); - skopiuj zawartość łańcucha pom4 do tym3
        pom4:=nil; - przypisz zmiennej łańcuchowej pom4 adres pusty
        z_nawiasu(tym1);
        z_nawiasu(tym2); } usuń skrajną parę nawiasów z łańcuchów tym1. tym2 i tym3
        z_nawiasu(tym3);
        rx1:=wartosc(tym1); - jaką wartość posiada łańcuch tym1?
        rx2:=wartosc(lancuch2); - jaką wartość posiada łańcuch lancuch2?
        . . .
end;
```

Z tak przygotowanych łańcuchów można już składać pochodną wynikową. Nim to jednak nastąpi, trzeba zidentyfikować znak rozdzielający funkcje oraz, na podstawie wartości pochodnych przed i za znakiem rozdzielającym, wybrać metodę układania pochodnej. Wybór metody został uwarunkowany gąszczem instrukcji warunkowych, które zaprezentowano w poniższym, uproszczonym kodzie programu:

```
if zp1=94 then - czy znakiem rozdzielający jest potęgowaniem?
   if rx1=0 then - jeśli tak, to czy pochodna przed potęgowaniem jest zerem?
      if rx2=0 then - jeśli tak, to czy pochodna za potęgowaniem jest zerem?
      begin - jeśli tak, to...
              zrealizuj wzór „A”
                  f(x)=n^m ⇒ f'(x)=0
      end else - gdy wartość pochodnej za potęgowaniem nie jest zerowa, to...
         begin
                zrealizuj wzór „B”
                  f(x)=n^g(x) ⇒ f'(x)=n^g(x)·[g(x)]'·ln(n)
          end
   else - gdy wartość pochodnej przed potęgowaniem nie jest zerowa, to...
```

```
if rx2=0 then - czy wartość pochodnej za potęgowaniem jest zerowa?
   if rx1=1 then - jeśli tak, to czy wartość pochodnej…
                   przed potęgowaniem jest jedynką?
      if czy_policzalne(tym3) then - jeśli tak, to czy funkcję za...
      potęgowaniem można policzyć tak, by w wyniku była liczba całkowita?
      begin - jeśli tak, to...
```

zrealizuj wzór „C”

$$f(x)=x^n \Rightarrow f'(x)=n\cdot x^{n-1}$$

```
      end else - gdy funkcji za potęgowaniem nie można policzyć, to...
          begin
```

zrealizuj wzór „D”

$$f(x)=x^n \Rightarrow f'(x)=x^n\cdot\frac{n}{x}$$

```
          end
   else - gdy wartość pochodnej przed potęgowaniem nie jest zerem ani jedynką, to...
   if czy_policzalne(tym3) then - czy funkcję za potęgowaniem...
                  można policzyć tak, by w wyniku była liczba całkowita?
   begin - jeśli tak, to...
```

zrealizuj wzór „E”

$$f(x)=g(x)^n \Rightarrow f'(x)=n\cdot g(x)^{n-1}\cdot[g(x)]'$$

```
   end else - gdy funkcji za potęgowaniem nie można policzyć, to...
       begin
```

zrealizuj wzór „F”

$$f(x)=g(x)^n \Rightarrow f'(x)=g(x)^n\cdot n\cdot\frac{[g(x)]'}{g(x)}$$

```
       end
else - gdy wartości pochodnych przed i za potęgowaniem są większe od zera, to...
begin
```

zrealizuj wzór „G”

$$f(x)=g(x)^{u(x)} \Rightarrow f'(x)=g(x)^{u(x)}\cdot[u(x)]'\cdot\ln[g(x)]+g(x)^{u(x)}\cdot u(x)\cdot\frac{[g(x)]'}{g(x)}$$

```
end;
if (zp1=42) or (zp1=47) then - czy znak rozdzielający jest mnożeniem
                               lub dzieleniem?
   if rx1=0 then - jeśli tak, to...
           czy wartość pochodnej przed mnożeniem lub dzieleniem jest zerem?
      if rx2=0 then - jeśli tak, to...
           czy wartość pochodnej za mnożeniem lub dzieleniem jest zerem?
      begin - jeśli tak, to...
```

zrealizuj jeden ze wzorów „H”

$$f(x)=m\cdot n \Rightarrow f'(x)=0$$

$$f(x)=\frac{m}{n} \Rightarrow f'(x)=0$$

```
      end else - gdy wartość pochodnej za mnożeniem lub dzieleniem…
                 nie jest zerowa, to...
          begin
```

zrealizuj jeden ze wzorów „I”

$$f(x)=n\cdot g(x)\Rightarrow f'(x)=n\cdot[g(x)]'$$

$$f(x)=\frac{n}{g(x)}\Rightarrow f'(x)=-n\cdot\frac{[g(x)]'}{g(x)^2}$$

```
        end
else - gdy wartość pochodnej przed mnożeniem lub dzieleniem nie jest zerowa, to...
begin
```

zrealizuj jeden ze wzorów „J”

$$f(x)=g(x)\cdot u(x)\Rightarrow f'(x)=[g(x)]'\cdot u(x)+g(x)\cdot[u(x)]$$

$$f(x)=\frac{g(x)}{u(x)}\Rightarrow f'(x)=\frac{[g(x)]'\cdot u(x)-g(x)\cdot[u(x)]'}{u(x)^2}$$

```
end;
```

Instrukcje budujące pochodną na podstawie zaprezentowanych wyżej wzorów matematycznych są do siebie podobne. Zarówno pod względem budowy, jak i sposobu działania, dlatego opisane zostaną tylko te, które pozwolą w pełni zrozumieć istotę budowania pochodnej oraz celowość doboru wykorzystanych instrukcji.

Wzór *„A”* realizowany jest wówczas, gdy pochodna przed i za znakiem potęgowania jest zerowa, czyli gdy funkcja `wartosc` zwróciła wartości zero z łańcuchów `tym1` oraz `lancuch2`. W tej sytuacji żaden łańcuch nie będzie brał udziału w budowaniu pochodnej. Dla łańcucha wynikowego `t3` przydzielana jest pamięć tylko dla dwóch znaków – pierwszym wpisanym znakiem jest dodawanie lub, gdy jeden z łańcuchów pochodnej rozpoczyna się od znaku odejmowania – odejmowanie, natomiast drugim znakiem jest cyfra zero. Po wstawieniu znaku pochodnej, łańcuch zostaje zamknięty poprzez dopisanie znaku pustego na końcu łańcucha. Ostatnią czynnością tego bloku jest wstawienie odpowiedniego znaku do zmiennej `slownik` będącego kodem zastosowanej reguły różniczkowania.

```
GetMem(lancuch^.t3,2*sizeof(PChar)); - przydziel pamięć o wielkości…
                                       odpowiedniej dla dwóch znaków typu PChar
limi3:=0; - wyzeruj zmienną zliczającą znaki odejmowania
if lancuch2[0]='-' then inc(limi3); - jeśli pierwszym znakiem w łańcuchu...
                                     lancuch2 jest odejmowanie, to zlicz ten znak w zmiennej limi3
if tym1[0]='-' then inc(limi3); - jeśli pierwszym znakiem w łańcuchu tym1...
                                  jest odejmowanie, to zlicz ten znak w zmiennej limi3
if (limi3 mod 2)>0 then lancuch^.t3[0]:='-' jeśli reszta z dzielenia przez dwa...
          wartości w zmiennej limi3 jest większa od zera, czyli ilość zliczonych znaków ...
          odejmowania jest liczbą nieparzystą, wpisz do łańcucha wynikowego znak minus,...
else lancuch^.t3[0]:='+'; - w przeciwnym przypadku wpisz znak plus
lancuch^.t3[1]:='0'; - za wprowadzonym znakiem +/– wpisz znak cyfry zero
lancuch^.t3[2]:=#0; - zamknij łańcuch
slownik:='8'; - wpisz kod znakowy zastosowanej metody różniczkowania...
                do zmiennej slownik
```

Wzór *„E”* realizowany jest wówczas, gdy spełnione są następujące warunki:

a. znakiem rozdzielającym dwie funkcje jest potęgowanie;
b. wartość łańcucha `tym1` (pochodna funkcji przed znakiem potęgowania) jest większa od jedności, co oznacza, że funkcja przed znakiem potęgowania nie jest liczbowa oraz nie składa się wyłącznie ze zmiennej x;

c. wartość łańcucha `lancuch2` (pochodna funkcji za znakiem potęgowania) jest równa zero;
d. funkcja za znakiem potęgowania jest liczbą całkowitą, ułamkiem zwykłym, ułamkiem dziesiętnym lub połączeniem wszystkich wymienionych możliwości.

Do zmiennej `slownik` wpisany zostaje znak identyfikujący tę regułę różniczkowania. Na podstawie wzoru matematycznego dla tego przypadku wyliczona zostaje orientacyjna liczba znaków, z których będzie składała się pochodna, uwzględniając przy tym pewien zapas na możliwe dopisywanie nawiasów czy innych znaków, których obecność na tym etapie budowania pochodnej nie jest jeszcze znana. Następnie przydzielona zostaje pamięć dla tymczasowego łańcucha wynikowego `pom3`, której rozmiar jest odpowiedni do wyliczonej ilości znaków. Wyzerowany zostaje licznik znaków odejmowania – `limi3`, na postawie którego ustalony będzie początkowy znak pochodnej. Kolejną instrukcją jest przydzielenie pamięci dla następnego łańcucha lokalnego `pom4` o rozmiarze równym rozmiarowi łańcucha `tym3`. Po skopiowaniu zawartości łańcucha `tym3` do `pom4`, oba łańcuchy wypełnione zostają znakami funkcji za znakiem potęgowania. Utworzenie bliźniaczego łańcucha jest niezbędne, ponieważ w łańcuchu wynikowym mają znaleźć się zarówno funkcja za znakiem potęgowania w niezmienionej postaci oraz ta sama funkcja, ale zmodyfikowana przez odjęcie jedynki od zawartej w niej liczby. Jeśli łańcuch `pom4` rozpoczyna się od znaku dodawania lub odejmowania, znak ten zostaje z łańcucha usunięty oraz, w przypadku znaku odejmowania, zostaje on zliczony w zmiennej `limi3`.

Układanie pochodnej w łańcuchu `pom3` rozpoczyna się od wstawienia znaku dodawania a za nim znaku pustego. Za wstawionym znakiem dopisana zostaje zawartość łańcucha `pom4` a za nim znak mnożenia. Po usunięciu istniejącego znaku dodawania lub odejmowania z początku łańcucha `tym2` oraz uzupełnieniu go skrajnymi nawiasami, jego zawartość zostaje dopisana do łańcucha `pom3`. W tym przypadku, usunięty znak odejmowania nie został zliczony w zmiennej `limi3` – gdyby funkcja przed znakiem potęgowania rozpoczynała się od znaku odejmowania, wówczas znak ten pojawiłby się także w jej pochodnej dlatego, by nie zliczać tego znaku dwukrotnie, został on tylko usunięty z łańcucha. Procedura `minus_jeden`, która zostaje wywołana, ma za zadanie dopisanie do łańcucha `pom3` znaku potęgowania a za nim, zmodyfikowanego łańcucha `tym3`. Postacią wyjściową łańcucha `tym3` są znaki reprezentujące liczbę całkowitą, dziesiętną lub ułamek zwykły, uzyskane w wyniku konwersji na liczbę, odjęciu od niej cyfry jeden i ponownej konwersji do łańcucha typu `Pchar`. Po opuszczeniu procedury, do łańcucha `pom3` dopisany zostaje znak mnożenia a za nim zawartość łańcucha `tym1` składającego się ze znaków pochodnej przed znakiem potęgowania – przedtem łańcuch musi zostać przygotowany by mógł zostać skopiowany, polega to na usunięciu z niego początkowego znaku dodawania lub odejmowania, a gdy usuniętym znakiem było odejmowanie, zliczenie tego znaku w zmiennej `limi3` oraz, gdy w łańcuchu znajdują się nieujęte w nawiasie znaki dodawania czy odejmowania, łańcuch ten zostaje uzupełniony o skrajną parę nawiasów. Pozostaje jeszcze ustalenie początkowego znaku pochodnej – polega to na zbadaniu, czy wartość mieszcząca się w zmiennej `limi3` jest parzysta czy też nie. Jeśli w wyniku dzielenia przez dwa owej wartości, reszta z dzielenia będzie większa od zera, oznacza to, że wartość jest nieparzysta i początkowy znak dodawania w łańcuchu `pom3` zostaje zamieniony na odejmowanie. Pochodna dla tego przypadku jest już gotowa, potrzeba tylko zliczenia znaków znajdujących się w łańcuchu `pom3`, przydzielenia pamięci dla łańcucha wynikowego `t3` o rozmiarze odpowiednim do zliczonej ilości znaków, skopiowania zawartości łańcucha `pom3` do `t3` oraz zwolnienia pamięci zajmowanej przez łańcuchy `pom3` i `pom4`.

```
slownik:='2'; - uzupełnij zmienną slownik znakiem identyfikującym ...
                                        wybraną metodę różniczkowania

rxp:=2*StrLen(tym3)+StrLen(tym2)+StrLen(tym1)+zapas; - ustal przybliżoną...
                                        ilość znaków, z których będzie składała się pochodna

GetMem(pom3,rxp*sizeof(PChar)); - przydziel pamięć dla łańcucha pom3

limi3:=0; - wyzeruj licznik znaków odejmowania
vb1:=StrLen(tym3); - z ilu znaków składa się łańcuch tym3?
GetMem(pom4,vb1*sizeof(PChar)); - przydziel pamięć dla łańcucha pom4
StrCopy(pom4,tym3); - skopiuj zawartość łańcucha tym3 do pom4
if czy_przeniesc_znak(pom4) then inc(limi3); - jeśli łańcuch pom4 zaczyna ...
  się od znaku '+' lub '– 'i można go usunąć, usuń go z łańcucha a znak '– ' zlicz w zmiennej limi3

pom3[0]:='+'; - wpisz do łańcucha pom3 pierwszy znak '+'
pom3[1]:=#0; - zamknij łańcuch
StrCat(pom3,pom4); - dopisz do łańcucha pom3 zawartość łańcucha pom4
StrCat(pom3,'*'); - dopisz do łańcucha pom3 znak mnożenia
czy_przeniesc_znak(tym2); - jeśli łańcuch tym2 zaczyna się od znaku '+' lub '– ' ...
                                        i można go usunąć, usuń go

if czy_brak_nawiasu(tym2,1) then w_nawias(tym2); - jeśli w łańcuchu
znajdują się znaki działania nie zamknięte w nawiasie, uzupełnij łańcuch o skrajną parę nawiasów

StrCat(pom3,tym2); - dopisz zawartość łańcucha tym2 do pom3
minus_jeden(pom3,tym3); - dopisz do łańcucha pom3 znak potęgowania oraz łańcuch ...
       tym3, po odjęciu jedynki od wartości, którą reprezentują znaki cyfr zawarte w łańcuchu

StrCat(pom3,'*'); - dopisz do łańcucha pom3 znak mnożenia
if czy_przeniesc_znak(tym1) then inc(limi3); - jeśli łańcuch tym1 zaczyna ...
  się od znaku '+' lub '– ' i można go usunąć, usuń go z łańcucha a znak '– ' zlicz w zmiennej limi3

if czy_brak_nawiasu(tym1,0) then w_nawias(tym1); - jeśli w łańcuchu tym1...
    znajdują się znaki '+' lub '– ' nieujęte w nawiasie, uzupełnij łańcuch o skrajną parą nawiasów

StrCat(pom3,tym1); - dopisz zawartość łańcucha tym1 do pom3
if (limi3 mod 2)>0 then pom3[0]:='-'; - jeśli reszta z dzielenia przez dwa...
      wartości zawartej w zmiennej limi3jest większa od zera (wartość jest nieparzysta),...
      zamień początkowy znak w łańcuchu z '+' na '– '

vb1:=StrLen(pom3); - z ilu znaków składa się łańcuch pom3?
GetMem(lancuch^.t3,vb1*sizeof(PChar)); - przydziel pamięć dla łańcucha z pola t3
StrCopy(lancuch^.t3,pom3); - skopiuj zawartość łańcucha pom3 do łańcucha z pola t3
FreeMem(pom3,rxp*sizeof(PChar)); - zwolnij pamięć zajmowaną przez łańcuch pom3
FreeMem(pom4,zp1*sizeof(PChar)) - zwolnij pamięć zajmowaną przez łańcuch pom4
```

Gdy funkcja za znakiem potęgowania nie jest liczbowa (może być nią znak litery 'π' lub 'e') bądź też zawarte w niej liczby łączone są znakami działania, nie można z niej utworzyć funkcji potęgującej, której wartość mogłaby być pomniejszona o jeden. W takiej sytuacji zastosowanie wzoru *„E"* jest niemożliwe, za to właściwy dla takiej funkcji jest wzór *„F"*.

Zmiennej `slownik` przypisany zostaje kod znakowy, odpowiedni do zastosowanej reguły różniczkowania. W zmiennej `rxp` zliczona zostaje przybliżona ilość znaków, z których składać się będzie pochodna, uwzględniając przy tym niewielki zapas. Na jej podstawie, dla tymczasowego łańcucha wynikowego `pom3` przydzielona zostaje pamięć. Ponieważ funkcja za znakiem potęgowania będzie wykorzystana dwukrotnie, trzeba było utworzyć jej kopię w dodatkowym łańcuchu lokalnym `pom4`, dla którego przydzielana zostaje pamięć o wielkości równej długości łańcucha `tym3`, powiększonej o dwa znaki na dodatkową parę nawiasów. Z łańcucha usunięty zostaje ewentualny, początkowy znak dodawania lub odejmowania, a usunięty znak odejmowania zostaje zliczony w zmiennej `limi3`.

Pierwszym wprowadzonym znakiem do łańcucha pom3 jest dodawanie. Po usunięciu z łańcucha tym2 możliwego znaku dodawania lub odejmowania oraz uzupełnieniu go skrajną parą nawiasów, zostaje on dopisany do łańcucha tym3. W tym przypadku, usunięty znak odejmowania nie został zliczony. Kolejnym dopisanym elementem do łańcucha pom3 jest znak potęgowania a za nim łańcuch tym3, po uprzednim uzupełnieniu go skrajną parą nawiasów. Do tego momentu, w łańcuchu pom3 znajdują się: znak dodawania, funkcja przed znakiem potęgowania, znak potęgowania i funkcja za znakiem potęgowania. Kolejnymi elementami dopisanymi do łańcucha pom3 są: znak mnożenia; zawartość łańcucha pom4, czyli kopii funkcji za znakiem potęgowania zamkniętej w nawiasie oraz kolejny znak mnożenia. Ponieważ za znakiem mnożenia ma znaleźć się ułamek składający się z pochodnej przed znakiem potęgowania i jej funkcji, korzystnym okazało się zamknięcie tego ułamka w nawiasie, dlatego za wstawionym znakiem mnożenia zostaje dodany znak nawiasu otwierającego, a po nim zawartość łańcucha tym1 (pochodna przed znakiem potęgowania), po wcześniejszym usunięciu z niego znaku dodawania lub odejmowania, zliczeniu usuniętego znaku odejmowania w zmiennej limi3 oraz dodaniu do niego skrajnej pary nawiasów, pod warunkiem, że w łańcuchu znajdują się nie zamknięte w nawiasie znaki dodawania lub odejmowania. Za ostatnim znakiem pochodnej, do łańcucha pom3 dodany zostaje znak dzielenia, a za nim zawartość łańcucha tym2 uzupełnionego parą nawiasów tak, by pierwszym znakiem był nawias otwierający a ostatnim – zamykający. W ten sposób, mianownik ułamka ma wymaganą parę nawiasów. Ostatnim dodanym znakiem jest nawias zamykający, zamykający kompletny ułamek. Pozostaje jeszcze ustalenie początkowego znaku pochodnej – polega to na sprawdzeniu, czy w liczniku znaków odejmowania limi3 znajduje się liczba parzysta czy nieparzysta – przy liczbie nieparzystej, znak z początku łańcucha pom3 zostaje zmieniony z dodawania na odejmowanie. Ostatnią czynnością przed zwróceniem pochodnej do łańcucha wyjściowego t3 jest zliczenie rzeczywistej ilości znaków wprowadzonych do łańcucha pom3, przydzielenie pamięci dla łańcucha z pola t3 o wielkości odpowiedniej do zliczonej ilości znaków, skopiowanie zawartości łańcucha pom3 do t3 i zwolnienie pamięci zajmowanej przez łańcuchy pom3 i pom4.

Ustalenie początkowego znaku pochodnej jest w tym przypadku dość kłopotliwe, ponieważ przed znakiem potęgowania znajduje się funkcja, której pierwszy znak dodawania czy odejmowania nie zawsze jest zgodny z pierwszym znakiem jej pochodnej. Aby pełna pochodna zawarta w łańcuchu pom3 rozpoczynała się od właściwego znaku, do jego ustalenia należy uwzględniać tylko pierwszy znak pochodnej sprzed znaku potęgowania, czyli z łańcucha tym1.

```
slownik:='3'; - uzupełnij zmienną slownik znakiem identyfikującym ...
                                        wybraną metodę różniczkowania

rxp:=2*StrLen(tym3)+2*StrLen(tym2)+StrLen(tym1)+zapas; - ustal...
                    przybliżoną liczbę znaków z których będzie składała się pochodna

GetMem(pom3,rxp*sizeof(PChar)); - przydziel pamięć dla łańcucha pom3
pom3[0]:='+'; - wpisz do łańcucha pom3 pierwszy znak '+'
pom3[1]:=#0; - zamknij łańcuch
limi3:=0; - wyzeruj licznik znaków odejmowania
zp1:=StrLen(tym3)+2; - do liczby znaków w łańcuchu tym3 dodaj 2
GetMem(pom4,zp1*sizeof(PChar)); - przydziel pamięć dla łańcucha pom4
StrCopy(pom4,tym3); - skopiuj zawartość łańcucha tym3 do pom4
if czy_przeniesc_znak(pom4) then inc(limi3); - usuń pierwszy znak +/– ...
          z łańcucha pom4 oraz zlicz usunięty znak odejmowania w zmiennej limi3

czy_przeniesc_znak(tym2); - jeśli to możliwe, usuń początkowy znak +/– ...
                                        z łańcucha tym2
```

```
if czy_brak_nawiasu(tym2,2) then w_nawias(tym2); - uzupełnij łańcuch...
```
tym2 skrajną parę nawiasów, gdy znajdują się w nim znaki '*' '+' '– ' '/' nie zamknięte w nawiasie

```
StrCat(pom3,tym2); - dopisz zawartość łańcucha tym2 do pom3
StrCat(pom3,'^'); - do łańcucha pom3 dopisz znak potęgowania
if czy_brak_nawiasu(tym3,2) then w_nawias(tym3); - uzupełnij łańcuch...
```
tym3 skrajną parę nawiasów, gdy znajdują się w nim znaki '*' '+' '– ' '/' nie zamknięte w nawiasie

```
StrCat(pom3,tym3); - do łańcucha pom3 dopisz zawartość łańcucha tym3
StrCat(pom3,'*'); - do łańcucha pom3 dopisz znak mnożenia
if czy_brak_nawiasu(pom4,0) then w_nawias(pom4); - uzupełnij łańcuch...
```
pom4 skrajną parą nawiasów, gdy znajdują się w nim znaki +/– nie zamknięte w nawiasie

```
StrCat(pom3,pom4); - dopisz do łańcucha pom3 zawartość łańcucha pom4
StrCat(pom3,'*'); - do łańcucha pom3 dopisz znak mnożenia
if czy_przeniesc_znak(tym1) then inc(limi3); - jeśli to możliwe, usuń...
```
pierwszy znak +/– z łańcucha tym1 oraz zlicz usunięty znak odejmowania w zmiennej limi3

```
if czy_brak_nawiasu(tym1,0) then w_nawias(tym1); - uzupełnij łańcuch...
```
tym1 skrajną parą nawiasów, gdy znajdują się w nim znaki +/– nie zamknięte w nawiasie

```
StrCat(pom3,'('); - do łańcucha pom3 dopisz znak nawiasu otwierającego
StrCat(pom3,tym1); - dopisz do łańcucha pom3 zawartość łańcucha tym1
StrCat(pom3,'/'); - do łańcucha pom3 dopisz znak dzielenia
if tym2[0]<>'(' then w_nawias(tym2); - jeśli łańcuch tym2 nie rozpoczyna się...
```
od nawiasu otwierającego, zamknij zawartą w nim funkcję w nawiasie

```
StrCat(pom3,tym2); - dopisz do łańcucha pom3 zawartość łańcucha tym2
StrCat(pom3,')'); - dopisz do łańcucha pom3 znak nawiasu zamykającego
if (limi3 mod 2)>0 then pom3[0]:='-'; - czy reszta z dzielenia przez dwa...
```
licznika znaków odejmowania jest większa od zera? Jeśli tak, zamień w łańcuchu pom3...
początkowy znak dodawania na odejmowanie

```
vb1:=StrLen(pom3); - z ilu znaków składa się łańcuch pom3?
GetMem(lancuch^.t3,vb1*sizeof(PChar)); - przydziel pamięć dla łańcucha...
```
z pola t3 o rozmiarze odpowiednim do zliczonej w zmiennej vb1 ilości znaków typu PChar

```
StrCopy(lancuch^.t3,pom3); - skopiuj zawartość łańcucha pom3 do łańcucha z pola t3
FreeMem(pom4,zp1*sizeof(PChar)); - zwolnij pamięć zajętą przez łańcuch pom4
FreeMem(pom3,rxp*sizeof(PChar)) - zwolnij pamięć zajętą przez łańcuch pom3
```

Wzór *„G”* stosowany jest wtedy, gdy wartość pochodnej przed znakiem potęgowania jest większa od jedności, natomiast wartość pochodnej za tym znakiem jest większa lub równa jeden. W tym przypadku pochodna wynikowa musi być złożona z dwóch części połączonych znakiem dodawania lub odejmowania (znak łączący obie części pochodnej będzie ustalony w czasie układania pochodnej całkowitej).

Po uzupełnieniu zmiennej `slownik` kodem znakowym identyfikującym ten wzór, wyliczona zostaje orientacyjna ilość znaków składających się na pochodną całkowitą. Liczba ta zapamiętana zostaje w zmiennej `rxp` – na jej podstawie, tymczasowemu łańcuchowi wynikowemu `pom3` przydzielona zostaje pamięć z niewielkim zapasem.

Znak dodawania jest pierwszym elementem wpisanym do łańcucha `pom3`. Po usunięciu z łańcucha `tym2` pierwszego znaku dodawania lub odejmowania, zliczeniu znaku odejmowania w zmiennej `limi3` (o ile istnieje) oraz wstawieniu nawiasu otwierającego przed pierwszym znakiem i zamykającego za ostatnim znakiem w łańcuchu, jego zawartość zostaje dodana do łańcucha `pom3`. Oczywiście, usunięcie początkowego znaku dodawania czy odejmowania nie może wpłynąć na wynik pozostałych elementów funkcji zawartej w tym łańcuchu, dlatego nad możliwością usunięcia znaku bądź jego pozostawienia w łańcuchu czuwa funkcja `czy_przeniesc_znak`. Kolejnym znakiem dodanym do łańcucha `pom3` jest potęgowa-

nie. Ponieważ łańcuch tym3 będzie wstawiany do łańcucha wynikowego wielokrotnie, utworzony został bliźniaczy łańcuch pom4, co pozwoliło zachować łańcuch tym3 w pierwotnej postaci, do ponownego wykorzystania. Po uzupełnieniu w łańcuchu bliźniaczym pom4 skrajnej pary nawiasów w sytuacji, gdy w łańcuchu znajduje się chociaż jeden znak dodawania, odejmowania, mnożenia, dzielenia czy pierwiastkowania nie zamknięty w nawiasie, jego zawartość zostaje dodana do łańcucha pom3. Skoro łańcuch pom4 jest już utworzony i możliwy do wykorzystania, zostaje do niego skopiowana zawartość łańcucha lancuch2 (pochodna za znakiem potęgowania). Z bliźniaczego łańcucha zostaje usunięty początkowy znak dodawania lub odejmowania a znak odejmowania zostaje zliczony w zmiennej limi3. Gdy wartość pochodnej za znakiem potęgowania jest większa od jedności, po uprzednim wstawieniu do łańcucha pom3 znaku mnożenia i uzupełnieniu łańcucha bliźniaczego pom4 skrajną parą nawiasów, jego zawartość zostaje dołączona do łańcucha pom3. W tym przypadku zachowana zostaje wyższość znaku mnożenia nad dodawaniem czy odejmowaniem. Po dołączeniu do łańcucha pom3 kolejnego znaku mnożenia, dopisane zostają znaki 'ln' zawarte w łańcuchu stałym log3. W tym miejscu łańcuch pom4 zostaje ponownie wykorzystany do dołączenia, do łańcucha tym3 argumentu logarytmu naturalnego. Przedtem w łańcuchu pom4 musi znaleźć się zawartość łańcucha tym2 (funkcja przed znakiem potęgowania), która dodatkowo zostaje zamknięta w parze nawiasów.

Pierwsza część pochodnej jest prawie gotowa, pozostało jeszcze ustalenie początkowego znaku '+' lub '– ' na podstawie parzystości licznika znaków odejmowania, czyli wartości w zmiennej limi3 – jeśli jest ona nieparzysta, pierwszy znak dodawania w łańcuchu pom3 zostaje zamieniony na odejmowanie.

```
slownik:='3'; - wpisz do zmiennej slownik kod znakowy identyfikujący zastosowany wzór
rxp:=StrLen(tym1)+4*StrLen(tym2)+3*StrLen(tym3)+StrLen(lancuch2)
+zapas;
wylicz w zmiennej rxp orientacyjną ilość znaków, z których składać się będzie pochodna

GetMem(pom3,rxp*sizeof(PChar)); - przydziel pamięć dla łańcucha pom3 o wielkości...
                                  proporcjonalnej do wyliczonej ilości znaków

pom3[0]:='+'; - wpisz do łańcucha pom3, w miejsce jego początku, znak '+'
pom3[1]:=#0; - zamknij łańcuch
limi3:=0; - wyzeruj zmienną – licznik znaków odejmowania
if czy_przeniesc_znak(tym2) then inc(limi3); - jeśli można usunąć
istniejący pierwszy znak '+' lub '– ', usuń go z łańcucha tym2 i zlicz usunięty znak odejmowania

if czy_brak_nawiasu(tym2,1) then w_nawias(tym2); - jeśli w łańcuchu tym2...
          znajduje się przynajmniej jeden ze znaków '*' '+' '– ' '/' oraz pierwiastek, zamknij...
                                  zawartą w nim funkcję w nawiasie

StrCat(pom3,tym2); - dołącz zawartość łańcucha tym2 do pom3
StrCat(pom3,'^'); - do łańcucha pom3 dopisz znak potęgowania
GetMem(pom4,rxp*sizeof(PChar)); - przydziel pamięć dla łańcucha pom4 o rozmiarze...
                                  takim samym jaki ma łańcuch pom3

StrCopy(pom4,tym3); - skopiuj zawartość łańcucha tym3 do pom4
if czy_brak_nawiasu(pom4,2) then w_nawias(pom4); - jeśli w łańcuchu pom4 ...
     znajdują się nie zamknięty w nawiasie przynajmniej jeden z możliwych znaków działania ...
          (oprócz potęgowania i pierwiastkowania), zamknij zawartą w nim funkcję w nawiasie

StrCat(pom3,pom4); - dopisz zawartość łańcucha pom4 do pom3
StrCopy(pom4,lancuch2); - skopiuj zawartość łańcucha lancuch2 do pom4
if czy_przeniesc_znak(pom4) then inc(limi3); - jeśli można usunąć…
istniejący pierwszy znak '+' lub '– ', usuń go z łańcucha pom4 i zlicz usunięty znak odejmowania

if rx2>1 then - czy wartość pochodnej za znakiem potęgowania jest większa od jedności?
begin - jeśli tak, to...
```

```
        StrCat(pom3,'*'); - dopisz do łańcucha pom3 znak mnożenia
        if czy_brak_nawiasu(pom4,0) then w_nawias(pom4); - jeśli w…
         łańcuchu pom4 znajduje się znak '+' lub '– ' nie zamknięty w nawiasie,…
         usuń go z łańcucha i zlicz znak odejmowania
        StrCat(pom3,pom4) - dopisz zawartość łańcucha pom4 do pom3
end;
StrCat(pom3,'*'); - do łańcucha pom3 dopisz znak mnożenia
StrCat(pom3,log3); - dopisz do łańcucha pom3 zawartość łańcucha stałego log3 – 'ln'
StrCopy(pom4,tym2); - skopiuj zawartość łańcucha tym2 do pom4
if pom4[0]<>'(' then w_nawias(pom4); - czy łańcuch pom4 rozpoczyna się...
        od znaku nawiasu otwierającego? Jeśli nie, zamknij zawartą w nim funkcję w nawiasie
StrCat(pom3,pom4); - dopisz zawartość łańcucha pom4 (funkcja logarytmowana) do pom3
if (limi3 mod 2)>0 then pom3[0]:='-'; - czy reszta z dzielenia przez dwa...
licznika znaków odejmowania jest większa od zera? Jeśli tak, zamień w łańcuchu pom3...
początkowy znak dodawania na odejmowanie
```

Część druga pochodnej rozpoczyna się od wyzerowania licznika znaków odejmowania, czyli zmiennej `limi3` oraz zapamiętania w zmiennej `vb1` pozycji następnego znaku w łańcuchu `pom3` – pozycja ta będzie wskazywała na miejsce wstawienia znaku dodawania, od którego będzie rozpoczynała się drugą część pochodnej. Znajomość tej pozycji będzie wykorzystana przy ustalaniu znaku drugiej części pochodnej. Tak jak w pierwszej części pochodnej, do łańcucha `pom3`, za wstawionym znakiem dodawania dopisana zostaje zawartość łańcucha `tym2`, a za nim znak potęgowania. Również w tym przypadku, do łańcucha `pom4` skopiowana zostaje zawartość łańcucha `tym3` (funkcja za znakiem potęgowania). Po utworzeniu kopii, zawartość łańcucha `tym2` zostaje dołączona do tymczasowego łańcucha wynikowego `pom3`. W kolejnej instrukcji warunkowej, łańcuch `pom4` zostaje sprawdzony pod kątem istnienia w jego początku znaku dodawania lub odejmowania, jeśli znak taki istnieje i można go usunąć z łańcucha, zostaje on usunięty a znak odejmowania – zliczony w zmiennej `limi3`. Po usunięciu początkowego znaku dodawania lub odejmowania z łańcucha `pom4`, taki sam znak sprawdzany jest na początku łańcucha `tym1` (pochodna funkcji przed znakiem potęgowania) – tak jak poprzednio, znak ten zostaje usunięty z łańcucha a znak odejmowania zliczony w zmiennej `limi3`. Kolejnym elementem dołączonym do łańcucha `pom3` jest funkcja za znakiem potęgowania, której kopia wciąż znajduje się w łańcuchu `pom4`. Z uwagi na to, że za tą funkcją ma znaleźć się ułamek, korzystnym okazało się zamknięcie tego ułamka w nawiasie, lecz skoro licznik ułamka nie jest jeszcze określony, dołączenie funkcji za znakiem potęgowania, a właściwie poprawne jej zamknięcie w nawiasie, uzależnione jest od wartości pochodnej sprzed znaku potęgowania, czyli:

- jeśli pochodna ta jest równa jeden, zostaje ona pominięta a licznikiem ułamka staje się funkcja za znakiem potęgowania, która musi być zamknięta w nawiasie tak, by w całości znalazła się w liczniku;
- jeśli pochodna jest większa od jedności, wówczas to ta pochodna znajdzie się w liczniku ułamka, natomiast funkcja za znakiem potęgowania będzie usytuowana przed ułamkiem.

Mianownikiem ułamka jest zawartość łańcucha `tym2` zawierającego znaki funkcji przed znakiem potęgowania – przed dołączeniem jego zawartości do łańcucha `pom3`, musi on rozpoczynać się znakiem nawiasu otwierającego i kończyć znakiem nawiasu zamykającego, brak nawiasów jest uzupełniany przez procedurę `w_nawias`.

Pochodna jest prawie gotowa, pozostało jeszcze ustalenie znaku rozpoczynającego drugą jej część. Polega to, tak jak w przypadku pierwszej części, na sprawdzeniu parzystości licznika znaków odejmowania. Jeśli wartość ta jest nieparzysta, znak dodawania – którego

pozycję w łańcuchu pom3 zapamiętała zmienna vb1 – zostaje zamieniony na odejmowanie. Nim pochodna znajdzie się w łańcuchu wyjściowym dostępnym przez łańcuch w polu t3, pozostało jeszcze zliczenie znaków znajdujących się w łańcuchu pom3 (oczywiście znaków niepustych), przydzielenie pamięci dla łańcucha wynikowego z pola t3 o rozmiarze odpowiednim dla typu PChar i wyliczonej ilości znaków, skopiowanie zawartości łańcucha pom3 do łańcucha wyjściowego oraz zwolnienie pamięci zajmowanej przez łańcuchy pom3 i pom4.

```
limi3:=0; - wyzeruj licznik znaków odejmowania
vb1:=StrLen(pom3); - zapamiętaj pozycję następnego znaku....
                     rozpoczynającego drugą część pochodnej

StrCat(pom3,'+'); - do łańcucha pom3 dopisz znak dodawania
StrCat(pom3,tym2); - dopisz zawartość łańcucha tym2 do pom3
StrCat(pom3,'^'); - do łańcucha pom3 dopisz znak potęgowania
StrCopy(pom4,tym3); - dopisz zawartość łańcucha tym3 do pom4
if czy_brak_nawiasu(tym3,2) then w_nawias(tym3); - jeśli łańcuch tym3...
          zawiera przynajmniej jeden ze znaków '*' '+' '– ' '/' nie zamknięty w nawiasie,...
          zamknij zawartą w nim funkcję w nawiasie

StrCat(pom3,tym3); - dopisz zawartość łańcucha tym3 do pom3
StrCat(pom3,'*'); - do łańcucha pom3 dopisz znak mnożenia
if czy_przeniesc_znak(pom4) then inc(limi3); - jeśli łańcuch pom4...
rozpoczyna się od znaku '+' lub '– ' i można go usunąć, usuń go i zlicz znak odejmowania

if czy_przeniesc_znak(tym1) then inc(limi3); - jeśli łańcuch tym1...
rozpoczyna się od znaku '+' lub '– ' i można go usunąć, usuń go i zlicz znak odejmowania

prz1:=false; - zainicjuj zmienną logiczną prz1 wartością false
if rx1>1 then - czy wartość pochodnej przed znakiem potęgowania jest większa…
od jedności?
begin - jeśli tak, to...
     if czy_brak_nawiasu(pom4,0) then w_nawias(pom4); - jeśli…
w łańcuchu pom4 znajdują się znaki '+' lub '– ' nie zamknięte w nawiasie, zamknij ...
                                                  zawartą w nim funkcję w nawiasie

     StrCat(pom3,pom4); - dopisz zawartość łańcucha pom4 do pom3 – licznik ułamka
     StrCat(pom3,'*'); - do łańcucha pom3 dopisz znak mnożenia
     StrCat(pom3,'('); - do łańcucha pom3 dopisz znak nawiasu otwierającego
     prz1:=true; - zapamiętaj potrzebę wstawienia nawiasu zamykającego na końcu…
                   ułamka
     if czy_brak_nawiasu(tym1,2) then w_nawias(tym1); - czy
                w łańcuchu...
                tym1 znajduje znak '*', '/', '+', '– ' nie zamknięty w nawiasie? Jeśli tak,...
                zamknij zawartą w nim funkcję w nawiasie

     StrCat(pom3,tym1) - dopisz zawartość łańcucha tym1 do pom3 – licznik ułamka
end else
    begin
         if czy_brak_nawiasu(pom4,2) then w_nawias(pom4);
           zawartość...
           łańcucha pom4 (funkcja za znakiem potęgowania) jest licznikiem ułamka, dlatego...
           zamknij ją w nawiasie, gdy w łańcuchu znajduje się choć jeden znak '+' '– ' '*' '/'...
           nie ujęty w nawiasie

         StrCat(pom3,pom4); - dopisz zawartość łańcucha pom4 do pom3
    end;
StrCat(pom3,'/'); - do łańcucha pom3 dopisz znak dzielenia
if tym2[0]<>'(' then w_nawias(tym2); - jeśli łańcuch tym2 nie rozpoczyna się...
                   od nawiasu otwierającego, zamknij zawartą w nim funkcję w nawiasie

StrCat(pom3,tym2); - dopisz zawartość łańcucha tym2 do tym3 – mianownik ułamka
```

```
if prz1 then StrCat(pom3,')'); - jeśli był wstawiony znak nawiasu otwierającego,...
                                wstaw znak nawiasu zamykającego

if (limi3 mod 2)>0 then pom3[vb1]:='- '; czy reszta z dzielenia przez dwa...
  licznika znaków odejmowania jest większa od zera? Jeśli tak, zamień w łańcuchu pom3...
  znak dodawania na odejmowanie w miejscu wskazanym zmienną vb1

vb1:=StrLen(pom3); - z ilu znaków składa się łańcuch pom3?
GetMem(lancuch^.t3,vb1*sizeof(PChar)); - przydziel pamięć dla łańcucha...
          z pola t3 o rozmiarze odpowiednim do typu PChar i wyliczonej ilości znaków

StrCopy(lancuch^.t3,pom3); - skopiuj zawartość łańcucha pom3 do łańcucha z pola t3
FreeMem(pom3,rxp*sizeof(PChar)); - uwolnij pamięć zajmowaną przez łańcuch pom3
FreeMem(pom4,rxp*sizeof(PChar)) - uwolnij pamięć zajmowaną przez łańcuch pom4
```

Nieco inaczej znajdowana jest pochodna, gdy znakiem rozdzielającym dwie funkcje jest mnożenie lub dzielenie. W tym przypadku program przewiduje zastosowanie dwóch zestawów wzorów, zależnie od sumy wartości znaków pochodnej funkcji sprzed znaku mnożenia czy dzielenia – poniżej opisany zostanie przypadek, w którym wartość ta jest większa od zera. Tradycyjnie, na początku realizacji wybranego wzoru zliczana jest orientacyjna ilość znaków, z których będzie składała się pochodna, uwzględniając przy tym niewielki naddatek znaków zapamiętany w stałej o nazwie `zapas`. Zaraz potem, dla dwóch łańcuchów lokalnych, `pom3` i `pom4` przydzielona zostaje pamięć o wielkościach proporcjonalnych do wyliczonej ilości znaków. Pierwszym wprowadzonym znakiem do łańcucha `pom3` jest dodawanie. Kolejne instrukcje podzielone są na dwie części – pierwsza z nich jest wspólna dla obydwu znaków rozdzielających funkcje.

Z łańcucha `tym1`, zawierającego znaki pochodnej przed znakiem rozdziału, zostaje usunięty możliwy, pierwszy znak dodawania lub odejmowania – znak odejmowania, po jego usunięciu zostaje zliczony w liczniku znaków odejmowania, czyli w zmiennej `limi3`. Jeśli suma wartości znaków w tym łańcuchu jest większa od jedności, jego zawartość zostaje dodana do łańcucha `pom3` a za nim znak mnożenia, przedtem łańcuch `tym1` uzupełniony zostaje skrajną parą nawiasów w sytuacji, gdy znajdują się w nim znaki dodawania lub odejmowania nie zamknięte w nawiasie. Tak jak łańcuch `tym1`, również `tym3` zostaje przygotowany przed dołączeniem jego zawartości do lokalnego łańcucha wynikowego `pom3` poprzez usunięcie ewentualnego, początkowego znaku dodawania lub odejmowania, zliczenie usuniętego znaku odejmowania oraz dodanie nawiasu otwierającego na początku łańcucha i zamykającego na jego końcu, gdyby w łańcuchu znajdował się chociaż jeden znak dodawania lub odejmowania nie zamknięty w nawiasie. Po dołączeniu zawartości łańcucha `tym3` do `pom3`, na podstawie parzystości wartości licznika znaków odejmowania, ustalany zostaje początkowy znak pochodnej w łańcuchu `pom3` – wartość nieparzysta w zmiennej `limi3` wpłynie na zamianę początkowego znaku z dodawania na odejmowanie. Gdy wartość pochodnej za znakiem rozdziału jest większa od zera, w łańcuchu `pom3` rozpocznie się budowa drugiej części pochodnej, wspólnej dla mnożenia i dzielenia. W zmiennej `vb1` zapamiętana zostaje pozycja następnego znaku w łańcuchu `pom3` – będzie nim znak dodawania, który może zostać zamieniony na odejmowanie w momencie, gdy do łańcucha `pom3` zostaną dołączone wszystkie elementy mające wpływ na ten znak. Po dodaniu do łańcucha znaku dodawania i wyzerowaniu zmiennej `limi3`, z łańcucha `tym2` składającego się z funkcji przed znakiem rozdziału, usunięty zostaje początkowy znak dodawania lub odejmowania – usunięcie znaku odejmowania zostaje zliczone w zmiennej `limi3` a sam łańcuch zostaje uzupełniony o brakującą parę nawiasów. Po przygotowaniu łańcucha `tym2`, jego zawartość zostaje dołączona do łańcucha `pom3`. Ponieważ łańcuch `lancuch2` został przekazany procedurze bez możliwości modyfikacji, jego zawartość musiała zostać skopiowana do drugiego łańcucha lokalnego `pom4`,

dzięki czemu pochodną za znakiem rozdziału można już modyfikować. Z łańcucha pom4 usunięty zostaje pierwszy znak dodawania lub odejmowania a usunięty znak odejmowania zostaje zliczony w zmiennej limi3. Gdy pochodna za znakiem podziału nie składa się wyłącznie z cyfry jeden, zawartość łańcucha pom4, po uzupełnieniu w nim brakujących nawiasów, zostaje dołączona do łańcucha pom3. Główna część pochodnej jest już gotowa, można więc ustalić znak rozpoczynający drugą jej część, którego miejsce w łańcuchu pom3 wskazuje zmienna vb1. Ustalenie znaku polega na zbadaniu parzystości wartości zawartej w zmiennej limi3 – w tym miejscu, ustalenie znaku uzależnione jest od samego znaku rozdzielającego obie funkcje, jeśli jest nim mnożenie, zamiana z plusa na minus następuje przy nieparzystej wartości w limi3, natomiast dla znaku dzielenia – przeciwnie.

```
rxp:=StrLen(tym1)+StrLen(tym2)+StrLen(tym3)+StrLen(lancuch2)+zapas;
```
określ, z ilu znaków może składać się pochodna, uwzględniając niewielki zapas

```
GetMem(pom3,rxp*sizeof(PChar)); - przydziel pamięć dla łańcucha pom3,...
```
odpowiednią dla typu PChar i wyliczonej ilości znaków dla pochodnej

```
GetMem(pom4,rxp*sizeof(PChar)); - (jak wyżej, tylko dla łańcucha pom4)
pom3[0]:='+'; - wpisz pierwszy znak dodawania do łańcucha pom3
pom3[1]:=#0; - zamknij łańcuch
limi3:=0; - wyzeruj zmienną limi3 – licznik znaków odejmowania
if czy_przeniesc_znak(tym1) then inc(limi3); - jeśli łańcuch tym1 ...
```
rozpoczyna się od znaku '+' lub '– ' i można go usunąć, usuń go i zlicz znak odejmowania

```
if rx1>1 then - czy wartość znaków pochodnej jest większa od jedności?
begin - jeśli tak, to...
     if czy_brak_nawiasu(tym1,0) then w_nawias(tym1); - jeśli...
```
w łańcuchu tym1 (pochodna przed znakiem rozdziału) znajduje się znak '+' lub '– ' nie zamknięty w nawiasie, zamknij zawartą w nim funkcję w nawiasie

```
     StrCat(pom3,tym1); - dopisz zawartość łańcucha tym1 do pom3
     StrCat(pom3,'*') - do łańcucha pom3 dopisz znak mnożenia
end;
if czy_przeniesc_znak(tym3) then inc(limi3); - jeśli łańcuch tym3 ...
```
rozpoczyna się od znaku '+' lub '– ' i można go usunąć, usuń go i zlicz znak odejmowania

```
if czy_brak_nawiasu(tym3,0) then w_nawias(tym3); - jeśli w łańcuchu...
```
tym3 (funkcja za znakiem rozdziału) znajduje się znak '+' lub '– ' nie zamknięty w nawiasie, zamknij zawartą w nim funkcję w nawiasie

```
 StrCat(pom3,tym3); - dopisz zawartość łańcucha tym3 do pom3
if (limi3 mod 2)>0 then pom3[0]:='-'; - czy reszta z dzielenia przez dwa...
```
licznika znaków odejmowania jest większa od zera? Jeśli tak, zamień w łańcuchu pom3... początkowy znak dodawania na odejmowanie

```
if rx2>0 then - czy wartość pochodnej za znakiem rozdziału jest większa od zera?
begin - jeśli tak, to...
     vb1:=StrLen(pom3); - zapamiętaj pozycję następnego znaku w łańcuchu pom3
     StrCat(pom3,'+'); - dopisz znak '+' do łańcucha pom3
     limi3:=0; - wyzeruj licznik znaków odejmowania – zmienną limi3
     if czy_przeniesc_znak(tym2) then inc(limi3); - jeśli łańcuch tym2 ...
```
rozpoczyna się od znaku '+' lub '– ' i można go usunąć, usuń go i zlicz znak odejmowania

```
     if czy_brak_nawiasu(tym2,0) then w_nawias(tym2); - jeśli...
```
w łańcuchu tym2 (funkcja przed znakiem rozdziału) znajduje się znak '+' lub '– ' nie zamknięty w nawiasie, zamknij zawartą w nim funkcję w nawiasie

```
     StrCat(pom3,tym2); - dopisz zawartość łańcucha tym2 do pom3
     StrCopy(pom4,lancuch2); - skopiuj zawartość łańcucha lancuch2 do pom4
     if czy_przeniesc_znak(pom4) then inc(limi3); - jeśli łańcuch tym2 ...
```
rozpoczyna się od znaku '+' lub '– ' i można go usunąć, usuń go i zlicz znak odejmowania

```
        if rx2>1 then - czy wartość pochodnej za znakiem rozdziału jest większa...
                        od jedności?
        begin - jeśli tak, to...
              StrCat(pom3,'*'); - dopisz do łańcucha pom3 znak mnożenia
              if czy_brak_nawiasu(pom4,0) then w_nawias(pom4); - jeśli...
               w łańcuchu pom4 (kopia pochodnej za znakiem rozdziału) znajduje się znak...
               '+' lub '– ' nie zamknięty w nawiasie, zamknij zawartą w nim funkcję w nawiasie
              StrCat(pom3,pom4) - dopisz zawartość łańcucha pom4 do pom3
        end;
        if zp1=42 then - czy znakiem rozdziału funkcji jest mnożenie?
          if (limi3 mod 2)>0 then pom3[vb1]:='-' - czy reszta z dzielenia...
                przez dwa licznika znaków odejmowania jest większa od zera? Jeśli tak,...
                zamień w łańcuchu pom3znak dodawania na odejmowanie...
                w miejscu wskazanym zmienną vb1
           else - w przeciwnym przypadku, gdy reszta z dzielenia jest równa zero
        else if (limi3 mod 2)=0 then pom3[vb1]:='-' - w przeciwnym...
                                    przypadku, gdy znakiem rozdziału jest dzielenie...
                                    oraz gdy reszta z dzielenia jest równa zero,...
                                    zamień istniejący znak '+' na '– ' w łańcucha pom3...
                                    w miejscu wskazanym zmienną vb1
end;
```

Jeśli znakiem rozdziału jest mnożenie, pochodna w łańcuchu pom3 jest już kompletna, pozostaje tylko uzupełnienie zmiennej slownik stosownym znakiem identyfikującym wybrany wzór do budowania pochodnej, zliczenie niepustych znaków znajdujących się w łańcuchu pom3, przydzielenie pamięci dla łańcucha wyjściowego, dostępnego przez łańcuch w polu t3, skopiowanie zawartości łańcucha pom3 do wyjściowego oraz zwolnienie pamięci zajmowanej przez łańcuchy lokalne pom3 i pom4.

```
if zp1=42 then - czy znakiem rozdziału funkcji jest mnożenie?
begin - jeśli tak, to...
      slownik:='5'; - do zmiennej slownik wpisz kod znakowy zastosowanego wzoru
      FreeMem(pom4,rxp*sizeof(PChar)); - zwolnij pamięć zajmowaną przez ...
                                         łańcuch pom4
      vb1:=StrLen(pom3); - z ilu znaków niepustych składa się łańcuch pom3?
      GetMem(lancuch^.t3,vb1*sizeof(PChar)); - przydziel pamięć dla łańcucha...
              z pola t3 o rozmiarze odpowiednim do typu PChar i ilości niepustych znaków
      StrCopy(lancuch^.t3,pom3); - skopiuj zawartość łańcucha pom3 do..
                                   łańcucha wynikowego w polu t3
      FreeMem(pom3,rxp*sizeof(PChar)) - zwolnij pamięć zajmowaną przez...
                                        łańcuch pom3
end else
```

Gdy znakiem rozdzielającym dwie funkcje jest dzielenie, zawartość łańcucha pom3 będzie jedynie licznikiem ułamka pochodnej wynikowej, dlatego by umożliwić zamknięcie licznika w nawiasie, zawartość tego łańcucha została skopiowana do drugiego łańcucha lokalnego pom4. Ustalony wcześniej, początkowy znak pochodnej musi także znaleźć się w liczniku. W tej sytuacji znak rozpoczynający pełną pochodną dla tego przypadku powinien być zgodny z pierwszym znakiem funkcji sprzed znaku dzielenia. Ponieważ pierwszy znak dodawania lub odejmowania został już usunięty z łańcucha tym2, można go jeszcze odtworzyć z łańcucha z pola t2, który wciąż posiada kompletne funkcje. Jeśli zatem pierwszym znakiem w łańcuchu z pola t2 jest odejmowanie, do łańcucha pom3 trafia pierwszy znak odejmowania,

w każdym innym przypadku jest to znak dodawania. By licznik ułamka nie rozpoczynał się od niepotrzebnego znaku dodawania, znak ten zostaje z łańcucha usunięty. Teraz można już sprawdzić, czy kompletny licznik ułamka, który znajduje się w łańcuchu pom4, trzeba zamknąć w nawiasie – warunkiem jest obecność chociaż jednego znaku dodawania lub odejmowania nie zamkniętego w nawiasie, jeśli znak taki istnieje, cała zawarta w nim funkcja zostaje zamknięta w nawiasie. Tak przygotowany licznik zostaje dopisany do łańcucha pom3. Kolejnym dopisanym znakiem jest dzielenie a za nim znak nawiasu otwierającego. Mianownikiem ułamka jest funkcja za znakiem dzielenia, która znajduje się w łańcuchu tym3. Ponieważ funkcja ta będzie podniesiona do potęgi drugiej, co jest zgodne ze wzorem *'I'* oraz *'J'* dla dwóch funkcji połączonych znakiem dzielenia, łańcuch tym3 musi być sprawdzony, pod kątem tego, czy znajdują się w nim znaki działania nie zamknięte w nawiasie, czyli dodawania, odejmowania, mnożenia, dzielenia oraz pierwiastkowania – wykrycie choć jednego takiego znaku powoduje wywołanie procedury w_nawias, która zamknie zawartą w łańcuchu funkcję w dodatkowej parze nawiasów. W ten sposób potęgowanie będzie obejmowało cały mianownik. Teraz już można przenieść zawartość łańcucha tym3 do pom3. Pozostało jeszcze dołączenie do łańcucha znaków: potęgowania, cyfry dwa i nawiasu zamykającego. Łańcuch pom3 jest już kompletny, można już jego zawartość przenieść do łańcucha wynikowego, dostępnego przez pole t3. Polega to na zliczeniu wszystkich, niepustych znaków zawartych w łańcuchu pom3, przydzieleniu pamięci dla łańcucha wynikowego o rozmiarze odpowiednim dla typu PChar i wyliczonej ilości znaków oraz skopiowaniu zawartości łańcucha pom3 do łańcucha z pola t3. Zwolnienie pamięci zajmowanej przez łańcuchy lokalne pom3 i pom4 kończy ten przypadek budowania pochodnej.

```
      if zp1=42 then
      begin
           . . .
      end else – jeśli znakiem rozdzielającym funkcje nie jest mnożenie, jest nim dzielenie
          begin
               slownik:='4'; – uzupełnij zmienną slownik kodem znakowym...
                                zastosowanego wzoru

               StrCopy(pom4,pom3); – zachowaj zawartość łańcucha pom3 w pom4
               if   lancuch^.t2[0]='-   '   then   pom3[0]:='-   '   else
pom3[0]:='+';
                jeśli funkcja przed znakiem dzielenia rozpoczyna się znakiem odejmowania,...
                wpisz znak odejmowania na początek łańcucha pom3, w przeciwnym przypadku...
                wpisz znak dodawania

               pom3[1]:=#0; – zamknij łańcuch
               if pom4[0]='+' then – czy łańcuch pom4 rozpoczyna się od znaku '+'?
               begin – jeśli tak, to...
                    rxp:=StrLen(pom4); – z ilu znaków składa się łańcuch pom4?
                    for rx1:=0 to rxp do pom4[rx1]:=pom4[rx1+1]
                                         usuń znak...
                                         zastępując każdy znak w pętli znakiem następnym
               end;
              if czy_brak_nawiasu(pom4,0) then w_nawias(pom4); – jeśli...
             w łańcuchu pom4 znajduje się choć jeden znak '+' lub '– ' nie zamknięty w nawiasie,...
              zamknij zawartą w nim funkcję w nawiasie

               StrCat(pom3,pom4); – skopiuj zawartość łańcucha pom4 do pom3
               StrCat(pom3,'/'); – dopisz do łańcucha pom3 znak dzielenia
               StrCat(pom3,'('); – dopisz do łańcucha pom3 znak nawiasu otwierającego
               if czy_brak_nawiasu(tym3,1) then w_nawias(tym3); – jeśli...
                     w łańcuchu tym3 znajduje się choć jeden ze znaków '*' '+' '– ' '/' '\' ...
                     zamknij zawartą w nim funkcję w nawiasie
```

```
            StrCat(pom3,tym3); - dopisz zawartość łańcucha tym3 do pom3
            StrCat(pom3,'^'); - dołącz do łańcucha pom3 znak potęgowania
            StrCat(pom3,'2'); - dołącz do łańcucha pom3 znak cyfry dwa
            StrCat(pom3,')'); - dołącz do łańcucha pom3 znak nawiasu zamykającego
            vb1:=StrLen(pom3); - z ilu niepustych znaków składa się łańcuch pom3?
            GetMem(lancuch^.t3,vb1*sizeof(PChar)); - przydziel pamięć dla ...
                                    dla łańcucha z pola t3 o rozmiarze odpowiednim…
                                    dla typu PChar i wyliczonej ilości znaków
            StrCopy(lancuch^.t3,pom3); - skopiuj zawartość łańcucha pom3 do…
                                          łańcucha wynikowego
            FreeMem(pom3,rxp*sizeof(PChar)); - zwolnij pamięć zajmowaną przez...
                                                łańcuch pom3
            FreeMem(pom4,rxp*sizeof(PChar)) - jak wyżej, tylko dla łańcucha pom4
      end;
```

Przed opuszczeniem procedury `trzy` nie wolno zapomnieć o zwolnieniu pamięci zajmowanej jeszcze przez trzy łańcuchy lokalne `tym1`, `tym2` i `tym3`. Dzięki zachowaniu ich pierwotnych rozmiarów w zmiennych `tx1`, `tx2` i `tx3`, pamięć zostanie zwolniona w całości, niezależnie od zawartości łańcuchów.

```
FreeMem(tym1,tx1*sizeof(PChar));
FreeMem(tym2,tx2*sizeof(PChar));
FreeMem(tym3,tx2*sizeof(PChar));
```

Ostatnią instrukcją jest sprawdzenie, czy pochodna została znaleziona przez opisywaną procedurę – pusty łańcuch z pola `t3` wskazuje na błąd, dlatego w takiej sytuacji, procedura `trzy` musi być opuszczona z ustawioną zmienną błędu i przypisanym, stosownym kodem błędu informującym użytkownika o nierozpoznaniu funkcji.

```
if lancuch^.t3=nil then - czy łańcuch z pola t3 posiada adres pusty?
begin - jeśli tak, to...
     bl:=true; - ustaw zmienną błędu w pozycji true
     blw:=10 - przypisz stosowny kod liczbowy błędu zmiennej kodu błędu
end;
```

X. 9.1. Czy funkcja potęgująca jest liczbowa? Funkcja `czy_policzalne`

Funkcja ma za zadanie przeanalizować dostarczony łańcuch znakowy i udzielić odpowiedzi w postaci zwracanej wartości logicznej, czy łańcuch ten można przekonwertować do postaci liczbowej. By konwersja była możliwa, łańcuch może składać się wyłącznie ze znaków cyfr wyposażonych w ewentualne separatory dziesiętne w postaci kropki lub przecinka. Dopuszczalne jest także istnienie jednego znaku dzielenia między znakami cyfr oraz znaku dodawania lub odejmowania na początku łańcucha bądź – w przypadku znaku dzielenia – tuż za tym znakiem. Trzeba tu wyraźnie zaznaczyć, że jest to jedynie wstępna ocena łańcucha – to, czy z zawartości łańcucha będzie można uzyskać liczbę wymierną czy całkowitą, zostanie stwierdzone podczas konwersji, już poza opisywaną funkcją.

Ułatwieniem w identyfikacji znaków z łańcucha jest zadeklarowanie dwóch łańcuchów stałych – pierwszy składa się z wartości znaków reprezentujących cyfry a drugi – wartości znaków dodawania i odejmowania. Sczytywanie znaków oraz ich sprawdzanie odbywa się

w pętli `while..do`. Gdy analizowany znak znajduje się w tablicy `cyfry`, zmienna logiczna `sp1` ustawiona zostaje w pozycji `true`. Jeśli kolejnym znakiem okaże się dodawanie lub odejmowanie, pętla zostaje opuszczona z ustawioną wartością zmiennej logicznej `sp1` w pozycji `false`. Oznacza to, że znak dodawania lub odejmowania znajduje się w innym miejscu w łańcuchu niż jest to dozwolone, dlatego funkcja zwróci wartość `false`. Z kolei odczytanie znaku dzielenia przy ustawionej zmiennej `sp1` w pozycji `true`, powoduje ustawienie tej zmiennej w pozycję `false` i kontynuowanie pętli – gdyby zmienna miała wartość `false`, oznaczałoby to, że przed znakiem dzielenia nie ma znaków cyfr, dlatego w takiej sytuacji pętla zostaje opuszczona a funkcja zwraca wartość `false`. Odczytanie znaku innego niż dozwolony powoduje natychmiastowe opuszczenie pętli z ustawioną zmienną `sp1` w pozycji `false` a sama funkcja zwraca wartość `false`. Warto zaznaczyć, że separatory dziesiętne oraz nawiasy nie są w pętli sprawdzane a jedynie dopuszczone.

```
function czy_policzalne(const tablica:PChar):boolean;
const cyfry=[48,49,50,51,52,53,54,55,56,57];
       tablica stała, składająca się z wartości znaków cyfr i separatorów dziesiętnych
      plusminus=[43,45];
       tablica stała, składająca się z wartości znaków dodawania i odejmowania
var ip1,ap1:integer;
    sp1:boolean;
begin
     sp1:=false; - przypisz zmiennej sp1 wartość false
     ip1:=0; - zainicjuj zmienną ip1 wartością zero
     ap1:=ord(tablica[ip1]); - odczytaj wartość pierwszego znaku z łańcucha
     while ap1<>0 do - wykonaj krok pętli, póki wartość znaku jest różna od zera
     begin
          if ap1 in cyfry then sp1:=true; - jeśli wartość odczytanego...
                    znaku należy do cyfry, ustaw zmienną sp1 w pozycji true
          if ap1=47 then - czy jest to wartość odczytanego znaku dzielenia?
             if sp1 then sp1:=false - jeśli tak, to gdy poprzednio,...
            odczytanym znakiem była cyfra, ustaw zmienną sp1 w pozycji false
             else break; - jeśli zmienna sp1 ma wartość false, opuść pętlę
          if sp1 then - czy znak cyfry został już odczytany?
             if ap1 in plusminus then - jeśli tak, to czy wartość obecnie...
                       odczytanego znaku zawiera się w tablicy plusminus?
             begin - jeśli tak, to znak znajduje się w niedozwolonym…
                       miejscu, więc...
                  sp1:=false; - ustaw zmienną sp1 w pozycji false
                  break - opuść pętlę
             end;
          if (ap1=42) or (ap1>57) then - czy odczytana wartość znaku...
                                              należy do mnożenia lub litery?
          begin - jeśli tak, to...
               sp1:=false; - ustaw zmienną sp1 w pozycji false
               break - opuść pętlę
          end;
          inc(ip1); - zwiększ o jeden wartość w zmiennej ip1
          ap1:=ord(tablica[ip1]) - odczytaj wartość następnego znaku ...
                                                    z tablicy tablica
     end;
     Result:=sp1 - zwróć przez funkcję wartość zawartą w zmiennej sp1
end;
```

X. 9.2. Zmniejszenie funkcji potęgującej o jeden – procedura `minus_jeden`

Zadaniem procedury jest przetworzenie zawartości dostarczonego łańcucha znakowego na liczbę, odjęcie od tej liczby cyfry jeden, przetworzenie otrzymanej liczby z powrotem do postaci znakowej i zwrócenie otrzymanego wyniku do pierwszego dostarczonego łańcucha znakowego. Otrzymany łańcuch składa się ze znaków dających się przetworzyć na postać liczbową – takie zapewnienie daje procedurze wcześniejsze zbadanie łańcucha przez funkcję `czy_policzalne`, która zwróciwszy wartość logiczną `true` dopuszcza istnienie w łańcuchu znaków cyfr, separatorów dziesiętnych w postaci kropki lub przecinka oraz jednego znaku dzielenia między cyframi. Obecność znaku dzielenia daje możliwość wykonania działania także na ułamku zwykłym – w tym przypadku jedynkę trzeba przedstawić w postaci ułamka zwykłego, zgodnie z poniższym wzorem:

$$\frac{3}{5}-1=\frac{3}{5}-\frac{5}{5}=-\frac{2}{5}$$

Na początku pracy procedury, dla lokalnego łańcucha znakowego typu `PChar`, przydzielona zostaje pamięć, której rozmiar będzie identyczny z rozmiarem pamięci przydzielonej dostarczonemu łańcuchowi w parametrze. Następnie w pętli `while..do`, znaki cyfr i separatorów dziesiętnych zostają kopiowane do utworzonego łańcucha lokalnego znak po znaku. Jeśli zidentyfikowany zostanie znak dzielenia, dotychczas skopiowane znaki z łańcucha lokalnego zostają przetworzone do łańcucha typu `string` a następnie do liczby zmiennoprzecinkowej. Ponieważ może się zdarzyć, że zastosowany separator dziesiętny będzie niezgodny z ustawieniami regionalnymi w systemie operacyjnym, co podczas konwersji może wywołać błąd, dlatego zamiana łańcucha typu `string` na liczbę została zamknięta w instrukcji `try..except..end` – niewłaściwy separator dziesiętny spowoduje ustawienie zmiennej błędu w pozycji `true` i opuszczenie pętli. Jeśli zamiana na liczbę przebiegła pomyślnie, pierwsza zmienna `a1` posiada już liczbą będącą licznikiem ułamka. By można było w tej samej pętli sczytać znaki cyfr mianownika, przed opuszczeniem instrukcji warunkowej, w której wykryty został znak dzielenia, zmiennej `ax3`, odpowiedzialnej za wpis do łańcucha lokalnego, przypisano wartość zero, by wpis rozpoczął się od początku łańcucha; natomiast zmiennej logicznej `druga` przypisano wartość `true`, by w ten sposób zaznaczyć obecność ułamka zwykłego. Po opuszczeniu pętli, zawartość łańcucha lokalnego zostaje przekonwertowana do łańcucha typu `string`. Od tej chwili, działanie procedury uzależnione jest od wartości w zmiennej logicznej `druga`:

- dla wartości `false`, procedura zajmuje się tylko jedną liczbą. Zatem po przekonwertowaniu zawartości łańcucha znakowego typu `PChar` do łańcucha `ciag1` typu `string`, konwersja do liczby zmiennoprzecinkowej została zamknięta w instrukcji `try..except..end`, by nie dopuścić do powstania błędu konwersji. Gdy konwersja przebiegnie bez błędu, od otrzymanej w wyniku konwersji liczby, zawartej w zmiennej `a1`, odejmowana jest jedynka. Otrzymana liczba jest z powrotem konwertowana, najpierw do łańcucha typu `string` a następnie do łańcucha znakowego `pom1` typu `PChar`. Do łańcucha wyjściowego `tab1` dopisany zostaje znak potęgowania, a za nim zawartość łańcucha lokalnego `pom1` zawierającego wynik konwersji;
- dla wartości `true`, procedura musi zająć się dwiema liczbami. Również w tym przypadku, konwersja z postaci `string` do liczby zmiennoprzecinkowej została zamknię-

ta w instrukcji `try..except..end`. Błąd konwersji będzie skutkował ustawieniem zmiennej błędu w pozycji `true`, zwolnieniem pamięci zajmowanej przez łańcuch lokalny i opuszczeniem procedury. Po bezbłędnej konwersji, od pierwszej liczby, będącej licznikiem ułamka, utworzonej jeszcze w pętli `while..do`, odejmowana jest druga liczba, czyli mianownik. Uzyskany w ten sposób wynik odejmowania, zapamiętany w pierwszej zmiennej `a1` wraz z mianownikiem, którego wartość mieści się w zmiennej `b1`, stanowią wynikowy ułamek zwykły, pomniejszony o liczbę jeden. Jeżeli wartości bezwzględne obu liczb różnią się wzajemnie, zostają one przekonwertowane, najpierw do postaci `string`, a następnie do łańcucha znakowego typu `PChar`. Wynik konwersji wraz z dodatkowymi znakami zostaje przeniesiony do łańcucha wyjściowego `tab1`.

Ostatnią czynnością procedury przed jej opuszczeniem jest zwolnienie pamięci zajmowanej przez łańcuch lokalny `pom1`.

```
procedure minus_jeden(var tab1:PChar;const tab2:PChar);
 var pom1:PChar;
     ciag1:string;
     a1,b1:double;
     az1,ax1,ax2,ax3:integer;
     druga:boolean;
 begin
      ax1:=StrLen(tab2); - z ilu znaków składa się łańcuch wejściowy tab2?
      GetMem(pom1,ax1*sizeof(PChar)); - przydziel pamięć dla łańcucha...
           lokalnego pom1 typu PChar i o wielkości wskazanej zmienną ax1
      ax2:=0; - wyzeruj zmienną wskazującą miejsce w odczytywanym łańcuchu
      ax3:=0; - wyzeruj zmienną wskazującą na miejsce wpisu do łańcucha pom1
      a1:=0.0; - wyzeruj pierwszą zmienną – wartość liczbowa po konwersji
      druga:=false; - zainicjuj zmienną logiczną informującą o ułamku zwykłym
      az1:=ord(tab2[ax2]); - odczytaj wartość pierwszego znaku z łańcucha tab2
      while az1<>0 do - wykonaj krok pętli, póki wartość znaku nie jest zerowa
      begin
           if ((az1>=44) and (az1<=46)) or ((az1>=48)
              and (az1<=57)) then - czy odczytany znak jest cyfrą ...
                                    lub separatorem dziesiętnym?
           begin - jeśli tak, to...
                pom1[ax3]:=tab2[ax2]; - przepisz ten znak do łańcucha pom1
                inc(ax3); - zwiększ wartość w zmiennej ax3 o jeden
                pom1[ax3]:=#0 - za wpisanym znakiem cyfry wpisz znak pusty
           end;
           if az1=47 then - czy odczytanym znakiem jest dzielenie?
           begin - jeśli tak, to...
                druga:=true; - zaznacz obecność ułamka zwykłego
                ciag1:=StrPas(pom1); - przekonwertuj zawartość łańcucha...
                        pom1 do postaci string i zapamiętaj wynik konwersji...
                        w zmiennej ciag1
                try - zabezpiecz proces następnej instrukcji przed błędem systemu
                   a1:=StrToFloat(ciag1); - przekonwertuj zawartość...
                      łańcucha typu string na liczbę zmiennoprzecinkową...
                                          i zapamiętaj ją w zmiennej a1
                except - gdy wystąpi błąd...
                       bl:=true; - ustaw zmienną błędu w pozycji true
                       blw:=6; - przypisz zmiennej błędu liczbowy kod błędu
                       break - opuść pętlę
                end;
```

```
            ax3:=0 - przypisz zmiennej ax3 wartość zero, by wpis do łańcucha...
                     lokalnego pom1 rozpoczął się od jego początku
      end;
      inc(ax2); - zwiększ wartość zmiennej ax2 o jeden
      az1:=ord(tab2[ax2]) - odczytaj wartość następnego znaku
                            z łańcucha tab2
end;
ciag1:=StrPas(pom1); - przekonwertuj łańcuch pom1 z PChar...
        do string i zapamiętaj wynik konwersji w zmiennej ciag1

if not bl then - czy zmienna błędu ma wciąż wartość false?
   if druga then - jeśli tak, to czy zmienna druga wskazuje na...
                   ułamek zwykły?
   begin - jeśli tak, to...
         try - zabezpiecz proces następnej instrukcji przed błędem systemu
            b1:=StrToFloat(ciag1); - przekonwertuj zawartość...
                 łańcucha typu string na liczbę zmiennoprzecinkową...
                 i zapamiętaj ją w zmiennej b1

          except - gdy wystąpi błąd...
               bl:=true; - ustaw zmienną błędu w pozycji true
               blw:=6; - przypisz zmiennej błędu liczbowy kod błędu
               FreeMem(pom1,ax1*sizeof(PChar)); - zwolnij ...
                       pamięć zajmowaną przez łańcuch lokalny pom1

               exit - opuść procedurę
         end;

         a1:=a1-b1; - odejmij wartość mianownika od wartości licznika...
                      a wynik odejmowania zapamiętaj w zmiennej a1

         if abs(a1)<>abs(b1) then - czy wartości bezwzględne...
                                    licznika i mianownika ułamka różnią się?

         begin - jeśli tak, to...
               ciag1:=FloatToStr(a1); - przekonwertuj liczbę...
                      (licznik) do postaci string, wynik konwersji...
                      zapamiętaj w zmiennej ciag1

               StrPCopy(pom1,ciag1); - przekonwertuj zawartość...
                                       łańcucha typu string do typu PChar...
                                       – wynik konwersji w łańcuchu pom1

               StrCat(pom1,'/'); - do łańcucha pom1...
                                   dopisz znak dzielenia

               StrCat(pom1,'('); - dopisz znak nawiasu otwierającego
               StrCat(tab1,'^'); - do łańcucha wyjściowego tab1...
                                   dopisz znak potęgowania

               StrCat(tab1,'('); - do łańcucha wyjściowego tab1...
                                   dopisz znak nawiasu otwierającego

               StrCat(tab1,pom1); - do łańcucha wyjściowego tab1...
                                    dopisz zawartość łańcucha pom1

               ciag1:=FloatToStr(b1); - przekonwertuj liczbę...
                    (mianownik) do postaci string, wynik konwersji....
                    zapamiętaj w zmiennej ciag1

               StrPCopy(pom1,ciag1); - przekonwertuj zawartość...
                                       łańcucha typu string do typu PChar...
                                       – wynik konwersji w łańcuchu pom1

               StrCat(tab1,pom1); - do łańcucha wyjściowego tab1...
```

```
                                                  dopisz zawartość łańcucha pom1
                    StrCat(tab1,')');  } do łańcucha wyjściowego tab1...
                    StrCat(tab1,')')   } dopisz dwa znaki nawiasu zamykającego
                end;
            end else - jeśli zmienna druga posiada wartość false, to...
                begin
                    try - zabezpiecz proces konwersji przed błędem systemu
                       a1:=StrToFloat(ciag1); - przekonwertuj zawartość...
                            łańcucha typu string na liczbę zmiennoprzecinkową...
                            i zapamiętaj ją w zmiennej a1
                    except - gdy wystąpi błąd...
                          bl:=true; - ustaw zmienną błędu w pozycji true
                          blw:=6; - przypisz zmiennej błędu liczbowy kod błędu
                          FreeMem(pom1,ax1*sizeof(PChar));
                          uwolnij pamięć zajmowaną przez łańcuch lokalny pom1
                          exit - opuść procedurę
                    end;
                    a1:=a1-1.0; - od wartości w zmiennej a1 odejmij jedynkę
                    if abs(a1)>1.0 then - czy wartość bezwzględna...
                                          w zmiennej a1 jest większa od jeden?
                    begin - jeśli tak, to...
                         ciag1:=FloatToStr(a1); - przekonwertuj...
                              wartość w zmiennej a1 do postaci string...
                              i zapamiętaj wynik konwersji w zmiennej ciag1
                         StrPCopy(pom1,ciag1); - przekonwertuj ...
                        zawartość łańcucha typu string do postaci PChar...
                        – wynik konwersji z łańcuchu pom1
                         StrCat(tab1,'^'); - do łańcucha wyjściowego...
                                             tab1 dopisz znak potęgowania
                         StrCat(tab1,pom1) - do łańcucha wyjściowego...
                                             tab1 dopisz zawartość łańcucha pom1
                    end;
                end;
        FreeMem(pom1,ax1*sizeof(PChar)); - zwolnij pamięć zajmowaną...
                                           przez łańcuch lokalny pom1
end;
```

X. 10. Przygotowanie łańcuchów do szukania pochodnej – procedura `wpisz`

Procedura ma za zadanie cykliczne uzupełnianie nowym znakiem łańcuchów znakowych, zawartych w polach `t1` oraz `t2`, należących do elementów `bloki`, które dostępne są za pośrednictwem parametrów. Tak naprawdę procedura otrzymuje w parametrach dwa adresy, z których jeden posiada tzw. wartość domyślną równą adresowi pustemu, przez co możliwe jest pominięcie tej wartości podczas wywoływania procedury. Zgodnie z wymogami języka programowania, parametr z wartością domyślną został umieszczony jako ostatni w nagłówku procedury. Wszystkie parametry zostały poprzedzone literałem `const`, który co prawda nie pozwala na zmianę ich wartości, ale umożliwia modyfikację pól należących do tych elementów `bloki`, na które wskazują owe parametry. Jednym z parametrów jest także wartość znaku, który ma być dołączony do łańcuchów. Jeśli wartość tego znaku nie reprezentuje znaku pustego oraz gdy pierwszy parametr `d1` wskazuje na element `bloki`, który z kolei posiada niepusty łańcuch w polu `t1`, praca procedury jest kontynuowana. Kolejną czynnością procedury jest wpisanie nowego znaku do lokalnej tablicy dwuznakowej typu `Char`. Za wprowadzonym znakiem dołączony zostaje znak pusty zamykający łańcuch. Gdy dla łańcucha docelowego, którym jest pole `t2`, została już przydzielona pamięć, jego adres zostaje zachowany w lokalnej zmiennej `tymcz`, natomiast dla łańcucha z pola `t2` przydzielony jest nowy obszar pamięci, powiększony o dodatkowy, nowy znak. Po przydzieleniu pamięci, do łańcucha z pola `t2` trafia poprzednia jego zawartość zachowana w zmiennej `tymcz` oraz nowy znak, zapisany w lokalnej tablicy dwuznakowej. Po skopiowaniu wszystkich znaków, poprzedni obszar pamięci, której adres znajduje się w zmiennej `tymcz`, zostaje uwolniony. Jeśli łańcuch z pola `t2` posiada adres pusty, zostaje dla niego przydzielona pamięć o rozmiarze odpowiednim dla jednego znaku a sam znak zostaje do niego skopiowany z dwuznakowej tablicy lokalnej. Gdy ostatni parametr z wartością domyślną posiada adres elementu `bloki`, w pierwszej kolejności sprawdzana jest obecność łańcucha docelowego z pola `t1` należącego do tego elementu. Jeśli łańcuch ten nie posiada adresu pustego, tak jak w poprzednim przypadku, jego adres zostaje zachowany w zmiennej `tymcz` a dla łańcucha z pola `t1` przydzielony zostaje nowy obszar pamięci powiększony o nowy znak. Po skopiowaniu do łańcucha z pola `t1` jego poprzedniej zawartości oraz nowego znaku z tablicy lokalnej, poprzedni obszar pamięci, którego adres wciąż istnieje w zmiennej `tymcz`, zostaje uwolniony. Gdy jednak łańcuch z pola `t1` posiada adres pusty, dla łańcucha przydzielona zostaje pamięć o rozmiarze odpowiednim dla jednego znaku typu `PChar`, a sam znak zostaje do łańcucha skopiowany z lokalnej tablicy dwuznakowej.

```
procedure wpisz(const d1:bloki;const d3:integer;
                const d2:bloki=nil); - trzeci parametr z wartością domyślną
var tymcz:PChar;
    znak:array[0..1] of Char;
    a1:integer;
begin
    if d1=nil then exit; - jeśli pierwszy parametr ma adres pusty, opuść procedurę
    if d1^.t1=nil then exit; - jeśli łańcuch z pola t1 ma adres pusty,…
                               opuść procedurę

    if d3>0 then - czy wartość dostarczonego znaku jest większa od zera?
    begin - jeśli tak, to...
         znak[0]:=Chr(d3); - przekonwertuj z liczby na znak dostarczoną wartość...
                             znaku i wpisz go na początek tablicy znak

         znak[1]:=#0; - zamknij łańcuch dopisując znak pusty
```

```
            if d1^.t2<>nil then - czy łańcuch z pola t2 elementu bloki,...
                                  dostarczonego w parametrze o nazwie d1, nie jest pusty?
            begin - jeśli nie jest pusty, to...
                 tymcz:=d1^.t2; - zachowaj jego adres w zmiennej tymcz
                 a1:=StrLen(d1^.t2); - z ilu znaków składa się łańcuch?
                 GetMem(d1^.t2,(a1+1)*sizeof(PChar)); - przydziel nowy ...
                                  obszar pamięci dla łańcucha z pola t2, uwzględniając…
                                  dotychczasowy rozmiar i nowy znak
                 StrCopy(d1^.t2,tymcz); - skopiuj do łańcucha z pola t2 jego...
                                          dotychczasową zawartość
                 StrCat(d1^.t2,znak); - dopisz zawartość tablicy znak
                 FreeMem(tymcz,a1*sizeof(PChar)) - zwolnij pamięć...
                                  zajmowaną poprzednio przez łańcuch z pola t2
            end else - gdy łańcuch z pola t2 jest pusty, to...
                begin
                     GetMem(d1^.t2,sizeof(PChar)); - przydziel pamięć...
                              dla łańcucha z pola t2 o rozmiarze dla jednego znaku
                     StrCopy(d1^.t2,znak) - skopiuj zawartość tablicy znak…
                                            do łańcucha z pola t2
                end;
            if d2<>nil then - czy adres trzeciego parametru nie jest równy nil?
            begin - jeśli nie jest pusty, to...
                 if d2^.t1<>nil then - czy łańcuch z pola t1 nie jest równy nil?
                 begin - jeśli nie, to...
                      tymcz:=d2^.t1; - zachowaj w zmiennej tymcz adres…
                  łańcucha z pola t1 elementu dostępnego przez trzeci parametr d2…
                      a1:=StrLen(d2^.t1); - z ilu znaków składa się łańcuch…
                                            z pola t1?
                      GetMem(d2^.t1,(a1+1)*sizeof(PChar)); - dla…
                                  łańcucha z pola t1 przydziel nowy obszar…
                                  pamięci uwzględniając...
                                  dotychczasowy rozmiar i nowy znak
                      StrCopy(d2^.t1,tymcz); - skopiuj do łańcucha z pola t1…
                                               jego dotychczasową zawartość
                      StrCat(d2^.t1,znak); - dopisz zawartość tablicy znak
                      FreeMem(tymcz,a1*sizeof(PChar)) - zwolnij pamięć...
                                  zajmowaną poprzednio przez łańcuch z pola t1…
                                  elementu wskazywanego przez trzeci parametr
                 end else - gdy łańcuch z pola t1 trzeciego parametru jest pusty, to...
                     begin
                          GetMem(d2^.t1,sizeof(PChar)); - przydziel…
                                pamięć...
                                dla łańcucha z pola t1 o rozmiarze dla jednego znaku
                          StrCopy(d2^.t1,znak)- skopiuj zawartość tablicy…
                                   Znak do łańcucha z pola t1 trzeciego parametru
                     end;
            end;
        end;
    end;
```

X. 11. Przygotowanie podpowiedzi – procedura `wpis_zewnetrzny`

Zadaniem procedury jest przygotowanie podpowiedzi, z których będzie korzystała procedura `System_Edukacji` znajdująca się w bloku edukacji. Poza tym, utworzone podpowiedzi są grupowane w tzw. poziomy trudności. Numer poziomu trudności jest przygotowywany dla nowej podpowiedzi jeszcze przed wywołaniem procedury i dostępny poprzez parametr `gdzie`. Podstawą do określenia tego numeru jest numer poziomu rozkładu funkcji, który na bieżąco aktualizowany jest w głównej pętli bloku pochodnej, w zmiennej `p1`.

Procedura wywoływana jest w następujących sytuacjach:

– po bezbłędnym znalezieniu pochodnej przez procedurę `jeden`, gdy sczytanym znakiem z łańcucha dostępnego przez pole `t1` było dodawanie lub odejmowanie;

```
if ((k=42) or (k=43) or (k=45) or (k=47) or (k=94))
    and (k1>0) then – czy sczytany znak nie jest pierwszy w łańcuchu i należy...
                      do znaków działania?
begin – jeśli tak, to...
     if not biez1^.fw then – czy pochodna nie jest jeszcze znaleziona?
     begin – jeśli nie jest znaleziona, to...
          jeden(biez1^.t3,biez1^.t2,Z,tym);
          if not bl then – czy pochodna została znaleziona bezbłędnie?
          begin – jeśli tak, to...
               if (k=43) or (k=45) then – czy sczytanym znakiem…
                                          z łańcucha...
                                          jest dodawanie lub odejmowanie?
                  if wpis then – czy parametr wpis ma wartość true?
                      wpis_zewnetrzny(biez1^.t2,Z,p1,tym);
                                  jeśli tak,...
                                  wywołaj procedurę wpis_zewnetrzny,...
                                  dostarczając jej sczytaną funkcję w łańcuchu...
                                  z pola t2, pochodną w łańcuchu Z...
                                  numer poziomu w zmiennej p1 oraz...
                                  znakową informację o rozpoznanej funkcji
```

– po bezbłędnym znalezieniu pochodnej ostatecznej przez procedurę `jeden`, przed wcześniejszym sczytaniu z łańcucha dostępnego przez pole `t1` znaku pustego;

```
if k=0 then – czy sczytany znak jest pusty?
begin – jeśli tak, to...
     if biez1^.fw then przepisz(biez1)
     else – jeśli pochodna nie została znaleziona, to...
     begin
          jeden(biez1^.t4,biez1^.t2,Z,tym);
          if not bl then – czy pochodna została znaleziona bezbłędnie?
          begin – jeśli tak, to...
               if wpis then wpis_zewnetrzny(biez1^.t2,Z,p1,tym);
          gdy parametr wpis ma wartość true, wywołaj procedurę wpis_zewnetrzny
```

– po sczytaniu jednego ze znaków mnożenia, dzielenia lub potęgowania, a więc tuż przed przygotowaniem następnego poziomu do pracy;

```
if (k=94) or (k=42) or (k=47) then – czy sczytanym znakiem jest…
                                     potęgowanie, mnożenie lub dzielenie?
begin – jeśli tak, to...
     . . .
```

```
if wpis then - czy parametr wpis ma wartość true?
begin - jeśli tak, to...
     inc(p1); - zwiększ numer poziomu o jeden
     poziom_wstecz(p1); - wyrównaj numery poziomów w dotychczasowych...
                          podpowiedziach
     if not biez1^.fw then - jeśli pochodna nie została jeszcze...
                             znaleziona, to...
        wpis_zewnetrzny(biez1^.t2,biez1^.t3,p1,tym) - wywołaj..
                                                      procedurę wpis_zewnetrzny
end;
```

– po znalezieniu pochodnej przez procedurę `trzy`;

```
if biez1^.fn then - czy procedura trzy znalazła już pochodną?
begin - jeśli nie, to...
     trzy(biez1,Z,tym);
     if ord(tym)>0 then - czy w zmiennej tym znajduje się informacja...
                          o rozpoznaniu funkcji?
        if wpis then - czy parametr wpis ma wartość true?
            wpis_zewnetrzny(biez1^.t2,biez1^.t3,p1,tym); - jeśli tak,...
                                                           to wywołaj procedurę wpis_zewnetrzny
```

– po znalezieniu pochodnej przez procedurę `dwa`;

```
if k=41 then - czy sczytanym znakiem jest nawias zamykający?
   if k4<>0 then - jeśli tak, to czy licznik nawiasów k4 wskazuje na ich niezgodność?
   begin
        . . .
   end else - jeśli nawiasy są zgodne, to...
       if Z<>nil then - czy łańcuch Z posiada pochodną argumentu funkcji?
       begin - jeśli tak, to...
            . . .
            if ord(tym)>0 then - czy w zmiennej tym znajduje się informacja...
                                 o rozpoznaniu funkcji?
               if wpis then - jeśli tak, to czy parametr wpis ma wartość true?
                   wpis_zewnetrzny(biez1^.t2,biez1^.t3,p1,tym);
                          - jeśli tak, to wywołaj procedurę wpis_zewnetrzny
```

– gdy po opuszczeniu głównej pętli bloku, zmienna logiczna `dot_wpis` posiada wartość `true`. Przypadek ten dotyczy funkcji, której poszczególne wyrazy łączone są znakami dodawania i/lub odejmowania. Jeśli znaki te wykryte zostały w pętli na poziomie podstawowym rozkładu funkcji, nie mogła zostać utworzona podpowiedź odnosząca się do dwóch lub większej ilości wyrazów łączonych znakami dodawania czy odejmowania. By utworzyć brakującą podpowiedź i sprawić, by tylko ona miała przypisany zerowy poziom trudności, należało zaznaczyć przypadek takiej funkcji za pomocą zmiennej logicznej `dot_wpis` i dokonać korekty poziomów trudności pozostałym podpowiedziom, zaraz po opuszczeniu głównej pętli bloku. Zmienna ta ustawiona jest w pozycji `true`, zaraz po sczytaniu znaku dodawania lub odejmowania na poziomie podstawowym, czyli wtedy, gdy zmienna poziomu `p1` posiada wartość zero.

```
if ((k=42) or (k=43) or (k=45) or (k=47) or (k=94))
   and (k1>0) then
begin
. . .
    if not bl then
```

```
if (k=43) or (k=45) then - czy sczytanym znakiem jest...
                                dodawanie lub odejmowanie?
begin - jeśli tak, to...
      . . .
      if k1>0 then - czy nie jest to pierwszy znak?
      begin - jeśli nie jest to pierwszy znak, to...
            if zero then zero:=false else zero:=true;
               zamień wartość zmiennej logicznej zero na przeciwny
            if p1=0 then dot_wpis:=true jeśli aktualnym poziomem ...
           jest poziom podstawowy, wpisz zmiennej  dot_wpis wartość true
      end;
```

Zatem po opuszczeniu głównej pętli bloku pochodnej i zidentyfikowaniu w zmiennej `dot_wpis` wartości `true`, wszystkim dotychczasowym podpowiedziom zostają zwiększone o jeden numery poziomów trudności. Ponadto zostaje utworzona dodatkowa podpowiedź, której dostarczona zostaje zmienna znakowa `tym` z przypisanym znakiem '7', podpowiadającej procedurze `wpis_zewnetrzny` utworzenie podpowiedzi dla funkcji połączonych znakiem lub znakami dodawania i/lub odejmowania.

```
if dot_wpis then - czy zmienna dot_wpis ma wartość true?
begin - jeśli tak, to...
      t_pom:=t_biez; - przypisz zmiennej t_pom adres ostatniej podpowiedzi
      while t_pom<>nil do - wykonaj krok, póki zmienna t_pom nie ma adresu…
                                pustego
      begin
            inc(t_pom^.poziom); - zwiększ o jeden wartość w polu poziom
            t_pom:=t_pom^.lewa - wczytaj do zmiennej t_pom adres...
                                poprzedniej podpowiedzi
      end;
      tym:='7'; - przypisz zmiennej tym znak identyfikujący przypadek funkcji...
                  łączonych znakiem dodawania lub odejmowania
      wpis_zewnetrzny(cialo,Z,0,tym)
                                          wywołaj procedurę wpis_zewnetrzny
end;
```

W każdym opisanym przypadku, wywołanie opisywanej procedury uzależnione jest od wartości parametru o nazwie `wpis` (jeśli program uruchomiony został bez ćwiczeń, wówczas tworzenie podpowiedzi nie miałoby sensu, dlatego przygotowywanie podpowiedzi zostało w tym przypadku zablokowane przez dostarczenie za pośrednictwem parametru `wpis` wartości `false`).

Wyjaśnienia wymaga jeszcze wyrównywanie poziomów podpowiedzi. Gdy po sczytaniu znaku mnożenia, dzielenia czy potęgowania, przygotowywany jest następny poziom do pracy, funkcja znajdująca się za wspomnianymi znakami działania jest analizowana na następnym poziomie rozkładu funkcji, co oznacza, że obie funkcje: przed i za znakiem działania, różnią się między sobą numerami poziomów rozkładu. Jest to oczywiście zgodne z procesem szukania pochodnej, lecz kłopotliwe dla podpowiedzi. Pożądane jest, by podpowiedzi zawierające funkcje przed i za znakiem np. potęgowania, miały taki sam stopień trudności. Takie uporządkowanie podpowiedzi pozwala na bardziej racjonalne ich wyszukiwanie a tym samym poprawia zdecydowanie poziom edukacji. By tak się stało, w programie została utworzona procedura `poziom_wstecz`, która w utworzonych podpowiedziach poprawia numerację poziomów trudności w taki sposób, że przypisuje im różnicę ich dotychczasowego numeru i bieżącego, z którego wywołana została ta procedura. Bieżący numer poziomu rozkładu jest przekazywany procedurze w parametrze poprzez wartość, dlatego można ją w procedurze

modyfikować. Poprawianie poziomów odbywa się w pętli `repeat..until`, której opuszczenie nastąpi po poprawieniu pierwszej podpowiedzi (poprawianie odbywa się wstecz) lub gdy pole `cz_zero` oraz zmienna logiczna `zero` będą posiadały różne wartości.

```
procedure poziom_wstecz(rozn:byte);
var pom5:nauka;
begin
     pom5:=t_biez; - przypisz zmiennej lokalnej pom5 adres ostatniej podpowiedzi
     if pom5<>nil then - czy utworzona została przynajmniej jedna podpowiedź?
     begin - jeśli tak, to...
          rozn:=pom5^.poziom-rozn; - wylicz różnicę między polem...
           poziom należącym do ostatniej podpowiedzi a bieżącym numerem poziomu
          repeat
               if pom5.cz_zero<>zero then break; - jeśli wartości...
                    pola cz_zero oraz zmiennej zero różnią się, opuść pętlę
               pom5.poziom:=pom5^.poziom-rozn; - przypisz polu...
                  poziom wartość będącą różnicą dotychczasowej wartości...
                  i wyliczonej różnicy w zmiennej rozn
               pom5:=pom5^.lewa - odczytaj adres poprzedniej podpowiedzi
          until pom5=nil; - opuść pętlę po poprawieniu pierwszej podpowiedzi
     end;
end;
```

Zmienna `zero` modyfikowana jest w ten sposób, że po sczytaniu w pętli bloku znaku dodawania lub odejmowania, jest jej przypisywana wartość przeciwna do obecnej. Dzięki temu, można było utworzyć granicę poprawiania poziomów przez procedurę `poziom_wstecz`, nie dopuszczając do poprawiania poziomów tym podpowiedziom, które posiadają funkcje zakończone znakiem dodawania lub odejmowania. Brzmi to dość zawile, dlatego najlepszym sposobem na zrozumienie tego zagadnienia będzie przykład ujęty w tabeli 1.

Tab. 1. Współudział procedury `poziom_wstecz` oraz zmiennej `zero` w numerowaniu poziomów trudności funkcji cząstkowych wykorzystanych w podpowiedziach

Funkcja elementarna i jej funkcje cząstkowe	Numery poziomów trudności		
	Niepoprawne, bez udziału procedury `poziom_wstecz` i zmiennej `zero`	Niepoprawne, przy współudziale procedury `poziom_wstecz`, ale bez udziału zmiennej `zero`	Poprawne, przy współudziale procedury `poziom_wstecz` i zmiennej `zero`
$e+[x^2+\sin(x)\cdot 3]$	0	0	0
$\sin(x)\cdot 3$	2	2	2
3	3	3	3
$\sin(x)$	2	3	3
x^2	2	3	2
2	3	4	3
x	3	4	3
e	1	4	1

Na podstawie tabeli 1 można zaobserwować różnice między błędnymi numerami poziomów a poprawnymi, zaznaczonymi w kolorze zielonym. Przy braku udziału procedury

poziom_wstecz, podpowiedzi zawierające funkcje przed i za znakiem mnożenia nie są na tych samych poziomach trudności, z kolei przy współudziale wspomnianej procedury, ale bez kontroli zmiennej zero, poziomy podpowiedzi zawierające funkcję x^2 oraz jej funkcje elementarne, zostały bezprawnie zmodyfikowane, co może mieć wpływ na niewłaściwą kolejność ukazywania się podpowiedzi w procesie edukacji.

Do procedury dostarczane są dwa łańcuchy znaków typu PChar, które zawierają odpowiednio: funkcję i jej pochodną. Trzecią informacją jest numer poziomu, z którego procedura została wywołana, co jest porównywalne ze stopniem trudności podpowiedzi. Jest to ważna informacja, gdyż na jej podstawie będą wybierane podpowiedzi przez system edukacji. Czwartym i ostatnim parametrem dostarczanym procedurze jest znak typu char zawierający kod podpowiedzi. By podpowiedzi mogły być przygotowane, została zadeklarowana struktura podpowiedzi, której elementy tworzone są dynamicznie, przy każdym wywołaniu procedury i stanowią tzw. listę dwukierunkową. Jej deklaracja jest następująca:

```
type
    nauka=^tablica_podpowiedzi;
    tablica_podpowiedzi=record
                                lipo:byte;
                                fun_A:PChar;
                                fun_B:PChar;
                                poziom:byte;
                                cz_zero:boolean;
                                podp1:string[18];
                                lewa:nauka;
                                prawa:nauka
                        end;
```

Znaczenie poszczególnych pól jest następujące:

- lipo – liczba porządkowa podpowiedzi,
- fun_A – funkcja cząstkowa,
- fun_B – pochodna funkcji cząstkowej,
- poziom – poziom trudności podpowiedzi równoznaczny poziomowi rozkładu funkcji,
- cz_zero – pole pomocnicze, wykorzystane podczas poprawiania poziomów podpowiedzi,
- podp1 – ścieżka do pliku podpowiedzi,
- lewa, prawa – pola służące do wiązania elementów nauka z poprzednikiem i następnikiem.

Do obsługi opisanej struktury o nazwie nauka, służą trzy zmienne globalne, które wykorzystane będą przez inne moduły programu. Deklaracja ich jest następująca:

```
public
      t_nau,t_biez,t_pom:nauka;
```

Po wywołaniu procedury i stwierdzeniu, że dostarczona funkcja składa się przynajmniej z jednego znaku, przydzielona zostaje pamięć dla łańcucha lokalnego pom1, którego rozmiar jest odpowiedni dla typu PChar i ilości znaków tworzących otrzymaną funkcję. Po skopiowaniu do łańcucha pom1 znaków funkcji, z łańcucha pom1 usuwany jest początkowy i końcowy znak działania, czyli: dodawania, odejmowania, mnożenia, dzielenia oraz potęgowania – jedynie początkowy znak odejmowania jest w łańcuchu zachowany.

```
if lan_fun=nil then exit; - jeśli łańcuch lan_fun jest pusty, opuść procedurę
```

```
if lan_poch=nil then exit; - jeśli łańcuch lan_poch jest pusty, opuść procedurę
w1:=StrLen(lan_fun); - z ilu znaków składa się łańcuch lan_fun?
if w1>0 then - czy liczba znaków jest większa od zera?
begin - jeśli tak, to...
     GetMem(pom1,w1*sizeof(PChar)); - przydziel pamięć dla łańcucha…
                                      lokalnego...
                                      pom1 o rozmiarze odpowiednim do liczby znaków i typu PChar
     StrCopy(pom1,lan_fun); - skopiuj zawartość łańcucha lan_fun do pom1
     znak:=pom1[w1-1]; - odczytaj ostatni, niepusty znak z łańcucha pom1
     if (znak='*') or (znak='+') or (znak='-') or (znak='/')
         or (znak='^') then pom1[w1-1]:=#0; - jeśli znak w zmiennej…
                                              znak...
                                              jest jednym ze znaków działania, zastąp go znakiem pustym
     znak:=pom1[0]; - odczytaj pierwszy znak z łańcucha pom1
     if (znak='*') or (znak='+') or (znak='/') or (znak='^')
     then for v1:=0 to (w1-1) do pom1[v1]:=pom1[v1+1]; - jeśli ...
     pierwszy znak w łańcuchu rozpoczyna się od znaku mnożenia, dodawania, dzielenia...
     lub potęgowania, usuń go przez zastąpienie każdego znaku znakiem następnym
```

Omawiana procedura otrzymuje funkcję i pochodną w stanie surowym, dlatego usunięcie zbędnych znaków jest nieodzowne. W pętli `while..do`, zawartość łańcucha `pom1` jest porównywana z już istniejącymi podpowiedziami zapisanymi w kolejnych elementach `nauka`; jeśli funkcja ta znajduje się w jednym z elementów podpowiedzi, zostaje ona pominięta, by jej nie powtarzać. Przed opuszczeniem procedury instrukcją `exit`, pamięć zajmowana przez łańcuch `pom1` musi zostać zwolniona.

```
tab1:=t_nau; - do lokalnej zmiennej tab1 wczytaj adres pierwszej podpowiedzi
while tab1<>nil do - wykonaj krok pętli, póki zmienna tab1 nie ma adresu pustego
begin
    if StrComp(tab1^.fun_A,pom1)=0 then - czy zawartości…
                                          łańcuchów fun_A oraz pom1 są takie same?
    begin - jeśli tak, to...
         FreeMem(pom1,w1*sizeof(PChar)); - zwolnij pamięć zajmowaną...
                                           przez łańcuch pom1
         exit - opuść procedurę
    end;
    tab1:=tab1^.prawa - odczytaj adres następnej podpowiedzi
end;
```

Jeśli funkcja nie jest powtórzona, w pamięci zostaje utworzony nowy element `nauka`, czyli nowa podpowiedź. Po zliczeniu ilości niepustych znaków z łańcucha lokalnego `pom1`, dla łańcucha `fun_A`, będącego polem utworzonego elementu `nauka`, przydzielona zostaje pamięć. Do przygotowanego łańcucha `fun_A` skopiowana zostaje zawartość łańcucha `pom1`, po czym pamięć zajmowana przez łańcuch lokalny zostaje uwolniona. Polu `fun_B` przypisany zostaje adres pusty na wypadek, gdyby okazało się, że dostarczony łańcuch `fun_B` nie posiada pochodnej. Kolejnym uzupełnionym polem w utworzonym elemencie jest `lipo`, do którego trafia numer kolejny podpowiedzi od zmiennej `lp7`, będącej licznikiem podpowiedzi.

```
new(tab1); - utwórz w pamięci nowy element nauka
tab1^.fun_B:=nil; - przypisz polu – łańcuchowi fun_B adres pusty
w2:=StrLen(pom1); - z ilu znaków składa się łańcuch lokalny pom1?
GetMem(tab1^.fun_A,w2*sizeof(PChar)); - przydziel pamięć dla łańcucha fun_A
StrCopy(tab1^.fun_A,pom1); - skopiuj zawartość łańcucha pom1 do fun_A
FreeMem(pom1,w1*sizeof(PChar)); - zwolnij pamięć zajmowaną…
```

przez łańcuch pom1

```
tab1.lipo:=lp7; - przypisz polu lipo wartość ze zmiennej lp7…
                  – numer porządkowy podpowiedzi
inc(lp7); - zwiększ wartość w zmiennej lp7 o jeden, przygotowując ją dla następnego
            wpisu
```

Kolejny dostarczony łańcuch zawiera pochodną, którą także należy zachować w polu utworzonego elementu podpowiedzi. Tak jak w przypadku funkcji, dla łańcucha lokalnego `pom1` przydzielona zostaje pamięć, której rozmiar odpowiada liczbie znaków, z których składa się otrzymana pochodna. Gdy łańcuch `pom1` jest przygotowany, zostają do niego skopiowane znaki pochodnej. Podobnie jak w przypadku funkcji, pierwszy i ostatni znak działania zostają z łańcucha usunięte, jedynie pierwszy znak odejmowania, z oczywistych względów, pozostaje w łańcuchu. Po zliczeniu niepustych znaków z łańcucha `pom1`, dla łańcucha `fun_B`, będącego kolejnym polem elementu podpowiedzi, zostaje przydzielona pamięć o wielkości odpowiedniej do przechowania w niej wszystkich znaków pochodnej. Po skopiowaniu do niego zawartości łańcucha `pom1`, pamięć zajmowana przez ten łańcuch zostaje uwolniona.

Pole `podp1` jest przewidziane do zapamiętania ścieżki, do pliku zawierającego podpowiedź dla aktualnego przykładu. Przypisanie właściwej ścieżki odbywa się w instrukcji wyboru `case`, w której wybór dokonywany jest na podstawie znakowej informacji dostarczonej w parametrze o nazwie `pod`.

```
case pod of
     'a':tab1^.podp1:='.\Obrazki\p1.bmp';
     'b':tab1^.podp1:='.\Obrazki\p2.bmp';
     'c':tab1^.podp1:='.\Obrazki\p3.bmp';
     'd':tab1^.podp1:='.\Obrazki\p4.bmp';
     'e':tab1^.podp1:='.\Obrazki\p5.bmp';
     'f':tab1^.podp1:='.\Obrazki\p6.bmp';
     'g':tab1^.podp1:='.\Obrazki\p7.bmp';
     'h':tab1^.podp1:='.\Obrazki\p8.bmp';
     'i':tab1^.podp1:='.\Obrazki\p9.bmp';
     'j':tab1^.podp1:='.\Obrazki\p10.bmp';
     'k':tab1^.podp1:='.\Obrazki\p11.bmp';
     'l':tab1^.podp1:='.\Obrazki\p22.bmp';
     'm':tab1^.podp1:='.\Obrazki\p23.bmp';
     'n':tab1^.podp1:='.\Obrazki\p24.bmp';
     '0':tab1^.podp1:='.\Obrazki\p12.bmp';
     '1':tab1^.podp1:='.\Obrazki\p13.bmp';
     '2':tab1^.podp1:='.\Obrazki\p14.bmp';
     '3':tab1^.podp1:='.\Obrazki\p15.bmp';
     '4':tab1^.podp1:='.\Obrazki\p16.bmp';
     '5':tab1^.podp1:='.\Obrazki\p18.bmp';
     '6':tab1^.podp1:='.\Obrazki\p17.bmp';
     '7':tab1^.podp1:='.\Obrazki\p19.bmp';
     '8':tab1^.podp1:='.\Obrazki\p20.bmp';
     '9':tab1^.podp1:='.\Obrazki\p21.bmp';
     else tab1^.podp1:='';
end;
pod:=#0;
```

Po uzupełnieniu pól, pozostaje jeszcze związanie aktualnego elementu `nauka` z ewentualnym poprzednikiem – jeśli istnieje, również i on jest wiązany z bieżącym elementem, bowiem elementy te stanowią tzw. listę dwukierunkową.

```
tab1^.poziom:=gdzie; - przypisz polu poziom numeru poziomu,...
                       z którego procedura została wywołana
tab1^.cz_zero:=zero; - przypisz polu cz_zero wartość zmiennej zero
tab1^.lewa:=t_biez; - zwiąż element nowej podpowiedzi z elementem poprzednim
tab1^.prawa:=nil; - jest to ostatni element listy, dlatego do pola prawa wpisz nil
if t_nau=nil then t_nau:=tab1 else t_biez^.prawa:=tab1; - jeśli jest ...
to pierwszy element listy, przypisz jego adres zmiennej t_nau – pierwsza podpowiedź, jeśli nie...
                                    zwiąż go z ostatnim elementem, zapamiętanym w zmiennej t_biez
t_biez:=tab1; - zmienna t_biez pamięta adres ostatniego elementu podpowiedzi
```

XI. Blok wartości funkcji – `Unit8`

Zadaniem tego bloku jest wyliczenie wartości funkcji otrzymanej w parametrze i zwrócenie tej wartości za pośrednictwem głównej funkcji bloku. Dodatkowym jego zadaniem jest kontrola błędów. W bloku zadeklarowana została klasa o nazwie `wartość` i główna funkcja bloku – `wynik`, której dostarczony został łańcuch znaków stanowiących funkcję do wyliczenia oraz, jako drugi parametr, zmienną `iks8`, której wartość zastępuje wszystkie znaki 'x' w tym łańcuchu. Trzeci parametr – `rodzaj8`, narzuca blokowi typ zwracanego wyniku, który w przypadku funkcji trygonometrycznych, może być wyliczony w radianach lub stopniach. Parametr ten posiada wartość domyślną `false`, który pozwala na opcjonalne wywołanie funkcji `wynik` – jeśli podczas wywołania funkcji parametr ten został pominięty, jego domyślna wartość `false` narzuci wewnętrznym procedurom bloku obliczanie wartości funkcji trygonometrycznych w radianach.

Deklaracja klasy jest następująca:

```
unit Unit8;

interface

uses SysUtils, Math;

type
    wartosc = class
    function wynik
             (const lancuch:array of char;
              const iks8:double
              rodzaj8:boolean=false):double;

    private

    public
          bl:boolean;
          blw:byte;

end;
```

Do obsługi klasy zadeklarowano zmienną o nazwie `wart1`:

```
  var wart1:wartosc;
```

Budowa głównej funkcji `wynik`, poza jej wewnętrznymi procedurami, jest bardzo podobna do procedury `blok_glowny` z bloku pochodnej. Jej główna różnica to inna konstrukcja struktury, która wynika z innego typu wartości przechowywanych w polach kolejnych jej elementów. Poza tą różnicą, sposób rozkładu otrzymanej w parametrze funkcji jest prawie taki sam, z kilkoma wyjątkami, które dokładnie opisano poniżej.

Budowa struktury bloku jest następująca:

```
type wagony=^budowa;
     budowa=record
                 t1:PChar;
                 t2:PChar;
                 tym:double;
                 ost:double;
                 p_tym:double;
                 p_ost:double;
                 fw:boolean;
                 p_fw:boolean;
                 fn:boolean;
```

```
                dzial:array[0..1] of integer;
                pozc:integer;
                poprz:wagony;
                nast:wagony
    end;
```

Przeznaczenie pól zaprezentowanej struktury jest następujące:
`t1` – łańcuch znakowy, który alokowany jest dynamicznie w pamięci o rozmiarze zależnym od przechowywanej w nim zawartości. Jest to pierwszy element rozkładu funkcji, którego zawartość analizowana jest w ramach jednego poziomu;
`t2` – drugi łańcuch znakowy, również alokowany dynamicznie w pamięci. Jest to drugi element rozkładu funkcji, przygotowywany do zamiany z postaci znakowej na liczbową;
`tym` – pole liczbowe, przeznaczone do zapamiętania wartości tymczasowej, wyliczonej w bieżącym poziomie;
`ost` – pole liczbowe, przeznaczone do zapamiętania wartości ostatecznej na danym poziomie;
`p_tym` – pole liczbowe, przechowujące wyliczoną wartość tymczasową w poprzednim poziomie;
`p_ost` – pole liczbowe, przechowujące wyliczoną wartość ostateczną w poprzednim poziomie;
`fw` – pole logiczne, którego wartość decyduje o wywołaniu wewnętrznych podprogramów:

- `false` – wykrycie tej wartości, gdy odczytany aktualnie znak jest jednym ze znaków działania lub pusty lub gdy pole `fn` posiada wartość `true`, pozwala na zamianę łańcucha znakowego do postaci liczbowej;
- `true` – wykrycie tej wartości po odczytaniu znaku dodawania, odejmowania lub bezpośrednio po przekształceniu łańcucha `t2` z postaci znakowej na liczbową, powoduje wykonanie działania na dwóch liczbach;

`p_fw` – pole logiczne, które decyduje o tym, jaką wartość: tymczasową czy ostateczną, uzyskaną w poprzednim poziomie, należy wykorzystać do obliczeń na poziomie bieżącym;
`fn` – pole logiczne, niosące informację o gotowości wykonania działania mnożenia, dzielenia czy potęgowania na dwóch liczbach. Wykrycie wartości `true` w tym polu pozwala na wykonanie obliczeń;
`dzial` – liczbowa tablica dwuelementowa, przechowująca wartości znaków działania;
`pozc` – pole liczbowe, zapamiętujące pozycję w łańcuchu `t1`, od której, po powrocie do poprzednika elementu `wagony`, ma rozpocząć się przepisywanie znaków z łańcucha z pola `t1` do łańcucha z pola `t2`;
`poprz`, `nast` – pola służące do wiązania ze poprzednikiem i następnikiem elementu `wagony`.

Po zbadaniu, że dostarczony łańcuch znakowy nie jest pusty, dla pierwszego elementu `wagony` przydzielona zostaje pamięć, po czym zawartość otrzymanego w parametrze łańcucha zostaje przepisana do łańcucha z pola `t1`, które jest polem utworzonego elementu.

```
bl:=false; - przypisz zmiennej logicznej błędu wartość false
luka:=false; - zmiennej logicznej luka, pomocnej przy odrzucaniu ewentualnych,...
                początkowych znaków pustych, wartość false

blw:=0; - przypisz zmiennej kodu błędu wartość zero
if lancuch<>nil then k:=StrLen(lancuch) else - jeśli łańcuch lancuch nie jest...
                pusty, zlicz ilość niepustych znaków do zmiennej k, w przeciwnym przypadku...

begin - jeśli łańcuch jest pusty (czyli zmienna łańcuchowa lancuch posiada adres pusty), to...
     Result:=0.0; - za pomocą literału Result zwróć przez funkcję wynik wartość zero
```

```
      exit - opuść funkcję
end;
new(biez8); - utwórz w pamięci pierwszy element wagony
GetMem(biez8^.t1,k*sizeof(PChar)); - dla łańcucha z pola t1 przydziel pamięć…
    o wielkości proporcjonalnej dla typu PChar i ilości znaków zapamiętanych w zmiennej k
StrCopy(biez8^.t1,lancuch); - skopiuj zawartość łańcucha lancuch do łańcucha…
                                  z pola t1 bieżącego elementu wagony
with biez8^ do - w instrukcji wiążącej przypisz wartości startowe pozostałym polom...
                 elementu wagony
begin
      t2:=nil;
      tym:=0.0;
      ost:=0.0;
      p_tym:=0.0;
      p_ost:=0.0;
      fw:=false;
      p_fw:=false;
      fn:=false;
      dzial[0]:=0;
      dzial[1]:=0;
      pozc:=0;
      poprz:=nil;
      nast:=nil
end;
```

Gdy pierwszy element `wagony` jest już gotowy, w pętli `while..do` rozpoczyna się proces rozkładu funkcji. Rozpoczęcie kroku pętli uzależnione jest od niepustego adresu zawartego we wskaźniku `biez8` zawierającego adres pierwszego elementu `wagony`. Proces sczytywania znaków z łańcucha z pola `t1` do łańcucha z pola `t2`, podobnie jak w bloku `pochodna`, został zamknięty w instrukcji `try..except..end`. Po pierwszym zidentyfikowaniu znaku potęgowania, funkcja `czy_trygon` sprawdza, czy potęgowanie dotyczy funkcji trygonometrycznej. Jeśli tak, to za pomocą funkcji `trygon` potęga zostaje przeniesiona na koniec argumentu – mechanizm ten został dokładnie opisany w bloku `pochodna`.

Po sczytaniu w pętli jednego ze znaków działania, przy dodatkowym warunku, że nie jest to pierwszy znak sczytany z łańcucha, o czym informuje zmienna `k1` o wartości większej od zera, reakcja programu jest nieco inna niż we wspomnianym bloku `pochodna`. W instrukcji warunkowej sprawdzana jest wartość pola `fw`, jeśli wynosi `false`, oznacza to, że łańcuch z pola `t2` można zamienić na wartość liczbową, dlatego wywołana zostaje procedura `alicz`, która zajmie się tą zamianą – wynik zamiany do postaci liczby zmiennoprzecinkowej zostaje zapamiętany w polu `tym` bieżącego elementu `wagony`. Po opuszczeniu procedury, pamięć zajmowana przez łańcuch z pola `t2` zostaje uwolniona a pole `fw` – zmodyfikowane do wartości `true`, pozwalając na wykonanie obliczeń, jednak pod warunkiem, gdy sczytany znak okazał się dodawaniem lub odejmowaniem. Jeśli warunek został spełniony, wywołana zostaje procedura `licz_licz`, która wykona działanie matematyczne na dwóch liczbach, zapamiętanych w polach bieżącego elementu `wagony`, jednak rodzaj działania nie jest podyktowany sczytanym obecnie znakiem działania, ale wartością znaku zapamiętanego w tablicy `dzial`. Po opuszczeniu procedury `licz_licz`, do tablicy `dzial` dopisana zostaje wartość bieżącego znaku działania, lecz by zachować kolejność działań, wartość ta zostaje dopisana tak, by nie zastępować już istniejącej wartości znaku działania zapisanej wcześniej.

Może się zdarzyć, że po sczytaniu dowolnego znaku działania, pole `fw` będzie posiadało wartość `true` – w tej sytuacji, procedura `alicz` nie zostanie wywołana, a co za tym idzie, pamięć jest wciąż zajmowana przez łańcuch z pola `t2`, dlatego przed opuszczeniem opisywa-

nej instrukcji warunkowej sprawdzana jest zajętość tego łańcucha, jeśli jego zmienna łańcuchowa nie posiada adresu pustego, pamięć zajmowana przez łańcuch zostaje zwolniona.

```
if ((k=42) or (k=43) or (k=45) or (k=47) or (k=94)) and (k1>0) then
begin - jeśli sczytanym znakiem jest jeden ze znaków działania i nie jest to pierwszy znak, to...
     if not biez8^.fw then alicz(biez8); - jeśli pole fw posiada wartość false...
                                           wywołaj procedurę alicz

     if (k=43) or (k=45) then - czy sczytanym znakiem jest dodawanie...
                                lub odejmowanie?

        if biez8^.fw then - jeśli tak, to czy wartość pola fw wynosi true?
        begin - jeśli tak, to...
             licz_licz(biez8); - wywołaj procedurę licz_licz
             biez8^.fw:=false; - przypisz polu fw wartość false
             if biez8^.dzial[0]>0 then - czy pole zero tablicy dzial...
                                         posiada wpisany znak działania?

                biez8^.dzial[1]:=biez8^.dzial[0]; - jeśli tak, to przenieś go...
                                                    do następnego pola w tablicy

             biez8^.dzial[0]:=k; - wartość bieżącego znaku działania zapamiętaj w...
                                   tablicy dzial pod indeksem zerowym

             if biez8^.t2<>nil then - czy łańcuch z pola t2 zajmuje pamięć?
             begin - jeśli tak, to...
                  jp:=StrLen(biez8^.t2); - z ilu niepustych znaków...
                                           składa się łańcuch?

                  FreeMem(biez8^.t2,jp*sizeof(PChar)); - zwolnij pamięć...
                                                         zajmowaną przez łańcuch

                  biez8^.t2:=nil przypisz zmiennej łańcuchowej t2 adres pusty
             end;
        end;
end;
```

Gdy sczytany znak z łańcucha z pola `t1` okazał się pusty a zmienna błędu ma wartość `false`, za pomocą procedury `licz_licz` wykonane jest obliczenie matematyczne, którego wynik zapamiętany zostaje w polu `ost` będący wartością ostateczną, wyliczoną na tym poziomie rozkładu funkcji. Przedtem, w instrukcji warunkowej sprawdzona zostaje wartość pola `fw` – jeśli wynosi `false`, przed dokonaniem obliczenia wywołana zostaje procedura `alicz`, która zamieni łańcuch znakowy z pola `t2` na liczbę, zapamiętując ją w polu `tym`. Po wykonaniu obliczeń, uwalniana zostaje pamięć zajmowana przez łańcuchy z pól `t1` i `t2`. Jeśli istnieje poprzednik bieżącego elementu `wagony`, w jego polach zostają zapamiętane te wartości z pól bieżącego elementu, które muszą zostać zachowane, gdy bieżący element zostanie z pamięci usunięty. Są to: wyliczona wartość – ostateczna i tymczasowa, pochodzące z pól `ost` i `tym`, a także wartość z pola `fw`. Do zmiennej `k1` wczytana zostaje wartość z pola `pozc`, wskazując miejsce, od którego ma rozpocząć się przepisywanie znaków z łańcucha dostępnego poprzez pole `t1` do łańcucha z pola `t2`, należących do poprzednika. Po usunięciu bieżącego elementu `wagony`, jego poprzednik staje się elementem bieżącym.

```
if k=0 then - czy odczytany znak z łańcucha z pola t1 jest pusty?
   if not bl then - jeśli tak, to czy nie ma błędu?
   begin jeśli -jeśli nie ma błędu, to...
        if biez8^.fw then - czy pole fw posiada wartość true?
        begin - jeśli tak, to...
             licz_licz(biez8); - wykonaj obliczenie matematyczne…
                                 – wynik w polu ost
```

```
                biez8^.fw:=false - przypisz polu fw wartość false – obliczenie…
                                   wykonane
            end else - jeśli nie, to...
                begin
                    alicz(biez8); - zamień łańcuch z pola t2 na liczbę – wynik…
                                    zapamiętaj w polu tym bieżącego elementu wagony
                    if biez8^.fw then - czy zamiana przebiegła pomyślnie?
                    begin - jeśli tak, to...
                        licz_licz(biez8); - wykonaj obliczenie matematyczne ...
                                            – wynik w polu ost
                        biez8^.fw:=false - przypisz polu fw wartość false...
                                           zaznaczając, że obliczenie zostało...
                                           wykonane
                    end;
                end;
            if biez8^.t1<>nil then - czy łańcuch z pola t1 istnieje w pamięci?
            begin - jeśli tak, to...
                jp:=StrLen(biez8^.t1); - z ilu znaków składa się łańcuch z pola t1?
                FreeMem(biez8^.t1,jp*sizeof(PChar)); - zwolnij pamięć ...
                                zajmowaną przez łańcuch dostępny przez pole t1
                biez8^.t1:=nil - zmiennej łańcuchowej – polu t1 przypisz adres pusty
            end;
            if biez8^.t2<>nil then - (jak wyżej, tylko dla łańcucha z pola t2)
            begin
                jp:=StrLen(biez8^.t2);
                FreeMem(biez8^.t2,jp*sizeof(PChar));
                biez8^.t2:=nil
            end;
            pom8:=biez8^.poprz; - odczytaj adres poprzednika
            if pom8<>nil then - czy poprzednik istnieje?
            begin - jeśli tak, to...
                k1:=pom8^.pozc; - zapamiętaj w zmiennej k1 pozycję, od której...
                    będzie sczytywany znak w łańcuchu t1, należący do poprzednika
                pom8^.p_tym:=biez8^.tym; - zachowaj wartość z pola tym...
                                           w polu p_tym poprzednika
                pom8^.p_ost:=biez8^.ost; - jak wyżej, dla pola ost
                pom8^.p_fw:=biez8^.fw; - jak wyżej, dla pola fw
                pom8^.nast:=nil - poprzednik nie będzie miał następnika
            end else zk8:=biez8^.ost; - jeśli poprzednik nie istnieje, w zmiennej zk8...
                                zapamiętaj wartość z pola ost bieżącego elementu wagony...
                                jest to końcowy wynik funkcji
                dispose(biez8); - usuń z pamięci element bieżący wagony
            biez8:=pom8 - element poprzedni jest teraz bieżącym
        end;
```

Po wykryciu znaku nawiasu otwierającego czy znaku akcji, a więc mnożenia, dzielenia lub potęgowania, uruchamiana zostaje grupa instrukcji, których zadaniem jest przygotowanie następnego poziomu rozkładu do pracy. W każdym z wymienionych przypadków, utworzony zostaje w pamięci nowy element `wagony`, a w pętlach programowych, z łańcucha z pola `t1`, wydzielona zostaje ta część funkcji, która ma być analizowana w następnym poziomie rozkładu. Zarówno sposób przygotowywania następnego poziomu, jak i zastosowane warunki opuszczenia pętli programowych, w których wydzielana jest funkcja potęgująca, mianownik ułamka czy funkcja zamknięta w nawiasie są identyczne z tymi samymi, zastosowanymi w bloku znajdowania pochodnej, dlatego ich opis zostanie pominięty.

Po zwiększeniu wartości w zmiennej k1 o jeden, umożliwiając wskazanie następnego znaku w łańcuchu z pola t1, sprawdzany jest bieżący element wagony – jeśli istnieje oraz nie ma błędu, sprawdzana jest wartość pola fn (pole to ustawiane jest w pozycji true po odczytaniu znaku mnożenia, dzielenia lub potęgowania), jeśli jego wartość wynosi true, w kolejnej instrukcji warunkowej sprawdzana jest wartość pola fw – wartość false pozwala na zamianę łańcucha znakowego dostępnego przez pole t2 na liczbę, dlatego wywołana zostaje procedura alicz, która uzyskaną liczbę zmiennoprzecinkową zapamiętuje w polu tym bieżącego elementu wagony. Teraz można już wykonać wyliczenie matematyczne przez procedurę licz_licz, która wynik działania umieści w polu tym bieżącego elementu. Po wykonaniu obliczeń, polom logicznym fn i p_fw przypisane zostają wartości false, zaznaczając tym samym, że działanie mnożenia, dzielenia lub potęgowania na liczbie z bieżącego i poprzedniego poziomu rozkładu, zostało wykonane.

Gdy sczytanym znakiem okazał się nawias zamykający, wywołana zostaje procedura dwa, która wyliczy wartość funkcji trygonometrycznej, umieszczając wynik w polu tym bieżącego elementu wagony. Po opuszczeniu procedury, polu fw przypisana zostaje wartość true, zaznaczając tym samym gotowość do wykonania obliczeń przez procedurę licz_licz.

Ostatnia instrukcją zamkniętą w pętli while..do jest zbadanie, czy zmienna błędu bl nie posiada przypadkiem wartości true – jeśli tak, pętla zostaje opuszczona instrukcją break.

```
inc(k1); - zwiększ wartość w zmiennej k1 o jeden
if biez8<>nil then - czy bieżący element wagony istnieje?
   if not bl then - jeśli tak, to czy nie ma błędu?
   begin - jeśli nie ma błędu, to...
       if biez8^.fn then - czy wartość z pola fn wynosi true?
        begin - jeśli tak, to...
             if not biez8^.fw then alicz(biez8); - jeśli wartość z pola...
                                  fw wynosi false to wywołaj procedurę alicz,...
                                  by zamienić łańcuch znakowy z pola t2 na liczbę
             licz_licz(biez8); - wykonaj jedno z działań akcji, czyli mnożenia,..
                               dzielenia lub potęgowania i umieść wynik w polu tym ...
                               bieżącego elementu wagony
             biez8^.p_fw:=false; - przypisz polu p_fw wartość false
             biez8^.fn:=false - przypisz polu fn wartość false
        end;
        if k=41 then - czy odczytanym znakiem jest nawias zamykający?
           if not bl then - jeśli tak, to czy nie ma błędu?
           begin - jeśli nie ma błędu, to...
                dwa(biez8); - wylicz wartość z funkcji trygonometrycznej
                biez8^.fw:=true - zaznacz gotowość do obliczeń
           end;
   end;
   if bl then break - jeśli wystąpił błąd, opuść główną pętlę
end;
```

Jeśli po opuszczeniu pętli while..do zmienna błędu bl posiada wartość false, wynik końcowy powinien być dostępny w zmiennej zk8, dlatego za pośrednictwem zmiennej Result, wynik końcowy zwracany jest przez funkcję wynik. Gdyby jednak zmienna błędu została ustawiona w pozycji true przed opuszczeniem funkcji wynik, wszystkie elementy wagony, które zajmują jeszcze pamięć, muszą tę pamięć uwolnić. Ponieważ nie wiadomo, na

jakim etapie pracy funkcji pojawił się błąd, w pętli `while..do` wyszukany zostaje ostatni element `wagony`. Gdy jest już ustalony koniec listy, którą tworzą elementy, wszystkie one wraz z łańcuchami, dostępnymi przez pola `t1` i `t2`, uwalniają pamięć, począwszy od ostatniego elementu a skończywszy na pierwszym. Po uwolnieniu pamięci, funkcja zwróciwszy wartość zero, kończy swoją pracę.

```
if not bl then Result:=zk8 - jeśli nie ma błędu, zwróć wartość końcową…
                             ze zmiennej zk8
else - gdy istnieje błąd, to...
begin
     pom8:=nil; - przypisz zmiennej pom8 identyfikator adresu pustego
     while biez8<>nil do - wykonaj krok pętli, póki zmienna biez8 nie ma adresu…
                           pustego
     begin
          pom8:=biez8; - przypisz zmiennej pom8 adres elementu wagony...
                         ze zmiennej biez8
          biez8:=biez8^.nast - wpisz tej samej zmiennej adres następnego elementu
     end;
     while pom8<>nil do - wykonaj krok pętli, póki zmienna biez8 nie ma adresu pustego
     begin
          if pom8^.t1<>nil then - czy łańcuch z pola t1 nie jest pusty?
          begin - jeśli nie, to...
               k1:=StrLen(pom8^.t1); - zlicz ilość niepustych znaków z łańcucha…
                                       z pola t1
               FreeMem(pom8^.t1,k1*sizeof(PChar)) - uwolnij pamięć...
                    zajmowaną przez łańcuch z pola t1, o wielkości odpowiedniej...
                    dla typu PChar i ilości znaków
          end;
          if pom8^.t2<>nil then - (jak wyżej, dla łańcucha z pola t2)
          begin
               k1:=StrLen(pom8^.t2);
               FreeMem(pom8^.t2,k1*sizeof(PChar))
          end;
          biez8:=pom8^.poprz; - zapamiętaj w zmiennej biez8 adres poprzednika
          dispose(pom8); - zwolnij pamięć zajmowaną przez element wagony
          pom8:=biez8 - przypisz zmiennej pom8 adres poprzednika wagony
     end;
     Result:=0.0 - za pośrednictwem zmiennej Result zwróć wartość zero
end;
```

XI. 1. Zamiana łańcucha znakowego na liczbę – procedura `alicz`

Zadaniem procedury jest zamiana zawartości łańcucha znakowego typu `PChar` na liczbę zmiennoprzecinkową. Zamianie podlegają jedynie znaki nie zamknięte w nawiasie oraz znajdujące się między znakami działania, a więc znaki cyfr tworzących liczbę całkowitą lub zmiennoprzecinkową, znak 'e', znak 'π' lub zmienna pod postacią znaku 'x'. Łańcuch `t2` jest polem elementu `wagony`, który procedura otrzymuje w parametrze. By łatwiej można było identyfikować znaki cyfr wraz z ich separatorami dziesiętnymi, został zadeklarowany łańcuch stały, składający się z wartości tych znaków w kodzie ASCII. W pierwszej instrukcji warunkowej sprawdzana jest obecność łańcucha z pola `t2`. Gdy jego zmienna łańcuchowa zainicjowana jest wartością pustą, wskazuje to na błąd, dlatego procedura musi być opuszczona z ustawioną zmienną błędu w pozycji `true`. Gdy łańcuch z tego pola istnieje, dla lokalnego łańcucha typu `PChar` przydzielona zostaje pamięć, której rozmiar jest taki, jaki posiada łańcuch dostępny przez pole `t2`. W pętli `while..do` przepisywane są kolejno znaki z tego łańcucha do lokalnego łańcucha `w1`. Jeśli pierwszym sczytanym znakiem okazał się znak odejmowania, poza wpisaniem go do łańcucha lokalnego, jego obecność została zaznaczona poprzez ustawienie lokalnej zmiennej logicznej `min8` w pozycji `true`. Znaki cyfr wpisywane są do łańcucha lokalnego w ten sposób, że za wprowadzonym znakiem wstawiany jest znak pusty, by po każdym wpisie zamknąć łańcuch. Jeżeli w łańcuchu znajduje się wiele cyfr, przy następnym kroku pętli, nowy znak zastępuje wcześniej wstawiony znak pusty, a za wstawionym znakiem cyfry ponownie wstawiany jest znak pusty. Po przepisaniu do łańcucha lokalnego znaku cyfry, kolejnej zmiennej logicznej `n8` przypisana zostaje wartość `true`, zaznaczając tym, że w łańcuchu lokalnym znajduje się przynajmniej jedna cyfra. Gdy wartość tej zmiennej logicznej wynosi `false`, w kolejnej instrukcji warunkowej sprawdzany jest następny znak. Jeśli wartość tego znaku wynosi zero lub należy ona do jednego ze znaków działania, w instrukcji wyboru `case` identyfikowany jest znak sczytany w pętli. Rozpoznanie znaku 'x', 'e' lub 'π', powoduje wpisanie do lokalnej zmiennej liczbowej `lib1` wartości odpowiadającej rozpoznanemu znakowi, czyli:

- wartością znaku 'x' jest liczba dostarczona głównej funkcji `wynik` za pośrednictwem parametru o nazwie `iks8`;
- wartością znaku 'e' jest wynik funkcji bibliotecznej `exp` z parametrem równym jeden, co odpowiada liczbie 'e' podniesionej do potęgi jeden;
- wartością znaku 'π' jest wynik bezparametrowej funkcji o nazwie `pi`.

Po opuszczeniu instrukcji wyboru, do pola `tym` elementu `wagony` przepisana zostaje wartość ze zmiennej `lib1` – wcześniejsze rozpoznanie znaku minus zostaje uwzględnione we wpisywanej wartości. Gdyby znak nie został rozpoznany, do pola `tym` wpisana zostaje wartość zero, jednak bez ustawienia zmiennej błędu. Inna jest sytuacja, gdyby następny znak, za obecnie rozpoznanym w pętli, nie należał do znaku działania lub nie byłby pusty – w tej sytuacji pętla musi zostać opuszczona z ustawioną zmienną błędu w pozycji `true`.

Wartość `true` w zmiennej logicznej `n8` niesie informację, że w lokalnym łańcuchu `w1` znajdują się znaki przeznaczone do zamiany na wartość liczbową. W tej sytuacji zostaje on przetworzony na łańcuch typu `string`, a następnie na liczbę zmiennoprzecinkową. Ponieważ nie można mieć pewności, czy za znakiem cyfry nie znajduje się znak nie będący cyfrą lub zastosowany separator dziesiętny będzie zgodny z wybranym separatorem w ustawieniach regionalnych systemu operacyjnego, konwersję łańcucha typu `string` na liczbę zmiennoprzecinkową zamknięto w instrukcji `try..except..end`, by uchronić się przed komunikatem systemu operacyjnego, gdyby konwersja się nie powiodła. Jeśli podczas konwersji za-

istnieje błąd, wówczas zmienna logiczna błędu zostanie ustawiona w pozycji `true`, powodując obsłużenie błędu w ramach programu. Gdy konwersja przebiegła pomyślnie, do pola `tym` wpisana zostaje wartość liczbowa ze zmiennej `lib1`. W każdym przypadku bezbłędnej zamiany łańcucha znakowego na liczbę, do pola `fw` wpisana zostaje wartość `true`.

Tuż przed opuszczeniem procedury, pamięć zajmowana przez łańcuch `w1` oraz łańcuch dostępny przez pole `t2`, zostaje zwolniona. Ponieważ same zwolnienie pamięci nie powoduje automatycznego nadania zmiennej łańcuchowej identyfikatora adresu pustego, dlatego taki identyfikator trzeba przypisać w oddzielnej instrukcji przypisania, w tym przypadku zmiennej łańcuchowej `t2` – brak takiego identyfikatora doprowadziłby do nieokreślonych błędów.

```
procedure alicz(const wag1:wagony);
const liczby=[44,46,48,49,50,51,52,53,54,55,56,57]; - tablica stałych...
                składająca się z wartości w kodzie ASCII znaków cyfr, przecinka i kropki
var w1:PChar;
    lib1:double;
    ciag:string;
    v1,v2,v3,vw,vwp:integer;
    n8,min8:boolean;
begin
     if wag1^.t2=nil then - czy łańcuch z pola t2 istnieje w pamięci?
     begin - jeśli nie, to...
          bl:=true; - ustaw zmienną błędu w pozycji true
          blw:=12; - przypisz zmiennej kodu błędu stosowny numer błędu
          exit - opuść procedurę
     end;
     lib1:=0.0; - wyzeruj lokalną zmienną liczbową
     v1:=StrLen(wag1^.t2); - zlicz niepuste znaki z łańcucha z pola t2
     GetMem(w1,v1*sizeof(PChar)); - przydziel pamięć dla łańcucha lokalnego w1...
                          o wielkości proporcjonalnej dla typu PChar i ilości znaków
     w1[0]:=#0; - wpisz znak pusty na początku łańcucha
     vw:=ord(wag1^.t2[0]); - odczytaj wartość pierwszego znaku z łańcucha…
                               z pola t2
     v2:=0;
     v3:=0;
     n8:=false;
     min8:=false;
     while vw<>0 do - wykonaj krok pętli, póki wartość znaku jest różna od zera
     begin
          if (vw=45) and (v2=0) then - czy pierwszy znak to odejmowanie?
          begin - jeśli tak, to...
               min8:=true; - ustaw zmienną logiczną min8 w pozycji true
               w1[v3]:='-'; - wpisz znak odejmowania do łańcucha w1
               inc(v3); - zwiększ zmienną v3 o jeden, by wskazywała następną...
                          pozycję w łańcuchu w1
               w1[v3]:=#0; - na następnej pozycji w łańcuchu wpisz znak pusty
               inc(v2); - zwiększ zmienną v2 o jeden, by wskazywała następną...
                          pozycję w łańcuchu z pola t2
               vw:=ord(wag1^.t2[v2]); - odczytaj wartość następnego znaku
```

```
     continue - pomiń pozostałe instrukcje w pętli i rozpocznij następny...
                jej krok
end;
if vw in liczby then - czy odczytany znak zawiera się...
                       w tablicy liczby?
begin - jeśli tak, to...
     w1[v3]:=chr(vw); - wczytaj ten znak do tablicy w1 przez…
                        konwersję wartości znaku na znak typu Char
     inc(v3); - zwiększ zmienną v3 o jeden, by wskazywała następną...
                pozycję w łańcuchu w1
     w1[v3]:=#0; - na następnej pozycji w łańcuchu wpisz znak pusty
     n8:=true - przypisz zmiennej logicznej n8 wartość true
end else - jeśli odczytany znak nie zawiera się w tablicy liczby, to...
    if not n8 then - czy zmienna n8 nie wskazuje na odczytanie,...
                     w poprzednich  krokach pętli, znaku cyfry?
    begin - jeśli nie, to..
         vwp:=ord(wag1^.t2[v2+1]); - odczytaj wartość ...
                                     następnego znaku
         if (vwp=0) or (((vwp>40) and (vwp<48))
            or (vwp=94)) then - czy następny znak jest pusty lub ...
                                znakiem działania?
         begin - jeśli tak, to...
              case vw of
                   120: - czy wartość bieżącego znaku to ‘x’?
                   begin - jeśli tak, to...
                        lib1:=iks8; - do zmiennej lib1...
                              wpisz wartość parametru iks8
                        wag1^.fw:=true - ustaw pole fw ..
                                         w pozycji true
                   end;
                   101: - czy wartość bieżącego znaku to ‘e’?
                   begin - jeśli tak, to...
                        lib1:=exp(1); - do zmiennej lib1...
                                        wpisz wartość liczby ‘e’
                        wag1^.fw:=true - (jak wyżej)
                   end;
                   112: - czy wartość bieżącego znaku to ‘π’?
                   begin - jeśli tak, to...
                        lib1:=pi; - do zmiennej lib1...
                                    wpisz wartość liczby ‘π’
                        wag1^.fw:=true - (jak wyżej)
                   end;
              end;
              if min8 then wag1^.tym:=-lib1 - jeśli zmienna...
                   min8 wskazuje na odczytanie znaku odejmowania,...
                   wpisz do pola tym wartość ze zmiennej lib1...
                   z odwróconym znakiem
              else wag1^.tym:=lib1; - w przeciwnym przypadku...
                   do pola tym wpisz wartość ze zmiennej lib1...
                   bez zmiany znaku
              wag1^.fw:=true; - wpisz do pola fw wartość true
              break - opuść pętlę
         end else - gdy następny znak nie jest pusty...
                    i nie jest znakiem działania, to...
```

```
                        begin
                              bl:=true; - ustaw zmienną błędu w pozycji true
                              blw:=10; - zmiennej kodu błędu przypisz kod błędu
                              break - opuść pętlę
                        end
                  end;
            inc(v2); - zwiększ zmienną v2 o jeden
            vw:=ord(wag1^.t2[v2]) - odczytaj wartość następnego znaku z łańcucha…
                                    z pola t2
      end;
      if n8 then - czy zmienna n8 wskazuje na odczytanie znaku cyfry?
      begin - jeśli tak, to...
            ciag:=StrPas(w1); - dokonaj konwersji łańcucha PChar na string
            try - gdy spodziewamy się błędu, to...
               lib1:=StrToFloat(ciag); - do zmiennej lib1 wpisz wynik...
                  konwersji łańcucha typu string na liczbę zmiennoprzecinkową
            except - gdy wystąpi błąd konwersji, to...
                  bl:=true; - ustaw zmienną błędu w pozycji true
                  blw:=6 - zmiennej kodu błędu przypisz kod błędu
            end;
            if not bl then - czy nie ma błędu?
            begin - jeśli nie ma błędu, to...
                  wag1^.tym:=lib1; - do pola tym wpisz uzyskaną liczbę...
                                    ze zmiennej lib1
                  wag1^.fw:=true - wpisz do pola fw wartość true
            end;
      end;
      FreeMem(w1,v1*sizeof(PChar)); - zwolnij pamięć zajmowaną przez…
                                    łańcuch w1
      FreeMem(wag1^.t2,v1*sizeof(PChar)); - zwolnij pamięć zajmowaną przez...
                                          łańcuch z pola t2
      wag1^.t2:=nil - zmiennej łańcuchowej t2 przypisz identyfikator adresu pustego
end;
```

XI. 2. Zamiana funkcji trygonometrycznej na liczbę – procedura `dwa`

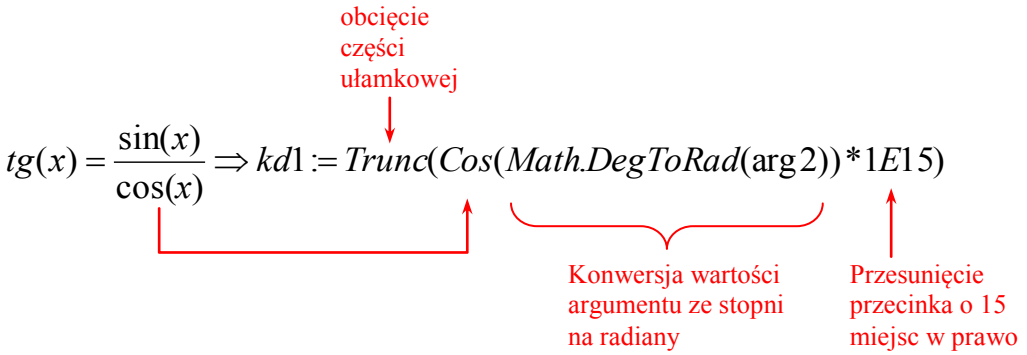
Zadaniem tej procedury, podobnie jak poprzednio opisywanej, jest zamiana łańcucha znakowego typu `PChar` na liczbę zmiennoprzecinkową. Jednak w tym przypadku, zadanie jest trudniejsze, ponieważ procedura musi rozpoznać nie jeden, lecz grupę znaków tworzących nazwę funkcji trygonometrycznej, cyklometrycznej, logarytmu oraz pierwiastka. Procedura otrzymuje w parametrze adres bieżącego elementu `wagony`. Ponieważ każdy z wymienionych przypadków musi składać się z części zamkniętej w nawiasie, dlatego w łańcuchu z pola `t2` otrzymanego elementu `wagony`, wyszukiwany jest pierwszy znak nawiasu otwierającego – jeśli funkcja biblioteczna `StrScan` nie znajdzie nawiasu, oznacza to, że łańcuch z pola `t2` nie zawiera funkcji, z której można byłoby wyliczyć wartość. W tej sytuacji procedura musi zostać opuszczona z ustawioną zmienną błędu w pozycji `true`. Jeśli nawias otwierający został znaleziony, w zmiennej łańcuchowej `tym1` będzie się znajdował adres łańcucha z tego pola, ale od miejsca znalezionego nawiasu. Skoro nawias został znaleziony, procedura może rozpocząć dalszą pracę. Kolejną czynnością jest uzyskanie odpowiedniej wartości liczbowej argumentu funkcji, która wyliczona została w poprzednich krokach programu. Wartość

ta może być dostępna zarówno w polu p_tym, jak i p_ost bieżącego elementu wagony. O tym, z którego pola ma być ona pobrana, decyduje ustawienie pola logicznego p_fw – jeśli zostało ustawione w pozycji true, właściwa liczba znajduje się w polu p_tym, w przeciwnym przypadku, w polu p_ost. W następnych instrukcjach ustalana jest pozycja znalezionego nawiasu otwierającego w łańcuchu z pola t2, która jest różnicą ilości wszystkich znaków zawartych w tym łańcuchu i znaków znajdujących się w wydzielonym łańcuchu pomocniczym tym1. Na podstawie wyliczonej różnicy można już wydzielić znaki stanowiące rdzeń funkcji. Może się zdarzyć, że dostarczona procedurze funkcja jest w całości zamknięta w nawiasie, dlatego w kolejnej instrukcji warunkowej sprawdzana jest wyliczona różnica. Jeśli instrukcja nie stwierdzi istnienia w łańcuchu z pola t2 rdzenia funkcji trygonometrycznej lub pierwiastka, do pola tym bieżącego elementu wagony wpisana zostaje wartość argumentu funkcji. Gdy rdzeń funkcji lub pierwiastek istnieje, za pośrednictwem lokalnej zmiennej łańcuchowej tym2 przydzielona zostaje pamięć dla łańcucha znakowego typu PChar o takim rozmiarze, by w łańcuchu zmieścił się znaleziony rdzeń funkcji lub sam znak pierwiastka. Po przydzieleniu pamięci, do łańcucha lokalnego tym2 skopiowana zostaje taka liczba znaków z łańcucha z pola t2, jaka wynika z wcześniej wyliczonej różnicy. W ten sposób, w łańcuchu tym2 znajdują się znaki rdzenia funkcji trygonometrycznej, logarytmu lub pierwiastka. Na podstawie jego zawartości będzie można rozróżnić rodzaj funkcji i wybrać stosowny wzór do obliczeń. Jeśli przed rdzeniem funkcji znajduje się znak dodawania lub odejmowania (pozostałe znaki działania zostały pominięte podczas rozkładu funkcji), znak ten musi zostać usunięty z łańcucha tym2. Podobnie jak w tej samej procedurze znajdującej się w bloku pochodna, do usunięcia początkowego znaku dodawania lub odejmowania zastosowano operator adresu oraz drugą zmienną łańcuchową tym1, do której wczytany został adres następnego znaku z łańcucha tym2, w wyniku czego, w łańcuchu tym1 znajduje się zawartość łańcucha tym2, bez początkowego znaku dodawania lub odejmowania. Teraz można już skopiować zawartości łańcucha tym1 do tym2. Między rdzeniem funkcji a pierwszym znakiem nawiasu otwierającego mogą znajdować się znaki cyfr, dlatego muszą one zostać przeniesione do lokalnego łańcucha ww typu Char (przedtem łańcuch został wyczyszczony z tzw. śmieci, czyli przypadkowych znaków, poprzez wpisanie do wszystkich pól łańcucha znaków pustych), a w samym łańcuchu tym2, miejsca po przeniesionych znakach zostały zastąpione znakami pustymi. Teraz, gdy łańcuch tym2 składa się wyłącznie ze znaków liter lub pojedynczego znaku pierwiastka, można go poddać identyfikacji. By wykluczyć przypadki niedozwolonej obecności cyfr za rdzeniem funkcji trygonometrycznej a przed jej pierwszym nawiasem otwierającym, w instrukcji warunkowej sprawdzany jest pierwszy znak w łańcuchu lokalnym ww, jeśli jest nim znak pusty, uruchamiany jest ciąg instrukcji warunkowych, identyfikujących funkcje trygonometryczne, cyklometryczne oraz logarytm naturalny. By uchronić się przed błędem podczas obliczeń, najpierw wartość argumentu musi być zbadana, pod kątem tego, czy mieści się w zakresie dozwolonym dla rozpoznanej funkcji – jeśli leży poza zakresem, ustawiona zostaje zmienna błędu w pozycji true, w przeciwnym przypadku wartość funkcji wyliczana zostaje przez zastosowanie jednej z funkcji bibliotecznych, należących do modułu system. Przed wyliczeniem, sprawdzany jest jeszcze rodzaj wyniku, jaki ma być zwrócony przez procedurę. Informację tę procedura otrzymuje za pośrednictwem dostarczonego parametru głównej funkcji wynik o nazwie rodzaj8 – jeśli jego wartość wynosi true, wynik funkcji trygonometrycznej lub cyklometrycznej zostanie zwrócony w stopniach, w przeciwnym przypadku – w radianach. Gdy wynik funkcji trygonometrycznej ma być zaprezentowany w stopniach, wartość kąta dodawana funkcjom bibliotecznym w parametrze, takim jak: sin, cos, tan, cotan, musi być poddana konwersji ze stopni na radiany. W przypadku funkcji cyklometrycznych, które są odwrotnością funkcji trygonometrycznych,

funkcjom bibliotecznym, takim jak: `arcsin`, `arccos` i `arctan`, podaje się w argumencie wartość kąta w stopniach, a konwersji z radianów na stopnie podlega tylko uzyskany wynik.

Nie wszystkie funkcje cyklometryczne można wyliczyć w sposób bezpośredni – przykładem jest funkcja 'arcctg', która jest odwrotnością funkcji 'arctg'. Wykorzystując tę właściwość, wartość funkcji 'arcctg' wyliczana jest za pomocą funkcji bibliotecznej `Arctan`, podając jej w argumencie odwrotność liczbową wartości argumentu, dostępnego w zmiennej `arg2`, przy dodatkowym warunku, że jest on różny od zera.

Funkcje 'tg' i 'ctg' nie istnieją dla wszystkich kątów. Niestety, funkcje biblioteczne `tan` i `cotan`, otrzymawszy wartości kątów, dla których wskazane funkcje nie istnieją, powodują wygenerowanie błędu systemu. By uchronić się przed tym błędem, wykorzystane zostały związki pomiędzy funkcjami trygonometrycznymi – funkcja 'tg' jest równa ilorazowi funkcji 'sin' i 'cos', natomiast funkcja 'ctg' – ilorazowi funkcji 'cos' i 'sin'. Ponieważ 'sin' i 'cos' istnieją dla wszystkich kątów, można bez obaw sprawdzić tę funkcję, która występuje w mianowniku, by jej wartość nie była równa zero. Aby uniezależnić się do ewentualnych błędów koprocesora, porównywana wartość funkcji 'sin' czy 'cos' została uzyskana przez przesunięcie przecinka dziesiętnego o 15 miejsc w prawo i obcięcia pozostałej części ułamkowej, uzyskując przez to liczbę całkowitą, zapamiętaną w zmiennej `kd1`.



Gdy po wykonaniu powyższego obliczenia, w zmiennej `kd1` będzie znajdowała się wartość różna od zera, można już bez obaw dokonać wyliczenia wartości funkcji 'tg'. Podobnie postępuje się w przypadku funkcji 'ctg', z tą różnicą, że do wyliczenia wartości porównawczej, zapamiętanej w zmiennej `kd1`, będzie zastosowana funkcja 'sin'.

Jeśli pierwszy znak w łańcuchu lokalnym `ww` nie jest pusty, oznacza to, że wyliczenie wartości może dotyczyć logarytmu o podanej podstawie liczbowej lub pierwiastka o stopniu większym niż dwa (wartości stopnia pierwiastka jest weryfikowana podczas jego wprowadzania). Zarówno stopień pierwiastka, jak i podstawa logarytmu, które znajdują się w łańcuchu znakowym `ww`, zostają przetworzone na postać liczbową, najpierw do postaci `string`, a następnie do liczby – w przypadku podstawy logarytmu, do liczby zmiennoprzecinkowej a stopień pierwiastka, do liczby całkowitej. W obu przypadkach, konwersja z postaci `string` do liczby została zamknięta w instrukcji `try..except..end`, z tych samych powodów, które opisane zostały w poprzednich przypadkach jej zastosowania. Jeśli wartości podstawy logarytmu i jej argumentu pozwalają na wyliczenie wartości samego logarytmu, wartość ta jest wyliczana przez zastosowaną funkcję modułu `Math` – `LogN`. W przypadku pierwiastka, koniecznym okazało się rozróżnienie parzystości jego stopnia, bowiem dla stopnia parzystego pierwiastek nie istnieje dla liczb ujemnych. Do zbadania parzystości zastosowano funkcję logiczną `odd`, która zbadawszy dostarczony jej argument liczbowy, zwraca wartość `true`, gdy jest on nieparzysty lub `false` w przeciwnym przypadku. Do wyliczenia wartości pierwiastka zastosowano funkcję `exp` należącą do modułu `system`. W jej argumencie znajduje się iloczyn odwrotności liczbowej stopnia pierwiastka i logarytmu naturalnego, którego

argumentem jest liczba pierwiastkowana. Obecność logarytmu ogranicza wyliczenie pierwiastka z liczby ujemnej, dlatego dodatkowo argument logarytmu zamknięty został w funkcji `abs`, która zwraca wartość bezwzględną argumentu. Dopiero po wyliczeniu pierwiastka można skorygować znak wyliczonej wartości na podstawie rzeczywistej postaci argumentu, dostępnego w zmiennej `arg2`.

Każde poprawne wyliczenie wartości przez zastosowane funkcje biblioteczne, jest zaznaczone przez ustawienie zmiennej logicznej `sd` w pozycji `true`. Jeśli identyfikacja funkcji dostępnej w łańcuchu lokalnym `tym2` przebiegła niepomyślnie, zmiennej logicznej błędu przypisana zostaje wartość `true`, w przeciwnym przypadku, gdy zmienna `sd` została ustawiona w pozycję `true`, weryfikowany jest uzyskany wynik. Mając na względzie fakt, że podczas ćwiczeń porównywane są ze sobą dwie wartości, ważna jest ich ochrona przed błędami koprocesora, dlatego wartości mniejsze od –1e15 są zastępowane zerami. Tak jest i w tym przypadku, dlatego gdy w zmiennej `dod2` znajduje się liczba mniejsza od –1e15, zastępowana jest zerem.

Przed wpisaniem wartości do pola `tym`, weryfikowany jest znak tej wartości na podstawie pierwszego znaku z łańcucha, z pola `t2`.

Konstrukcja początku kodu procedury oraz sposób rozpoznawania funkcji jest identyczny z procedurą o tej samej nazwie, znajdującą się w bloku `pochodna`, dlatego poniżej zaprezentowany zostanie tylko ten fragment kodu, który uwidacznia sposób wyliczania wartości po rozpoznaniu funkcji.

```
dod2:=0.0; – przypisz zmiennej wynikowej dod2 wartość zero
sd:=false; – ustaw zmienną logiczną sd w pozycji false
if ww[0]=#0 then – czy pierwszy znak w łańcuchu ww jest pusty?
begin – jeśli tak, to...
     if StrComp(tym2,'arcsin')=0 then – czy zawartość łańcucha tym2 oraz ...
                                          nazwą ‘arcsin’ są takie same?
        if abs(arg2)<=1.0 then – jeśli tak, to czy wartość bezwzględna argumentu,...
                                 dostępnego w zmiennej arg2 jest mniejsza lub równa jeden?
        begin – jeśli tak, to...
             if rodzaj8 then dod2:=Math.RadToDeg(Arcsin(arg2)) –jeśli..
                     parametr rodzaj8 jest równy true, wylicz wartość funkcji ‘arcsin’...
                     poddając konwersji uzyskany wynik z radianów na stopnie
             else dod2:=Arcsin(arg2); – jeśli parametr rodzaj8 jest równy...
                       false, wylicz bezpośrednio wartość funkcji ‘arcsin’ w radianach
             sd:=true –ustaw zmienną sd w pozycji true – wyliczenie wykonane
        end else – jeśli wartość bezwzględna argumentu jest większa od jeden, to...
            begin
                 bl:=true; – ustaw zmienną błędu w pozycji true
                 blw:=11 – przypisz zmiennej błędu kod błędu
            end;
        if not sd then – czy wyliczenie nie zostało jeszcze wykonane?
           if StrComp(tym2,'arccos')=0 then jeśli nie zostało wykonane, to ...
                czy zawartość łańcucha tym2 oraz nazwa funkcji ‘arccos’ są takie same?
             if abs(arg2)<=1.0 then – jeśli tak, to czy wartość bezwzględna...
               argumentu, dostępnego w zmiennej arg2 jest mniejsza lub równa jeden?
             begin jeśli tak, to (jak wyżej, tylko dla funkcji ‘arccos’)
                 if rodzaj8 then dod2:=Math.RadToDeg(Arccos(arg2))
                 else dod2:=Arccos(arg2);
                 sd:=true
             end else
```

```
          begin
               bl:=true;
               blw:=11
          end;
if not sd then - czy wyliczenie nie zostało jeszcze wykonane?
   if StrComp(tym2,'arctg')=0 then - jeśli nie zostało wykonane, to...
                                        (jak wyżej tylko dla funkcji 'arctg')
   begin
        if rodzaj8 then dod2:=Math.RadToDeg(Arctan(arg2))
        else dod2:=Arctan(arg2);
        sd:=true
   end;
if not sd then - czy wyliczenie nie zostało wykonane?
   if StrComp(tym2,'arcctg')=0 then - jeśli nie zostało wykonane, to ...
        czy zawartość łańcucha tym2 oraz nazwa funkcji 'arcctg' są takie same?

   begin -jeśli tak, to...
        if arg2<>0.0 then - czy argument jest różny od zera?
           if rodzaj8 then jeśli tak, to czy parametr rodzaj8...
                                   jest ustawiony w pozycji true?

              dod2:=Math.RadToDeg(Arctan(1/arg2)) - jeśli...
                        tak, to wylicz wartość funkcji 'arcctg', podając...
                        w argumencie funkcji arctan odwrotność...
                        liczbową argumentu funkcji a sam wynik poddaj...
                        konwersji z radianów na stopnie

           else dod2:=Arctan(1/arg2)- jeśli parametr rodzaj8...
                  ustawiony jest w pozycji false, wylicz wartość funkcji...
                  'arcctg', jak wyżej, tylko bez konwersji – wynik w radianach

        else if rodzaj8 then dod2:=90 - jeśli argument równy jest...
             zero, dla wyniku w stopniach wpisz do zmiennej dod2 wartość 90...

             else dod2:=pi/2; - a dla wyniku w radianach, wpisz ...
                    do zmiennej dod2 wartość π podzieloną przez dwa

        sd:=true -ustaw zmienną sd w pozycji true – wyliczenie wykonane
   end;
if not sd then - czy wyliczenie nie zostało wykonane?
   if StrComp(tym2,'sin')=0 then - jeśli nie zostało wykonane, to ...
        czy zawartość łańcucha tym2 oraz nazwa funkcji 'sin' są takie same?

   begin
        if rodzaj8 then dod2:=Sin(Math.DegToRad(arg2))
             jeśli parametr rodzaj8 ustawiony jest w pozycji true,...
             wylicz wartość funkcji podając w argumencie ...
             funkcji bibliotecznej sin wartość argumentu dostępnego...
             w zmiennej arg2 poddanego konwersji ze stopni na radiany...
             uzyskując wynik w stopniach

        else dod2:=Sin(arg2); - w przeciwnym przypadku,...
                        korzystając z funkcji bibliotecznej sin...
                        wylicz wartość tej funkcji w radianach...
                        podając jej w argumencie wartość argumentu...
                        w stopniach, czyli bez konwersji

        sd:=true -ustaw zmienną sd w pozycji true – wyliczenie wykonane
   end;
if not sd then - czy wyliczenie nie zostało wykonane?
```

```
   if StrComp(tym2,'cos')=0 then –jeśli nie zostało wykonane, to...
                                          (jak wyżej, tylko dla funkcji 'cos')
   begin
        if rodzaj8 then dod2:=Cos(Math.DegToRad(arg2))
        else dod2:=Cos(arg2);
        sd:=true
   end;
if not sd then – czy wyliczenie nie zostało wykonane?
   if StrComp(tym2,'ctg')=0 then –jeśli nie zostało wykonane, to...
                                          (jak wyżej, tylko dla funkcji 'ctg')
   begin
        if rodzaj8 then – czy parametr rodzaj8 ma wartość true?
        begin – jeśli tak, to...
             kd1:=Trunc(Sin(Math.DegToRad(arg2))*1E15);
             wylicz wartość funkcji 'sin', wykorzystując...
             funkcję biblioteczną sin, podając jej w argumencie...
             wartość poddaną konwersji ze stopni na radiany...
             a sam wynik przetwórz, przesuwając 15 cyfr...
             po przecinku do części całkowitej. Uzyskany wynik...
             oddziel od części ułamkowej uzyskując liczbę całkowitą

             if kd1=0 then – czy uzyskana liczba w zmiennej kd1...
                                  jest równa zero?

             begin – jeśli tak, to...
                  bl:=true; – ustaw zmienną błędu w pozycji true
                  blw:=11 – przypisz zmiennej kodu błędu stosowny kod
             end else dod2:=Cotan(Math.DegToRad(arg2))
          jeśli liczba w zmiennej kd1 nie jest zerem, to wylicz wartość funkcji...
          podając w argumencie funkcji bibliotecznej cotan liczbę ze zmiennej...
           arg2 poddaną konwersji ze stopni na radiany,...
           uzyskując wynik w stopniach

        end else dod2:=Cotan(arg2); – jeśli parametr rodzaj8...
            ustawiony jest w pozycji false, wylicz wartość funkcji, podając...
           w argumencie funkcji bibliotecznej cotan liczbę ze zmiennej arg2...
           bez konwersji, uzyskując wynik w radianach

        sd:=true –ustaw zmienną sd w pozycji true – wyliczenie wykonane
   end;
if not sd then – czy wyliczenie nie zostało wykonane?
   if StrComp(tym2,'tg')=0 then –jeśli nie zostało wykonane, to...
                                          (jak wyżej, tylko dla funkcji 'tg')
   begin
        if rodzaj8 then
        begin
             kd1:=Trunc(Cos(Math.DegToRad(arg2))*1E15);
             if kd1=0 then
             begin
                  bl:=true;
                  blw:=11
             end else dod2:=Tan(Math.DegToRad(arg2))
        end else dod2:=Tan(arg2);
        sd:=true
   end;
if not sd then – czy wyliczenie nie zostało wykonane?
   if StrComp(tym2,'ln')=0 then – jeśli zawartość łańcucha tym2...
                                     i nazwa funkcji 'ln' są takie same, to...
      if arg2>0.0 then – czy wartość w zmiennej arg2 jest większa…
                                od zera?
      begin – jeśli tak, to...
```

```
                    dod2:=ln(arg2); - wylicz wartość logarytmu naturalnego
                    sd:=true -ustaw zmienną sd w pozycji true – wyliczenie...
                                zostało wykonane
                end else - gdy wartość w zmiennej arg2 nie jest większa od zera, to...
                    begin
                         bl:=true; - ustaw zmienną błędu w pozycji true
                         blw:=11 - zmiennej kodu błędu przypisz stosowny kod błędu
                    end;
            if not sd then - czy wyliczenie nie zostało wykonane?
               if StrComp(tym2,'log')=0 then -jeśli nie zostało wykonane, to...
                                (jak wyżej, tylko dla logarytmu dziesiętnego)
                  if arg2>0.0 then
                  begin
                       dod2:=Math.Log10(arg2);
                       sd:=true
                  end else
                      begin
                           bl:=true;
                           blw:=11
                      end;
            if not sd then - czy wyliczenie nie zostało wykonane?
               if StrComp(tym2,'\')=0 then -jeśli nie zostało wykonane, to...
                                (jak wyżej, tylko dla pierwiastka drugiego stopnia)
                  if arg2>=0.0 then
                  begin
                       dod2:=Sqrt(arg2);
                       sd:=true
                  end else
                      begin
                           bl:=true;
                           blw:=11
                      end;
end else - jeśli pierwszy znak w łańcuchu ww nie jest pusty, to...
    begin
         if StrComp(tym2,'log')=0 then - jeśli nie zostało wykonane, to...
                        czy zawartość łańcucha tym2 i nazwa 'log' są takie same?
            if arg2>0 then - czy argument w zmiennej arg2 jest większy od zera?
            begin - jeśli tak, to...
                 lanc1:=StrPas(ww); - dokonaj konwersji łańcucha ww typu ...
                                      PChar do łańcucha lanc1 typu string
                 dod1:=0.0; - zainicjuj zmienną dod1 wartością zero
                 try -gdy spodziewamy się błędu konwersji
                    dod1:=StrToFloat(lanc1); - dokonaj konwersji...
                                zawartości łańcucha typu string...
                                do liczby zmiennoprzecinkowej dod1
                 except - jeśli wystąpi błąd konwersji, to...
                       bl:=true; - ustaw zmienną błędu w pozycji true
                       blw:=6 - przypisz zmiennej kod błędu
                 end;
                 if not bl then - jeśli po konwersji na liczbę nie było błędu, to...
                    if (dod1>0.0) and (dod1<>1.0) then - czy ...
                      uzyskana liczba (podstawa logarytmu) jest większa od zera...
                      i różna od jeden?
                    begin - jeśli tak, to...
                         dod2:=Math.LogN(dod1,arg2); wylicz...
                            wartość logarytmu o dowolnej podstawie podając...
                            w argumentach funkcji bibliotecznej modułu Math...
```

```
                        – logN wartość podstawy i argumentu logarytmu
                    sd:=true –ustaw zmienną sd w pozycji true
                                – wyliczenie zostało wykonane
                end else jeśli wartość podstawy logarytmu nie jest większa...
                          od zera lub jest równa jeden, to...
                    begin
                         bl:=true; – ustaw zmienną błędu
                         blw:=4 – przypisz zmiennej błędu kod błędu...
                                   informującego o niewłaściwej wartości...
                                   podstawy logarytmu
                    end
    end else – jeśli wartość argumentu logarytmu nie jest większa od zera, to...
        begin
             bl:=true; – ustaw zmienną błędu w pozycji true
             blw:=11 przypisz zmiennej kod błędu informującego...
                       o niemożliwości wykonania obliczenia
        end;
if not sd then – czy wyliczenie nie zostało jeszcze wykonane?
    if StrComp(tym2,'\')=0 then – czy zawartość łańcucha tym2...
                                  i znak pierwiastka są sobie równe?
    begin – jeśli tak, to...
         kd1:=0; – zainicjuj zmienną całkowitoliczbową wartością zero
         lanc1:=StrPas(ww); – dokonaj konwersji łańcucha ww typu ...
                              PChar do łańcucha lanc1 typu string
         try – gdy spodziewamy się błędu konwersji
            kd1:=StrToInt(lanc1); – dokonaj konwersji zawartości...
                        łańcucha typu string do liczby całkowitej...
                        i zapamiętaj ją w zmiennej kd1
         except – jeśli wystąpi błąd konwersji, to...
              bl:=true; – ustaw zmienną błędu w pozycji true
              blw:=10 – przypisz zmiennej błędu kod błędu informujący...
                        o błędnym zapisie
         end;
         if not bl then – jeśli konwersja przebiegła bez błędu, to...
            if Odd(kd1) then – jeśli wartość w zmiennej kd1 (stopień...
                               pierwiastka) jest liczbą nieparzystą, to...
              if arg2<>0.0 then – czy argument pierwiastka jest ...
                                  różny od zera?
              begin – jeśli tak, to...
                   dod2:=Exp(1/kd1*Ln(Abs(arg2)));
wylicz wartość pierwiastka, wykorzystując funkcję biblioteczną exp,...
podając jej w argumencie iloczyn odwrotności stopnia pierwiastka...
i logarytmu, podając mu w argumencie wartość bezwzględną...
argumentu pierwiastka. Uzyskany wynik jest dodatni
                   if arg2<0.0 then dod2:=- dod2; – jeśli ...
                                   argument pierwiastka jest ujemny,...
                                   zmień uzyskany wynik na ujemny
                   sd:=true – wyliczenie zostało wykonane
              end else sd:=true – jeśli argument pierwiastka...
              równy jest zero, zaznacz dokonane wyliczenie, ponieważ...
              wartość pierwiastka z zerowego argumentu wynosi zero,...
              (wartość ta występuje już w zmiennej dod2)
         else – gdy stopień pierwiastka jest liczbą parzystą, to...
         if arg2<0 then – czy argument pierwiastka jest ujemny?
```

```
                    begin - jeśli tak, to...
                          bl:=true; - ustaw zmienną błędu w pozycji true
                          blw:=11 - przypisz zmiennej kodu błędu liczbę 11
                    end else - jeśli argument nie jest ujemny, to...
                             if arg2=0 then sd:=true -jeśli argument...
                                                      ma wartość zero, to...(jak wyżej)
                             else - gdy argument jest większy od zera, to...
                             begin
                                  dod2:=Exp(1/kd1*Ln(arg2));
          wylicz pierwiastek parzystego stopnia, wykorzystując funkcję biblioteczną exp...
          podając jej w argumencie iloczyn odwrotności liczbowej stopnia pierwiastka...
          i  logarytmu z argumentu pierwiastka
                                  sd:=true - wyliczenie zostało wykonane
                             end
               end;
     end;
if sd then - czy po opuszczeniu bloku obliczeń, wyliczenie zostało wykonane?
begin - jeśli tak, to...
     if Trunc(dod2)=0 then - czy uzyskana wartość jest wyłącznie liczbą ułamkową?
        if Trunc(dod2*1E15)=0 then dod2:=0.0; jeśli tak, to jeśli jest ona...
            mniejsza od liczby –1E15, do zmiennej wynikowej dod2 wpisz wartość zero
     if lawag2^.t2[0]='-' then lawag2^.tym:=-dod2 - jeśli pierwszym...
                          znakiem w łańcuchu t2 jest odejmowanie, wpisz do pola...
                          tym bieżącego elementu wagony wartość ze zmiennej...
                          dod2 ze znakiem minus
     else lawag2^.tym:=dod2 - w przeciwnym przypadku, wpisz do pola tym...
                            niezmienioną wartość ze zmiennej dod2
end else - jeśli wyliczenie nie zostało wykonane, to...
     if not bl then - czy zmienna błędu nie została już ustawiona w pozycji true?
     begin - jeśli nie, to...
          bl:=true; - ustaw zmienną błędu w pozycji true
          blw:=7 - przypisz zmiennej blw kod liczbowy błędu informujący o...
                   nierozpoznaniu funkcji
     end;
```

Po uwolnieniu pamięci zajmowanej przez łańcuch lokalny `tym2` oraz łańcuch z pola `t2` należącego do bieżącego elementu `wagony`, procedura kończy swoją pracę.

XI. 3. Wykonanie obliczeń na dwóch liczbach – procedura `licz_licz`

Zadaniem procedury jest wykonanie działania na dwóch liczbach rzeczywistych, które zawarte są w odpowiednich polach liczbowych należących do elementu `wagony`. Adres owego elementu został procedurze dostarczony za pośrednictwem parametru. Znak działania w postaci liczbowej znajduje się na zerowej pozycji w tablicy `dzial`, będącej również polem elementu `wagony`. Gdy wartość znaku działania jest zerowa, procedura ma za zadanie tylko przepisanie wartości z pola `tym` do pola `ost` a po przepisaniu, zainicjowanie pola `tym` wartością zero. Jeśli wartość znaku działania nie jest zerowa, do lokalnych zmiennych liczbowych `p1` i `p2` przepisane zostają wartości z odpowiednich pól elementu `wagony`, czyli:

– gdy znakiem działania jest dodawanie lub odejmowanie, odpowiednie wartości znajdują się w polach `tym` i `ost`. Za pomocą dodatkowej zmiennej logicznej `kier3`, zaznaczone zostało miejsce docelowe uzyskanego wyniku, które w tym przypadku wskazuje na pole `ost` bieżącego elementu `wagony`;
– gdy znakiem działania jest mnożenie, dzielenie lub potęgowanie, pierwsza wartość znajduje się w polu `tym`, natomiast druga jest zależna od ustawienia pola logicznego `p_fw` – jeśli ustawione zostało w pozycji `true`, druga wartość do wyliczenia znajduje się w polu `p_tym`, w przeciwnym przypadku, w polu `p_ost`. Zmiennej logicznej `kier3` przypisana zostaje wartość `true`, która wskazuje, że uzyskany wynik znajdzie się w polu `tym`.

Pola, których nazwy zaczynają się od litery ‘p’ i znaku podkreślenia zostały uzupełnione wartościami pochodzącymi od elementu `wagony`, który został już usunięty z pamięci. Dzięki wartości logicznej znajdującej się w polu `p_fw` bieżącego elementu `wagony`, można wskazać właściwe pole zawierające wartość liczbową przeznaczoną do obliczeń.

```
c1:=wag3^.dzial[0]; - wpisz do zmiennej c1 wartość znaku działania, która będzie...
                      wykorzystana do bieżących obliczeń

if c1=0 then - czy wartość znaku działania jest zerowa?
begin - jeśli tak, to…
     wag3^.ost:=wag3^.tym; - przenieś wynik obliczeń z części tymczasowej...
                             do ostatecznej

     wag3^.tym:=0.0 - usuń wartość z pola tym
end else - jeśli wartość znaku działania jest niezerowa, to...
    begin
        if (c1=43) or (c1=45) then - czy wartość znaku działania...
                                     odpowiada dodawaniu lub odejmowaniu?
        begin - jeśli tak, to...
             kier3:=false; - ustaw zmienną kier3 w pozycji false...
                             (wynik obliczeń ma znaleźć się w polu ost)

             p1:=wag3^.ost; - wpisz do zmiennej p1 pierwszą wartość do obliczeń
             p2:=wag3^.tym - wpisz do zmiennej p2 drugą wartość do obliczeń
        end else - jeśli nie, znakiem działania jest mnożenie lub dzielenie, lub potęgowanie
            begin
                 kier3:=true; - ustaw zmienną kier3 w pozycji true…
                                (wynik obliczeń ma znaleźć się w polu tym)

                 if wag3^.p_fw then p2:=wag3^.p_tym - jeśli pole p_fw…
                                                      zostało ustawione w pozycji true, drugą...
                                                      wartość do obliczeń pobierz z pola p_tym

                    else p2:=wag3^.p_ost; - w przeciwnym przypadku…
                                            drugą wartość do obliczeń pobierz z pola p_ost

                 p1:=wag3^.tym; - pierwszą wartość do obliczeń pobierz z pola tym
            end;
        wag3^.dzial[0]:=wag3^.dzial[1]; } przesuń następną wartość znaku...
                                         } działania z pozycji pierwszej...
        wag3^.dzial[1]:=0;               } na zerową w tablicy dzial...
        . . .
```

Skoro wartości do wyliczenia zostały już ustalone, wykorzystany znak działania do bieżących obliczeń zostaje zastąpiony znakiem znajdującym się na następnej pozycji w tablicy `dzial`. Pozwala to na wykorzystanie kolejnego znaku działania przy następnym wywołaniu tej procedury.

W procedurze ustalona została maksymalna wartość, której uzyskany wynik nie może przekroczyć. Ma to na celu z jednej strony, zapobieżenie przekroczenia dopuszczalnej wartości dla typu `double`, a z drugiej, ograniczenie wpływu zaokrągleń dokonywanych przez koprocesor numeryczny wobec skrajnie małych i skrajnie dużych wartości. W procedurze zadeklarowana została stała o nazwie `maks8`, której wpisano liczbę 1e50. Jest to wystarczająco duża liczba, która z powodzeniem wystarczy do zrealizowania zadania, jakie powierzone zostało procedurze podczas obliczeń. Przed rozpoczęciem pracy części programu dokonującego obliczeń, wartości dwóch zmiennych `p1` i `p2` porównywane są z wartością maksymalną, czyli wspomnianą stałą – przekroczenie choć jednej z nich powoduje opuszczenie procedury z błędem. Jeśli wartości nie zostały przekroczone, identyfikowany jest rodzaj działania na podstawie wczytanego do zmiennej `c1` wartości znaku działania. Rozpoznawanie działania odbywa się w instrukcji wyboru `case` – każde dopasowanie tzw. selektora, którym jest zmienna `c1`, z jedną ze stałych wyboru, uruchamia instrukcje odpowiednie do rozpoznanego znaku działania. Nim działanie zostanie wykonane, program musi sprawdzić, czy nie nastąpi przekroczenie dopuszczalnej wartości, zapisanej w stałej `maks8`.

Ponieważ odwrotnością mnożenia jest dzielenie, mnożenie zostanie wykonane wtedy, gdy wartość bezwzględna ilorazu liczby maksymalnej i pierwszej wartości do wyliczenia będzie mniejsza lub równa wartości bezwzględnej drugiej liczby.

```
42: - czy działaniem do wykonania jest mnożenie?
begin - jeśli tak, to...
     if p1<>0.0 then - czy wartość w zmiennej p1 jest różna od zera?
        if Abs(maks8/p1)<=Abs(p2) then - jeśli tak, to czy nastąpi przekroczenie...
                                                wartości maksymalnej?
        begin - jeśli tak, to...
              bl:=true; - ustaw zmienną błędu w pozycji true
              blw:=11; - przypisz zmiennej kodu błędu wartość liczbową błędu
              exit - opuść procedurę
        end;
        p1:=p1*p2 - jeśli nie nastąpi przekroczenia maksymalnej wartości...
end;                wykonaj mnożenie
```

Odwrotnością dodawania jest odejmowanie, dlatego różnica stałej wartości maksymalnej i pierwszej liczby jest porównywana z drugą liczbą. Kontrola przekroczenia dopuszczalnej wartości wykonywana jest oddzielnie dla wartości dodatniej i ujemnej jednej z liczb. Rozwiązanie dla odejmowania jest porównywalne z dodawaniem – różnica polega tylko na odwrotnym warunku porównującym różnicę z drugą liczbą.

```
43: - czy działaniem do wykonania jest dodawanie?
begin - jeśli tak, to...
     if p1>=0.0 then - czy pierwsza liczba jest większa lub równa zero?
        if p2>(maks8-p1) then - jeśli tak, to czy druga liczba jest większa od ...
                                        różnicy wartości maksymalnej i pierwszej liczby?
        begin - jeśli tak, to...
              bl:=true; - ustaw zmienną błędu w pozycji true
              blw:=11; - przypisz zmiennej kodu błędu wartość liczbową błędu
              exit - opuść procedurę
        end else
     else if p2<(-maks8-p1) then - gdy pierwsza liczba jest mniejsza od zera to...
                                   czy druga liczba jest mniejsza od różnicy...
                                   wartości minimalnej i pierwszej liczby?
```

```
            begin - jeśli tak, to...
                  bl:=true;
                  blw:=11;    } (jak wyżej)
                  exit
            end;
      p1:=p1+p2 - jeśli nie nastąpi przekroczenia maksymalnej wartości...
end;              wykonaj dodawanie

45: - czy działaniem do wykonania jest odejmowanie?
begin - jeśli tak, to...
      if p1>=0.0 then - czy pierwsza liczba jest większa lub równa zero?
         if p2<-(maks8-p1) then - jeśli tak, to czy druga liczba jest mniejsza od...
                                  ujemnej różnicy maksymalnej wartości...
                                  i pierwszej liczby?
         begin - jeśli tak, to...
               bl:=true;
               blw:=11;    } (jak wyżej)
               exit
         end else
      else if p2>- (- maks8- p1) then - gdy pierwsza liczba jest mniejsza od zera,
                                        to, czy druga liczba jest większa od...
                                        ujemnej różnicy minimalnej wartości...
                                        maksymalnej i pierwszej liczby?
            begin - jeśli tak, to...
                  bl:=true;
                  blw:=11;    } (jak wyżej)
                  exit
            end;
      p1:=p1-p2 - jeśli nie nastąpi przekroczenia maksymalnej wartości...
end;              wykonaj odejmowanie
```

Gdy wartość bezwzględna drugiej liczby (dzielnika), jest większa lub równa wartości bezwzględnej ilorazu pierwszej liczby (dzielnej) i wartości maksymalnej, możliwe jest wykonanie dzielenia na dwóch liczbach `p1` i `p2`, zabezpieczając wynik dzielenia przed przekroczeniem wartości maksymalnej. Dodatkowym warunkiem jest zabezpieczenie przed dzieleniem przez zero.

```
47: - czy działaniem do wykonania jest dzielenie?
begin - jeśli tak, to...
      if Abs(p2)<Abs(p1/maks8) then - czy wynik dzielenia przekroczy...
                                      wartość maksymalną?
      begin - jeśli tak, to...
            bl:=true; - ustaw zmienną błędu w pozycji true
            blw:=11; - przypisz zmiennej kodu błędu wartość liczbową błędu
            exit - opuść procedurę
      end;
      if p2<>0.0 then p1:=p1/p2 - gdy dzielnik jest różny od zera, wykonaj dzielenie
      else w przeciwnym przypadku...
      begin
            bl:=true; - ustaw zmienną błędu w pozycji true
            blw:=9 - przypisz zmiennej kodu błędu liczbę informującą o dzieleniu przez zero
      end;
end;
```

Potęgowanie jest działaniem dość restrykcyjnym, dlatego należy mu poświęcić nieco więcej uwagi. W celu sprawdzenia, czy w wyniku realizacji działania, nie dojdzie do przekroczenia dopuszczalnej wartości, trzeba najpierw wyliczyć iloraz logarytmu naturalnego z licz-

by maksymalnej i wartości bezwzględnej logarytmu naturalnego z liczby potęgowanej, zawartej w zmiennej `p1`. Z uwagi na to, że w powyższym wyliczeniu występuje zarówno logarytm, jak i dzielenie, pożądane jest, by wartość znajdująca się w zmiennej `p1` była różna od zera i od wartości bezwzględnej równej jeden. Jeśli uzyskany wynik będzie większy lub równy liczbie potęgującej, dostępnej w zmiennej `p2`, oznacza to, że potęgowanie może zostać wykonane.

Może się zdarzyć, że liczba potęgująca jest ułamkiem dziesiętnym. W tej sytuacji trzeba go dokładnie zbadać, by mieć pewność, że wynik potęgowania będzie zgodny z zasadami matematyki. Rozważmy następujący przypadek:

$$-2^{\frac{1}{2}} = -2^{0,5}$$

Czy powyższe działanie można wykonać? Jednoznacznej odpowiedzi udzieli funkcja logiczna `czy_nieparzysta`, która otrzymawszy liczbę dziesiętną – w tym przypadku 0,5 – musi przetworzyć ją na ułamek zwykły i sprawdzić, czy uzyskany mianownik jest liczbą parzystą. Otrzymana logiczna odpowiedź sugeruje, że działania nie można wykonać, ponieważ nie istnieje pierwiastek parzystego stopnia z liczby ujemnej.

$$-2^{\frac{1}{2}} = \sqrt{-2}$$

Gdy liczba potęgująca jest całkowitoliczbowa, do określenia jej parzystości wykorzystana została funkcja biblioteczna `odd`.

Niezależnie od tego, czy liczba potęgująca jest całkowitoliczbowa czy ułamkiem dziesiętnym, potęgowanie wykonywane jest przez zastosowanie funkcji bibliotecznej modułu `system` – `exp`, podając jej w argumencie iloczyn liczby potęgującej (`p2`) i logarytmu naturalnego z liczby potęgowanej (`p1`).

```
94: - czy działaniem do wykonania jest potęgowanie?
begin - jeśli tak, to...
     if (p1<>0.0) and (Abs(p1)<>1.0) then - czy wartość potęgowana jest ...
                                    różna od zera i wartości bezwzględnej liczby jeden?
        pm:=ln(maks8)/Abs(ln(Abs(p1))) - jeśli tak, wylicz wartość do porównania
     else pm:=p2; - w przeciwnym przypadku przypisz zmiennej pm liczbę potęgującą
     if p2>pm then - czy liczba potęgująca jest większa od wartości porównawczej?
     begin - jeśli tak, to...
          bl:=true; - ustaw zmienną błędu w pozycji true
          blw:=11; - zmiennej kodu błędu przypisz odpowiedni kod błędu
          exit - opuść procedurę
     end;
     if p1<>0.0 then - czy liczba potęgowana jest różna od zera?
     begin - jeśli tak, to...
          pm:=Frac(p2); - do zmiennej pm wpisz część ułamkową liczby potęgującej
          if pm<>0.0 then - czy liczba potęgująca jest ułamkowa?
             if czy_nieparzysta(pm) then - jeśli tak, to czy jest nieparzysta?
                if p1>0.0 then p1:=Exp(p2*Ln(p1)) - jeśli liczba…
                                    potęgowana jest większa od zera, to wykonaj potęgowanie
                else p1:=-Exp(p2*Ln(Abs(p1))) - jeśli liczba potęgowana jest...
```

```
                                        mniejsza od zera, wykonaj potęgowanie ze zmianą znaku
            else if p1>0.0 then p1:=Exp(p2*Ln(p1)) -jeśli liczba ...
                                        potęgująca jest parzysta a liczba potęgowana...
                                        jest większa od zera, to wykonaj działanie
                 else - jeśli liczba potęgująca jest mniejsza od zera, to...
                 begin
                      bl:=true;
                      blw:=11;      } ustaw zmienną błędu w pozycji true...
                      exit          } i opuść procedurę
                 end
         else - jeśli liczba potęgująca jest całkowitoliczbowa, to...
         if Odd(Trunc(p2)) then - czy jest ona nieparzysta?
            if p1>0.0 then p1:=Exp(p2*Ln(p1)) - jeśli liczba potęgowana...
                                        jest większa od zera, wykonaj działanie
            else p1:=-Exp(p2*Ln(Abs(p1))) - w przeciwnym przypadku...
                                        wykonaj działanie ze zmianą znaku
         else p1:=Exp(p2*Ln(Abs(p1))) - jeśli liczba potęgująca jest parzysta...
                                        wykonaj działanie, podając w argumencie...
                                        logarytmu wartość bezwzględną...
                                        liczby potęgowanej
      end;
end;
```

Wyliczona wartość ze zmiennej `p1` trafia do pola `tym`, gdy zmienna logiczna `kier3` ustawiona została w pozycji `true` lub do pola `ost`, w przeciwnym przypadku.

```
if kier3 then wag3^.tym:=p1 else wag3^.ost:=p1 - umieść wynik obliczeń w...
polu tym, gdy zmienna kier3 ustawiona jest w pozycji true lub ost, w przeciwnym przypadku
```

XI. 3.1. Czy liczba dziesiętna jest nieparzysta? Funkcja `czy_nieparzysta`

Sprawdzenie parzystości liczby dziesiętnej, wymaga zamiany ułamka dziesiętnego na ułamek zwykły, po to, by uzyskać mianownik takiego ułamka i sprawdzić jego nieparzystość. W pętli `repeat- until`, w zmiennej `bx1` wyliczany jest iloraz zmiennej sterującej pętlą i dostarczonego ułamka dziesiętnego. Jeśli wynik dzielenia okaże się liczbą całkowitą, oznacza to, że zmienna sterująca `ax2` jest poszukiwanym mianownikiem. W kolejnej instrukcji warunkowej sprawdzana jest parzystość wartości tej zmiennej i jeśli jest ona parzysta, funkcja zwraca wartość `false` oznaczającą, że dostarczona liczba dziesiętna jest parzysta – w każdym innym przypadku liczba jest nieparzysta. Liczba iteracji pętli jest zależna od znalezienia ułamka zwykłego, którego licznik, dostępny w zmiennej `ax2`, podzielny jest bez reszty przez mianownik wyliczony w zmiennej `bx1`.

```
function czy_nieparzysta(const ulamek:double):boolean;
var ax2,ax3:integer;
    bx1:double;
begin
     Result:=true; - wartość domyślna zwracana przez funkcję – liczba nieparzysta
     ax2:=1; - zainicjuj zmienną ax2 początkową wartością licznika testowego
     repeat
           bx1:=ax2/ulamek; - wykonaj dzielenie zmiennej sterującej ax2...
                                i dostarczonego ułamka dziesiętnego
```

```
            if frac(bx1)=0.0 then - czy uzyskany iloraz jest liczbą całkowitą?
            begin - jeśli tak, to...
                  ax3:=trunc(bx1); - do zmiennej całkowitoliczbowej…
                        przepisz część całkowitą z liczby zmiennoprzecinkowej bx1
                  if (ax3 mod 2)=0.0 then Result:=false; - jeśli ...
                        liczba całkowita jest parzysta, przypisz zmiennej Result...
                        zwracaną wartość false – liczba parzysta
                  break - dalsze sprawdzanie jest już niepotrzebne, więc opuść pętlę
            end;
            inc(ax2) - zwiększ wartość licznika ułamka o jeden
      until ax2>99; - opuść pętlę, gdy licznik osiągnie wartość 100
end;
```

XII. Moduł komunikatów – Unit9

Moduł ten wraz z formą wyposażoną w przyciski i elementy informacyjne są elementami komunikacji programu z użytkownikiem. Forma wyświetlana jest wtedy, gdy wymagana jest reakcja użytkownika, która ma wpłynąć na dalsze działanie programu. Moduł składa się z formy opatrzonej dwiema tzw. etykietami, czyli elementami Label1 i Label2 z klasy Tlabel, opatrzonymi odpowiednim tekstem zależnym od zaistniałej sytuacji w programie. Informacja liczbowa o zaistniałej sytuacji przekazywana jest procedurze Wynik9 poprzez parametr o nazwie tryb9 (procedura ta jest elementem klasy zadeklarowanej w tym module). Na podstawie wartości liczbowej parametru, procedura wybiera odpowiedni tekst komunikatu oraz udostępnia użytkownikowi jeden lub dwa przyciski Button1 i Button2 z klasy TButton jako elementy wyboru dla użytkownika. Dołączony jest także odpowiednio dobrany obrazek w postaci tzw. buźki, która ma uatrakcyjnić wizerunek samego okna komunikatu. Odpowiedni obrazek wstawiany jest za pośrednictwem komponentu Image1 z klasy TImage. Komponent ten posiada ustawioną właściwość stretch w pozycji false oraz Autosize w pozycji true, co sprawia, że obrazki będą wstawiane w niezmienionym rozmiarze.

W sytuacjach, gdy potrzebny jest tylko jeden przycisk, drugi zostaje ukryty, poprzez wykorzystanie właściwości Visible danego przycisku, przypisując mu wartość false gdy przycisk ma być ukryty lub true w przeciwnym przypadku.

```
procedure Wynik9(tryb9:byte;aw9:byte=0;bw9:byte=0);
var iw9:string[14];

    function Bledy(const kod_bl:byte):string; - funkcja wewnętrzna procedury

begin
     iw9:=''; - przypisz zmiennej łańcuchowej iw9 pusty łańcuch znakowy
     case tryb9 of
          1..13: - przypadki błędów
          begin
               Form9.Caption:='Błąd'; - umieść tekst na pasku tytułowym formy
               Button1.Caption:='Rozumiem'; - umieść tekst...
                                              na pierwszym przycisku
               Button2.Visible:=false; - ukryj drugi przycisk
               Label2.Font.Color:=clBlack; - ustaw kolor czcionki dolnej...
                                             etykiety Label2 na czarny
               Label2.Caption:=Bledy(tryb9); - wstaw na dolnej etykiecie...
                                  Label2 tekst zwrócony przez funkcję Bledy
               Label1.Caption:=''; - górny etykieta Label1 bez tekstu
               iw9:='.\Wynik\H.bmp'; - przypisz zmiennej iw9 tekst zawierający...
                                       ścieżkę do pliku graficznego „buźki”
          end;
          14: - podsumowanie – po zakończeniu ćwiczeń
          begin
               Form9.Caption:='Podsumowanie';
               if aw9+bw9>=5 then - czy liczba wszystkich odpowiedzi...
                                    jest większa lub równa 5?
                  if aw9>bw9 then -jeśli tak, to czy liczba poprawnych odpowiedzi...
                                   jest większa od błędnych?
```

```
        if bw9=0 then iw9:='.\Wynik\C.bmp' - jeśli tak, to...
                      gdy wszystkie odpowiedzi są poprawne, zmiennej łańcuchowej ...
                      iw9 przypisz ścieżkę do „buźki” C
        else iw9:='.\Wynik\B.bmp' - gdy ilość poprawnych ...
                                    odpowiedzi jest większa od błędnych,...
                                    to zmiennej łańcuchowej iw9 przypisz...
                                    ścieżkę do pliku „buźki” B
    else if aw9=bw9 then iw9:='.\Wynik\A.bmp' - gdy ilość...
                                    błędnych i poprawnych odpowiedzi jest równa,...
                                    to zmiennej łańcuchowej iw9 przypisz ścieżkę...
                                    do pliku „buźki” A
        else if aw9=0 then iw9:='.\Wynik\F.bmp' - gdy...
                                    brak jest poprawnych odpowiedzi, to zmiennej...
                                    iw9 przypisz ścieżkę do „buźki” F
            else iw9:='.\Wynik\E.bmp' - gdy liczba…
                                    błędnych odpowiedzi jest większa…
                                    niż poprawnych, to ...
                                    zmiennej łańcuchowej iw9 przypisz ścieżkę...
                                    do pliku „buźki” E
    else iw9:='.\Wynik\D.bmp'; - jeśli liczba wszystkich odpowiedzi..
                                    jest mniejsza niż 5, to zmiennej łańcuchowej iw9...
                                    przypisz ścieżkę do pliku „buźki” D
    Label1.Font.Color:=clGreen;  - w górnej etykiecie ustaw...
                                    kolor czcionki na zielony
    Label1.Caption:=
    'Ilość poprawnych odpowiedzi = '+IntToStr(aw9);
    ułóż tekst podsumowania poprawnych odpowiedzi dla górnej etykiety,...
    składający się z tekstu statycznego i liczby poprawnych odpowiedzi...
    uzyskanej w wyniku konwersji z liczby na łańcuch typu string....
    Znak ‘+’ jest operatorem łączenia łańcuchów
    Label2.Font.Color:=clRed; - w dolnej etykiecie ustaw...
                                    kolor czcionki na czerwony
    Label2.Caption:=
    'Ilość błędnych odpowiedzi = '+IntToStr(bw9);
    ułóż tekst podsumowania błędnych odpowiedzi dla dolnej etykiety,...
    w taki sam sposób, jak dla górnej etykiety
    Button1.Caption:='Zamknij'; - umieść tekst na pierwszym
                                    przycisku
    Button2.Visible:=false; - ukryj drugi przycisk
end;
15: - poprawna odpowiedź i propozycja nowej funkcji
begin
    Form9.Caption:='Wynik'; - umieść tekst na pasku tytułowym formy
    Button1.Caption:='Tak'; - umieść tekst na pierwszym przycisku
    Button2.Visible:=true; - uwidocznij drugi przycisk
    Button2.Caption:='Nie'; - umieść tekst na drugim przycisku
    Label1.Font.Color:=clGreen; - ustaw kolor czcionki górnej...
                                    etykiety Label1 na zielony
    Label1.Caption:='Wpisana pochodna jest poprawna';
    wpisz tekst do górnej etykiety Label1
    Label2.Font.Color:=clBlack; - ustaw kolor czcionki dolnej...
                                    etykiety Label2 na czarny
    Label2.Caption:='Chcesz nową funkcję?';
```

```
      wpisz tekst do dolnej etykiety Label2
      iw9:='.\Wynik\A.bmp'; - zmiennej iw9 wpisz ścieżkę...
                                  do pliku „buźki" A
end;
16: - poprawna odpowiedź – informacja z jednym przyciskiem Button1
begin
      Form9.Caption:='Wynik';
      Button1.Caption:='Rozumiem';
      Button2.Visible:=false; - ukryj drugi przycisk
      Label2.Font.Color:=clGreen;
      Label2.Caption:='Wpisana pochodna jest poprawna';
      Label1.Caption:=''; - przypisz górnej etykiecie łańcuch pusty
      iw9:='.\Wynik\B.bmp';
end;
17: - błędna odpowiedź i propozycja nowej funkcji
begin
      Form9.Caption:='Wynik';
      Button1.Caption:='Tak';
      Button2.Visible:=true;
      Button2.Caption:='Nie';
      Label1.Font.Color:=clRed;
      Label1.Caption:='Wpisana pochodna jest błędna';
      Label2.Font.Color:=clBlack;
      Label2.Caption:='Chcesz nową funkcję?';
      iw9:='.\Wynik\D.bmp';
end;
18: - błędna odpowiedź z możliwością poprawienia błędu bez pomocy programu
begin
      Form9.Caption:='Wynik';
      Button1.Caption:='Tak';
      Button2.Visible:=true;
      Button2.Caption:='Nie';
      Label1.Font.Color:=clRed;
      Label1.Caption:='Wpisana pochodna jest błędna';
      Label2.Font.Color:=clBlack;
      Label2.Caption:='Dasz radę ją poprawić?';
      iw9:='.\Wynik\F.bmp';
end;
19: - finał przykładu – brak odpowiedzi
begin
      Form9.Caption:='Finał przykładu';
      Button1.Caption:='Tak';
      Button2.Visible:=true;
      Button2.Caption:='Nie';
      Label1.Font.Color:=clRed;
      Label1.Caption:='Brak wpisanej pochodnej';
      Label2.Font.Color:=clBlack;
      Label2.Caption:='Chcesz nową funkcję?';
      iw9:='.\Wynik\A.bmp';
end;
20: - brak pliku tekstowego zawierającego przykłady funkcji do ćwiczeń
begin
      Form9.Caption:='Błąd';
      Button1.Caption:='Rozumiem';
      Button2.Visible:=false;
      Label1.Font.Color:=clRed;
      Label1.Caption:=
           'Plik z przykładami funkcji nie istnieje';
      Label2.Font.Color:=clRed;
      Label2.Caption:='lub ma inną nazwę niż O2.txt';
```

```
            iw9:='.\Wynik\H.bmp';
        end;
        21: - błędna odpowiedź z propozycją skorzystania z pomocy
        begin
            Form9.Caption:='Wynik';
            Button1.Caption:='Tak';
            Button2.Visible:=true;
            Button2.Caption:='Nie';
            Label1.Font.Color:=clRed;
            Label1.Caption:='Wpisana pochodna jest błędna';
            Label2.Font.Color:=clBlack;
            Label2.Caption:='Chcesz skorzystać z pomocy?';
            iw9:='.\Wynik\F.bmp';
        end;
        22: - pytanie o zmianę funkcji
        begin
            Form9.Caption:='Zmiana funkcji';
            Button1.Caption:='Tak';
            Button2.Visible:=true;
            Button2.Caption:='Nie';
            Label2.Font.Color:=clPurple;
            Label2.Caption:='Chcesz inną funkcję?';
            Label1.Caption:='';
            iw9:='.\Wynik\A.bmp';
        end;
        23: - zachęta do skorzystania z pomocy
        begin
            Form9.Caption:='Pomocnik';
            Button1.Caption:='Tak';
            Button2.Visible:=true;
            Button2.Caption:='Nie';
            Label2.Font.Color:=clPurple;
            Label2.Caption:='Potrzebujesz pomocy?';
            Label1.Caption:='';
            iw9:='.\Wynik\G.bmp';
        end;
        24: - nie znaleziono wartości dla 'x' – funkcji nie można sprawdzić
        begin
            Form9.Caption:='Błąd';
            Button1.Caption:='Rozumiem';
            Button2.Visible:=false;
            Label1.Font.Color:=clBlack;
            Label1.Caption:='Tej funkcji nie można sprawdzić';
            Label2.Font.Color:=clBlack;
            Label2.Caption:='Wybierz inną';
            iw9:='.\Wynik\H.bmp';
        end;
        25: - brak pliku podpowiedzi
        begin
            Form9.Caption:='Błąd';
            Button1.Caption:='Rozumiem';
            Button2.Visible:=false;
            Label2.Font.Color:=clBlack;
            Label2.Caption:='Brak pliku podpowiedzi';
            Label2.Caption:='';
            iw9:='.\Wynik\H.bmp';
        end;
end;
if FileExists(iw9) then Image1.Picture.LoadFromFile(iw9)
```

załadowanie obrazka „buźki" do komponentu `Image1`, pod warunkiem, że plik istnieje

```
        else ShowMessage('Brak pliku graficznego'); jeśli plik nie istnieje...
                    wyświetl okno komunikatu ShowMessage ze stosowną informacją
        Form9.ActiveControl:=Button1; - zadeklaruj aktywną kontrolkę
end;
```

W nagłówku procedury `Wynik9` znajdują się trzy parametry przekazywane przez wartość, z czego dwa ostatnie posiadają wartości domyślne zainicjowane wartościami zero. Oznacza to, że wywołując procedurę w celu wyświetlenia komunikatu innego niż podsumowanie po zakończeniu ćwiczeń, można pominąć dwa ostatnie parametry:

```
Form9.Wynik9(16);
wywołanie procedury w celu wyświetlenia komunikatu o poprawnej odpowiedzi – dwa pozostałe
parametry zostały pominięte

Form9.Wynik9(14,dob2,zle2);
wywołanie procedury w celu wyświetlenia podsumowania po zakończeniu ćwiczeń
```

Funkcja `FileExists` sprawdza, czy plik graficzny z podaną lokalizacją w argumencie istnieje – jeśli tak, funkcja zwraca wartość `true`, co pozwala na załadowanie do komponentu `Image1` wskazanego pliku graficznego, w przeciwnym przypadku wyświetlony zostaje komunikat o braku pliku – procedura `Wynik9` będzie działała dalej, lecz bez pliku graficznego.

Zadeklarowanie aktywnej kontrolki zdecydowanie ułatwia użytkownikowi współpracę z programem. Aktywna kontrolka jako pierwsza reaguje na zdarzenia z klawiatury. Ułatwienie dla użytkownika polega więc na tym, że dla oczywistej odpowiedzi może użyć klawisza ENTER, bez angażowania myszki i celowania nią w odpowiedni przycisk na ekranie. Oczywiście, jeśli użytkownik chce nacisnąć drugi przycisk, musi użyć myszki komputerowej.

Dla ułatwienia wyboru tekstu komunikatu o błędzie, w procedurze `Wynik9` znajduje się wewnętrzna funkcja `Bledy`, która – na podstawie dostarczonego jej kodu błędu – zwraca odpowiedni tekst komunikatu.

```
function Bledy(const kod_bl:byte):string;
begin
     case kod_bl of
          1:Result:='Nieokreślony błąd programu';
          2:Result:='Za duży wykładnik potęgi';
          3:Result:='Nierówna liczba nawiasów';
          4:Result:='Niewłaściwa podstawa logarytmu';
          5:Result:='Za duża podstawa logarytmu';
          6:Result:='Niedozwolony znak';
          7:Result:='To nie jest funkcja';
          8:Result:='Zabrakło pamięci';
          9:Result:='Dzielenie przez zero';
          10:Result:='Niedozwolony zapis';
          11:Result:='Funkcji nie można policzyć';
          12:Result:='Brak funkcji za znakiem działania';
          13:Result:='Funkcji nie mogę sprawdzić';
          else Result:=''
     end;
end;
```

Zamknięcie formy może odbyć się tylko za pomocą jednego z przycisków osadzonych na formie, dlatego wszystkie inne przyciski, dostępne na pasku tytułowym formy, zostały usunięte poprzez nadanie im wartości `false` we właściwościach formy `BorderIcons[]`.

Użycie jednego z przycisków `Button1` lub `Button2`, poza zamknięciem formy instrukcją `close`, musi spowodować zwrócenie do programu wywołującego wartość zamknięcia formy, na podstawie której, program będzie mógł rozróżnić, który przycisk został użyty. Mając na uwadze fakt, że instrukcja `close` zwraca, za pośrednictwem zmiennej `ModalResult`, własną wartość zamknięcia, konieczne było zignorowanie tej wartości, czyli umieszczenie instrukcji `Form9.ModalResult` bezwzględnie po instrukcji `close`.

```
procedure TForm9.Button1Click(Sender: TObject);
zdarzenie kliknięcia przycisku Button1
begin
     Form9.Close; - zamknij formę komunikatów
     Form9.ModalResult:=1; - po zamknięciu formy zwróć, za pośrednictwem...
                             zmiennej ModalResult, wartość jeden
end;

procedure TForm9.Button2Click(Sender: TObject);
zdarzenie kliknięcia przycisku Button2
begin
     Form9.Close; - zamknij formę komunikatów
     Form9.ModalResult:=2; - po zamknięciu formy zwróć, za pośrednictwem...
                             zmiennej ModalResult, wartość dwa
end;
```

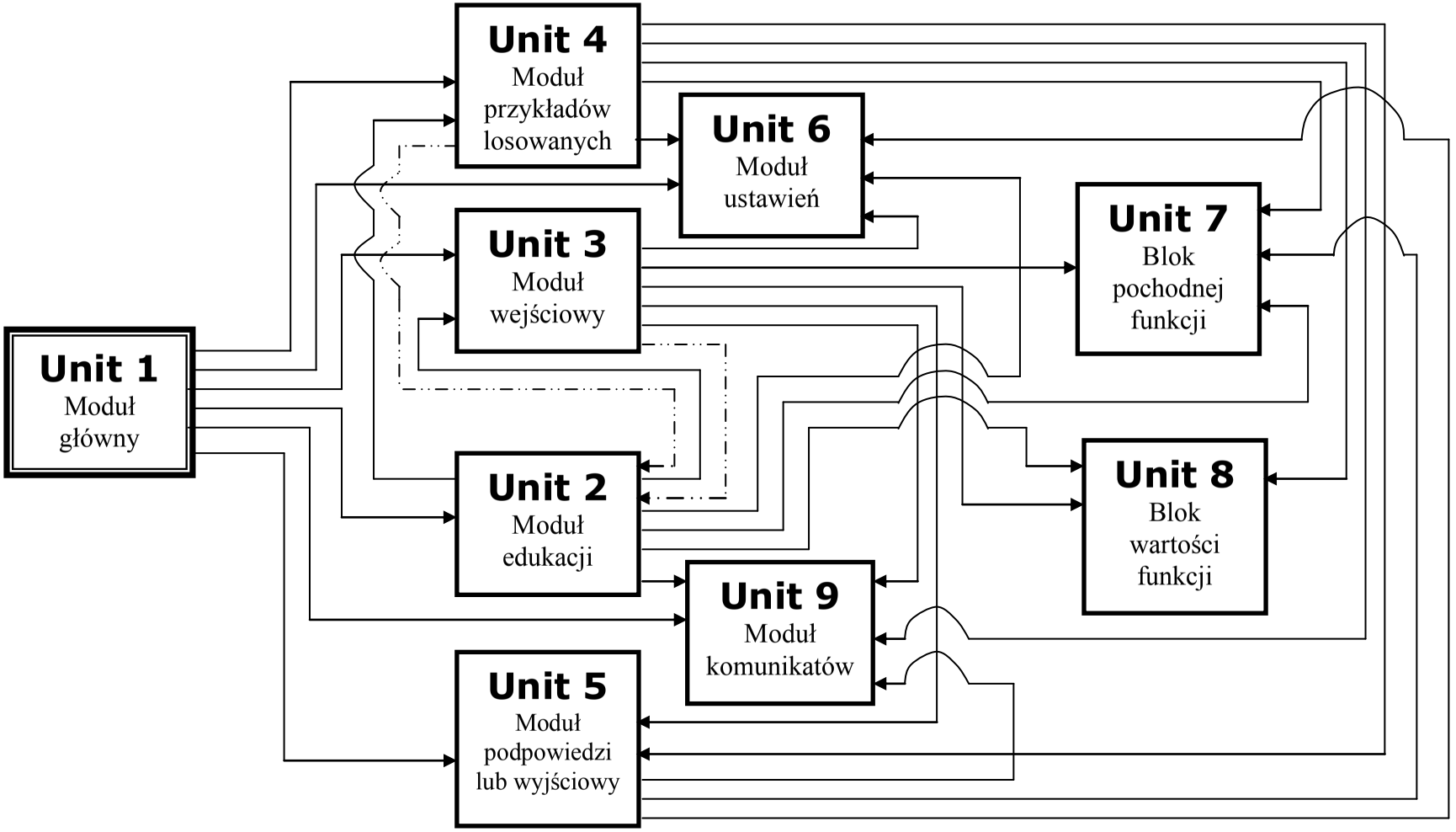
XIII. Współpraca modułów ze sobą

Moduły w Delphi są osobnymi plikami składającymi się z klas, procedur, zmiennych, stałych i innych konstrukcji językowych. Opisywana aplikacja składa się z dziewięciu modułów, które scharakteryzowano w poprzednich rozdziałach. Do poprawnej pracy aplikacji potrzebna jest lista odwołań do modułów, zawarta w sekcji `uses` każdego z nich. Zapewniony jest przez to dostęp jednego modułu do udostępnionych zasobów innego. Sekcja `uses` może znajdować się zarówno w części `interface`, jak i `implementation`, każdego z modułów. Część pierwsza zawiera, prócz deklaracji typów stałych, zmiennych, procedur i funkcji, także podstawową listę modułów. Część druga, poza kodem funkcji i procedur zadeklarowanych w części `interface`, może zawierać także dodatkową sekcję `uses`. Odwołania do modułów nie mogą być wzajemne, to znaczy nie jest dozwolona sytuacja, gdy moduł A zawiera w sekcji `uses` nazwę modułu B a moduł B w sekcji `uses` nazwę modułu A. Takie odwołanie nazywa się cyklicznym, które jest niepoprawne, aczkolwiek dozwolone – pod warunkiem, że nazwa modułu powodującego cykliczne odwołanie zostanie umieszczona w części `implementation`. Sytuacja taka pojawiła się w modułach `Unit3` i `Unit4` w stosunku do modułu `Unit2`.

Uwzględnienie nazwy modułu `Unit2` w sekcji `uses`, umieszczonej w części `implementation` wymienionych modułów, umożliwiło dostęp do procedury `SetFocus` należącej do klasy `TForm2`, powodując przyjęcie aktywności przez dolną formę edukacji i uruchomienia na niej mrugania kursora, jeśli po napisaniu funkcji przez użytkownika, na górnej formie wejściowej naciśnięty zostanie klawisz ENTER. Podobnie, gdy użytkownik korzystający z opcji programu, w którym funkcje są losowane przez komputer, kliknięciem wymusi zmianę funkcji. Tak jak w poprzednim przypadku, dolna forma edukacji staje się aktywna a mruganie kursora – automatycznie uruchomione. Można by zrezygnować z tych funkcjonalności, zmuszając użytkownika do każdorazowego kliknięcia myszką w dolną formę, ale w dłuższej perspektywie czasu może być to dla niego uciążliwe.

Zwiększanie liczby błędnych odpowiedzi przez procedurę `zwieksz_zle_odp`, należącej do klasy `TForm2`, możliwe było dzięki umieszczeniu nazwy modułu `Unit2` w części `implementation` modułów `unit3` i `unit4`. Zliczanie tych odpowiedzi odbywa się po wymuszeniu zmiany funkcji w czasie ćwiczeń.

Pozostałe odwołania są już poprawne. Schemat odwołań do modułów prezentuje rysunek 40.



Rys. 40. Odwołania modułów zastosowanych w programie. Strzałkami ciągłymi zaznaczono odwołania poprawne, natomiast przerywanymi – odwołania cykliczne

XIV. Edytor równań – z edycją

Edytor równań należy do najbardziej złożonej części programu. Składa się z wielu podprogramów, które ściśle współpracują ze sobą w procesie układania znaków na formie w sposób matematyczny. Najważniejszym elementem edytora jest główna struktura, której elementy lokowane są w pamięci komputera po każdym wprowadzonym znaku. Sposób łączenia elementów edytora ze sobą jest kombinowany. Trudno znaleźć nazwę takiego wiązania, choćby z racji tego, że składa się z dwóch rodzajów wiązań, może przypominać graf. Pierwsze wiązanie jest typowe dla listy dwukierunkowej, ale drugie zależne jest od istnienia lub nie znaków akcji, wymagających dodatkowego wiązania z pozostałymi elementami edytora, tworząc z nimi otoczenie znaku akcji. Dodatkowe związanie umożliwia swobodne poruszanie się po ułamku, pierwiastku itp. za pomocą przycisków sterowania kursorem oraz racjonalne dołączanie lub usuwanie składników otoczenia danego znaku akcji, zapewniając integralność tego otoczenia. Zarządzanie taką strukturą jest sporym utrudnieniem dla programisty, ale tylko w taki sposób można uzyskać pożądany efekt edytora ekranowego. Poniżej zaprezentowany został kod struktury edytora o nazwie `lisc`.

```
type
  lisc=^zawartosc;
  zawartosc=record
                  prawa:lisc;
                  lewa:lisc;
                  kur_g:lisc;
                  kur_d:lisc;
                  kur_l:lisc;
                  kur_p:lisc;
                  lx:integer;
                  ly:integer;
                  px:integer;
                  py:integer;
                  y1:integer;
                  y2:integer;
                  znak:char;
                  druk:boolean;
                  zazn:boolean
  end;
```

Przeznaczenie poszczególnych pól jest następujące:

- `prawa`, `lewa` – to główne wiązania składników tworzących tzw. listę dwukierunkową;
- `kur_g`, `kur_d`, `kur_l`, `kur_p` – to wiązania kursorowe, umożliwiające poruszanie się po funkcji za pomocą klawiszy kierunkowych klawiatury;
- `lx`, `px` – współrzędne poziome – pierwsze pole to współrzędna z lewej strony znaku, drugie to współrzędna z prawej strony znaku. Różnica obu pól wskazuje na szerokość znaku;
- `ly`, `py` – współrzędne pionowe poziomu znaków, na których znajduje się znak;
- `y1`, `y2` – współrzędne pionowe rzeczywiste znaku, określające jego wysokość;
- `znak` – znak należący do funkcji. Może być nim także spacja wstawiana automatycznie przez edytor;
- `druk` – pole zawierające informację logiczną, czy znak ma być widoczny czy ukryty;

– `zazn` – pole zawierające informację logiczną, czy znak został zaznaczony czy nie.

Wartości w polach `ly` i `y1` oraz `py` i `y2` najczęściej są sobie równe. Istnieją jednak znaki, takie jak: nawiasy, pierwiastek lub kreska ułamkowa, których wysokość rzeczywista musi być ustalana indywidualnie. Obecność dwóch par współrzędnych pionowych pozwala na zachowanie poziomu dla wstawianych znaków w jednym rzędzie, dzięki czemu każdy następny znak dołączany, np. za nawiasem zamykającym, będzie miał właściwe współrzędne pionowe, gdyż pobierze je od swego poprzednika z niezmienionych pól `ly` i `py`. Pola `y1` i `y2` będą wykorzystane podczas wyprowadzania znaków na formę przez procedurę `Wyswietl`.

Dostęp do elementów edytora umożliwiają dwa wskaźniki: `wsp` wskazujący na pierwszy element i `wsb` wskazuje na bieżący element struktury `lisc`, czyli ten, przy którym znajduje się kursor.

By można było łatwiej określić pewne grupy znaków, zastosowano tablice stałych wartości liczbowych, do których procedury będą się odnosić. Liczby te reprezentują wartości znaków w kodzie ASCII. Ich budowa jest następująca:

```
const
  otw=[40,91,123];- stałe nawiasów otwierających: '(', '[', '{'
  zam=[41,93,125];- stałe nawiasów zamykających: ')', ']', '}'
  dzia=[42,43,45];- stałe znaków działania: mnożenia, dodawania, odejmowania
  dzed=[47,92,94,95];- stałe znaków akcji: dzielenia, pierwiastkowania, potęgowania...
                        oraz znak podkreślenia wstawiany automatycznie przez…
                        program, rozpoznawany jako logarytm zwykły
```

XIV. 1. Wprowadzenie znaku

Wprowadzenie znaku z klawiatury uruchamia zdarzenie `FormKeyPress`. Procedura obsługująca to zdarzenie zwraca w parametrze `Key` wprowadzony znak. Po wczytaniu go do lokalnej zmiennej w postaci kodu ASCII, następuje sprawdzenie, czy znak ten odpowiada wielkiej literze, jeśli tak, zostaje on zamieniony na odpowiednik małej litery.

```
a1:=ord(Key); - wpisz do zmiennej a1 wartość wprowadzonego znaku z klawiatury
if (a1>=65) and (a1<=90) then - czy wartość znaku zawiera się między 65 a 90?
begin - jeśli tak, to znak jest wielką literą, więc zamień go na odpowiednik małej litery...
     a1:=a1+32; - do wartości znaku dodaj liczbę 32
     zn:=chr(a1) - tak zmodyfikowaną liczbę zamień na znak typu Char i zapamiętaj go…
                   w zmiennej znakowej zn
end else zn:=Key; - w przeciwnym razie przepisz znak do zmiennej znakowej zn
```

Kolejne instrukcje warunkowe odrzucają znaki, które nie będą wykorzystane w pisanych funkcjach czy pochodnych, np. przecinek czy pionowa kreska oraz znaki niedrukowalne.

```
if ((a1>=40) and (a1<=57)) or
   ((a1>=65) and (a1<=94)) or
   ((a1>=97) and (a1<=123)) or
   (a1=125) then
   begin
        zczas:=0; - wyzeruj czas oczekiwania wpisując zmiennej prywatnej zczas...
                    wartość zero
        Wstaw_znak(zn) - wywołaj procedurę Wstaw_znak
   end;
```

Po wczytaniu do zmiennej `zn` właściwego znaku, wywoływana jest procedura o nazwie `Wstaw_znak`, której w parametrze podany zostaje znak wprowadzony z klawiatury.

XIV. 1.1. Dołączenie nowego elementu do struktury edytora – procedura `Wstaw_znak`

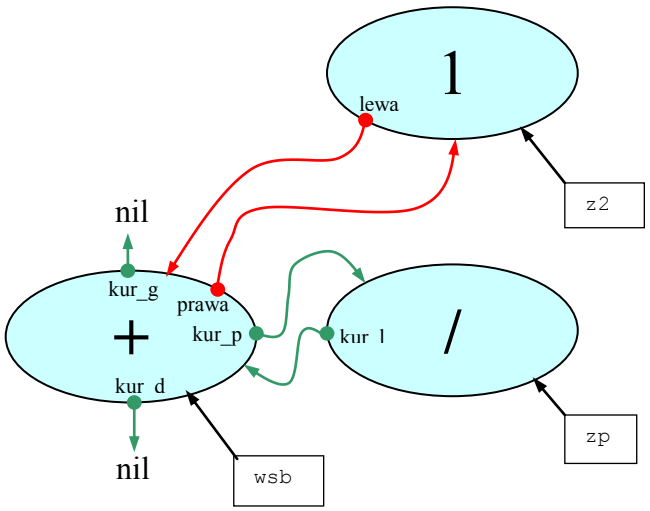
W pierwszej instrukcji procedury sprawdzana jest obecność bieżącego elementu `lisc` wskazywanego przez wskaźnik `wsb` – jeśli jego adres nie jest pusty, a więc bieżący element istnieje, sprawdzany jest znak wpisany w jego pole `znak` – jeśli jest w nim znak akcji, nowy element edytora nie może zostać z nim związany, gdyż doprowadziłoby to do rozbicia otoczenia tego znaku akcji a w konsekwencji do załamania się programu. By dołączenie nowego składnika stało się możliwe, bieżący element zostaje zmieniony na jeden ze składników otoczenia znaku akcji, a więc tego, który związany został wiązaniem kursorowym z jego lewej lub z prawej strony – o wyborze decyduje usytuowanie kursora (z lewej lub prawej strony znaku akcji). Po zmianie elementu bieżącego, zmienna `stat_kur` decydująca o usytuowaniu kursora, zostaje ustawiona tak, by nowy składnik edytora znalazł się między wybranym elementem bieżącym a tym, z wpisanym znakiem akcji.

```
. . .
z1:=nil;
if wsb<>nil then - czy wskaźnik na bieżący element lisc nie ma adresu pustego?
begin - jeśli nie, to...
     if ord(wsb^.znak) in dzed then - czy bieżący element zawiera…
                                      znak akcji?
        if stat_kur then - jeśli tak, to czy kursor ustawiony jest za znakiem?
        begin - jeśli tak, to…
             wsb:=wsb^.kur_p; - pozyskaj we wskaźniku wsb adres następnika
                                elementu bieżącego w wiązaniu kursorowym
             stat_kur:=false - ustaw kursor z lewej strony znaku
        end else - jeśli nie, kursor ustawiony jest przed znakiem, więc…
            begin
                 z1:=wsb^.kur_l; - do wskaźnika z1 wczytaj adres poprzednika
                                   w wiązaniu kursorowym elementu bieżącego
                 if z1<>nil then - czy element istnieje?
                 begin - jeśli tak, to...
                       wsb:=z1; - wpisz do wskaźnika wsb adres tego elementu
                       stat_kur:=true - ustaw kursor z prawej strony znaku
                 end;
            end;
        z1:=wsb; - wpisz lokalnemu wskaźnikowi z1 adres bieżącego elementu
        . . .
```

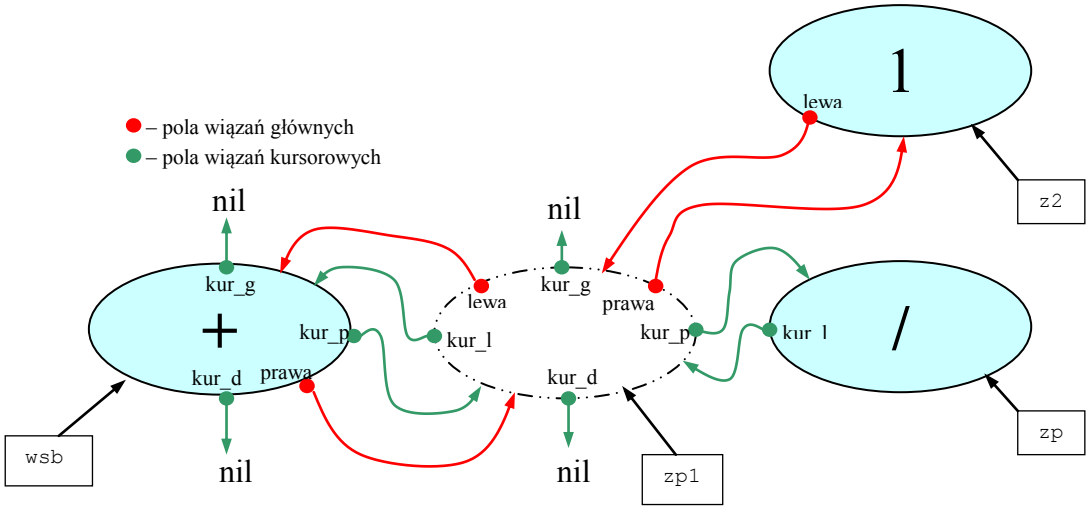
Wprowadzenie z klawiatury znaku akcji wymaga zbadania otoczenia istniejącego elementu bieżącego, dostępnego przez wskaźnik `wsb`. Jeśli zmienna `stat_kur` ustawiona została w pozycji `true` oznaczającą usytuowanie kursora z prawej strony znaku bieżącego, sprawdzany jest znak wpisany w pole `znak` elementu związanego z prawej strony elementu bieżącego. Gdy zawiera on znak dzielenia lub potęgowania, niezbędne jest utworzenie dodatkowego składnika struktury `lisc` z wpisanym znakiem spacji i dołączenie go między tym elementem a bieżącym, wskazywanym przez wskaźnik `wsb`. Podobnie, gdy kursor usytuowany jest z lewej strony znaku bieżącego, o czym informuje ustawienie zmiennej `stat_kur` w pozycji `false`. Sprawdzany jest wówczas znak zawarty w polu `znak` składnika związa-

nego z lewej strony elementu bieżącego – jeśli jest nim znak dzielenia lub potęgowania, tak jak w poprzednim przypadku, pomiędzy ten element a bieżący zostaje dołączony dodatkowy składnik struktury `lisc` z wpisanym znakiem spacji.

Może się zdarzyć, że bieżący element posiada wpisany znak akcji. Obecność dwóch znaków akcji w bezpośrednim sąsiedztwie doprowadziłaby do rywalizacji w pozyskiwaniu elementów edytora do swojego otoczenia znaku akcji, czego skutkiem byłoby niepoprawne ułożenie znaków na formie. Dołączenie dodatkowego składnika między bieżący a nowy element zapewnia ochronę otoczenia istniejącego już znaku akcji przed rywalizacją.



Rys. 41a. Sytuacja sprzed dołączenia dodatkowego składnika listy między bieżącym składnikiem (wsb) a jego następnikiem (zp), gdy wstawiany znak należy do znaków akcji



Rys. 41b. Sytuacja po dowiązaniu dodatkowego składnika listy, zaznaczonego na rysunku linią przerywaną, zastępujący dotychczasowy składnik otoczenia znaku dzielenia z wpisanym znakiem '+'

Rysunki 41a i 41b przedstawiają sytuację sprzed dołączenia dodatkowego składnika oraz po jego dołączeniu, gdy kursor ustawiony jest z lewej strony znaku dzielenia. Nowy element ze znakiem wprowadzonym z klawiatury zostanie dołączony między element wskazywany przez wskaźnik bieżący, czyli `wsb` a nowy element, wskazywany na rysunku przez

wskaźnik zp1. Gdy kursor ustawiony jest z prawej strony znaku akcji, sposób dowiązania nowego składnika jest odwrotny do zaprezentowanego na rysunku.

Ochrona otoczenia znaku akcji musi być zapewniona również wtedy, gdy bieżącym znakiem jest spacja. Element ze znakiem spacji zawsze jest granicznym elementem otoczenia znaku akcji wstawianym przez program automatycznie. Jeśli jest to element bieżący, dodatkowy element musi być dołączony między składnik ze znakiem spacji a nowym elementem, do którego będzie dołączony znak akcji wpisany z klawiatury.

```
. . .
if ord(wzn1) in dzed then - czy wprowadzony z klawiatury znak należy do...
                            znaków akcji?

begin - jeśli tak, to...
     rozb1:=false; - wpisz startową wartość false zmiennej logicznej rozb1
     kier1:=false; - wpisz startową wartość false zmiennej logicznej kier1
     if stat_kur then - czy kursor ustawiony jest za znakiem?
     begin - jeśli tak, to…
          kier1:=true; - ustaw zmienną kier1 w pozycji true
          zp:=wsb^.kur_p; - zmiennej lokalnej zp wpisz adres następnika...
                            bieżącego elementu w wiązaniu kursorowym

          if zp<>nil then - czy następnik bieżącego elementu istnieje?
             if (zp^.znak='/')
             or (zp^.znak='^') then rozb1:=true; - jeśli tak, to gdy…
                         element wskazywany przez wskaźnik zp ma wpisany...
                         znak dzielenia lub potęgowania (znak pierwiastkowania...
                         nie ma struktury otwierającej), ustaw zmienną rozb1...
                         w pozycji true

          if ord(wsb^.znak) in dzed then rozb1:=true; jeśli …
                                element bieżący ma wpisany znak akcji,...
                                to ustaw zmienną rozb1 w pozycji true

          if wsb^.znak=' ' then - czy element bieżący ma wpisaną spację?
          begin - jeśli tak, to…
               zp:=wsb^.kur_l; - zmiennej lokalnej zp wpisz adres...
                 poprzednika bieżącego elementu w wiązaniu kursorowym

               if zp<>nil then - czy poprzednik bieżącego elementu istnieje?
                  if ord(zp^.znak) in dzed then - jeśli tak, to czy…
                                     poprzednik ma wpisany znak akcji?
                  begin - jeśli tak, to…
                       rozb1:=true; - ustaw zmienną rozb1 w pozycji...
                                      true
                       kier1:=false - ustaw zmienną kier1 w pozycji...
                                      false
                  end
          end
     end else - jeśli nie, kursor ustawiony jest przed znakiem, więc…
         begin
              zp:=wsb^.kur_l; - zmiennej lokalnej zp wpisz adres poprzednika..
                                bieżącego elementu w wiązaniu kursorowym

              if zp<>nil then - czy poprzednik bieżącego elementu istnieje?
                 if ord(zp^.znak) in dzed then rozb1:=true;
                 jeśli tak, to gdy ma wpisany znak akcji,...
                 ustaw zmienną rozb1 w pozycji true


              if (wsb^.znak='/')
              or (wsb^.znak='^') then rozb1:=true; jeśli bieżący...
```

```
                    element ma wpisany znak dzielenia lub potęgowania,...
                    ustaw zmienną rozb1 w pozycji true
     if wsb^.znak=' ' then - czy element bieżący ma wpisaną...
                             spację?
     begin - jeśli tak, to...
          zp:=wsb^.kur_p; - zmiennej lokalnej zp wpisz adres...
             następnika bieżącego elementu w wiązaniu kursorowym
          if zp<>nil then - czy następnik bieżącego elementu…
                            istnieje?
             if (zp^.znak='/')
             or (zp^.znak='^') then - czy następnik ma…
                     wpisany znak dzielenia lub potęgowania?
             begin - jeśli tak, to…
                  rozb1:=true; - ustaw zmienną rozb1...
                                 w pozycji true
                  kier1:=true - ustaw zmienną kier1...
                                w pozycji true

             end
     end;
end;
if rozb1 then - czy zmienna rozb1 została ustawiona w pozycji true?
begin - jeśli tak, to...
new(zp1); - utwórz w pamięci nowy składnik listy
if kier1 then - czy zmienna kier1 została ustawiona w pozycji true?
begin - jeśli tak, to...
     z2:=wsb^.prawa;
     z2^.lewa:=zp1;
     zp^.kur_l:=zp1;
     zp1^.prawa:=z2;
     zp1^.kur_p:=zp;      wstaw nowy składnik między bieżącym...
     zp1^.lewa:=wsb;      elementem z lewej strony a elementem...
     zp1^.kur_l:=wsb;     znaku akcji z prawej strony
     wsb^.prawa:=zp1;
     wsb^.kur_p:=zp1;
     zp1^.lx:=wsb^.px
 end else - jeśli nie, kursor ustawiony jest z lewej strony znaku, więc...
     begin
          z2:=wsb^.lewa;
          z2^.prawa:=zp1;
          zp^.kur_p:=zp1;
          zp1^.lewa:=z2;       wstaw nowy składnik między...
          zp1^.kur_l:=zp;      elementem znaku akcji z lewej...
          zp1^.prawa:=wsb;     strony a bieżącym elementem...
          zp1^.kur_p:=wsb;     z prawej strony
          wsb^.lewa:=zp1;
          wsb^.kur_l:=zp1;
          zp1^.lx:=wsb^.lx
     end;
 with zp1^ do
 begin
      kur_g:=nil;
      kur_d:=nil;
      px:=lx;
      ly:=wsb^.ly;
      py:=wsb^.py;
      y1:=ly;
```

```
                    y2:=py;
                    znak:=' ';
                    druk:=true
               end;
          end;
     end;
```

Jeśli wprowadzony znak z klawiatury nie należy do znaków akcji a w polu `znak` bieżącego elementu `lisc` znajduje się spacja, nowy znak zastępuje znak spacji, z uwagi, że szerokość znaku spacji została wcześniej zredukowana do zera, dla nowego znaku szerokość ta musi zostać wyliczona, zaś pole `px` współrzędnej poziomej bieżącego elementu zredukowane stosownie do szerokości wprowadzonego znaku. By następne znaki nie przysłaniały nowego znaku, muszą one zostać przesunięte w prawo o wyliczoną szerokość nowego znaku.

```
if wsb^.znak=' ' then - czy pole znak bieżącego elementu ma wpisaną spację?
begin
     wsb^.znak:=wzn1; - zastąp spację nowym znakiem
     stat_kur:=true; - bezwzględnie ustaw zmienną stat_kur w pozycji...
                       true (kursor za znakiem)
     Form2.Szerokosc(wsb); - wylicz szerokość nowego znaku i zmodyfikuj...
                             pola współrzędnych poziomych bieżącego elementu
     Form2.Przesun(wsb,false); - przesuń w prawo pozostałe znaki,...
                 modyfikując pola współrzędnych poziomych tych elementów,...
                 które znajdują się na tym samym poziomie, co bieżący element
     Form2.Odswiez_funkcje; - ułóż ponownie funkcję/pochodną w sposób...
                              matematyczny, uwzględniając nowy znak
     exit - opuść procedurę
end;
```

Kolejną instrukcją jest sprawdzenie, czy znaki wpisane w pola `znak` sąsiadujących elementów `lisc`, nie tworzą słowa `log`, czyli logarytmu o dowolnej podstawie, jeśli tak, procedura musi zostać opuszczona, gdyż wstawienie nowego znaku między litery `log` mogłoby rozbić nazwę tej funkcji i uniemożliwić późniejsze jej rozpoznanie.

```
zp:=wsb; - do wskaźnika zp wczytaj adres bieżącego elementu lisc
if stat_kur then - czy zmienna stat_kur ustawiona jest w pozycji true?
   if czy_log(zp) then exit; - jeśli tak, to gdy funkcja czy_log zwróci...
                               wartość true, opuść procedurę
```

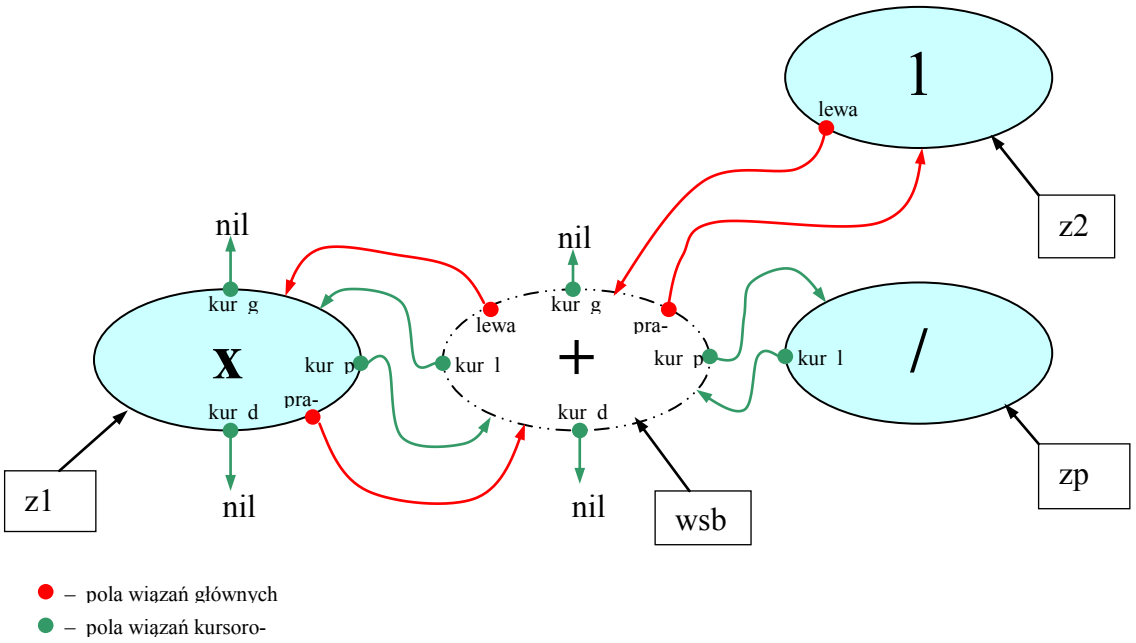
Jeśli w wyniku powyższych instrukcji nie nastąpiło opuszczenie procedury, w dalszych instrukcjach następuje utworzenie w pamięci nowego elementu `lisc` dla nowego znaku. Znajdują się tu trzy grupy instrukcji, których uruchomienie zależne jest od usytuowania kursora w obrębie bieżącego znaku oraz od istnienia lub nie bieżącego elementu `lisc`. Sposób wiązania nowego elementu z pozostałymi, gdy kursor w chwili wprowadzania nowego znaku z klawiatury usytuowany jest za znakiem bieżącego elementu `lisc`, prezentuje poniższy kod.

```
new(wsb); - utwórz w pamięci nowy element lisc,...
            wykorzystując główny wskaźnik edytora
if stat_kur then - czy zmienna stat_kur ustawiona jest w pozycji true?
begin - jeśli tak, to...
     z2:=z1^.prawa; - w zmiennej z2 pozyskaj adres następnika bieżącego elemen-
tu...
                      w wiązaniu głównym
```

```
            zp:=z1^.kur_p; - w zmiennej zp pozyskaj adres następnika bieżącego…
                             elementu w wiązaniu kursorowym
            with wsb^ do - instrukcja wiążąca wobec nowego elementu
            begin
                 prawa:=z2;
                 lewa:=z1;
                 kur_p:=zp;
                 kur_l:=z1;
                 kur_g:=nil;
                 kur_d:=nil;
                 lx:=z1^.px;
                 ly:=z1^.ly;
                 py:=z1^.py;
                 y1:=ly;
                 y2:=py;
                 znak:=wzn1
            end;
            if zp<>nil then zp^.kur_l:=wsb; - gdy następnik w wiązaniu...
                            kursorowym istnieje, związ go z nowym elementem
            z1^.kur_g:=nil;
            z1^.kur_d:=nil;
            Form2.Szerokosc(wsb); - wylicz szerokość nowego znaku, modyfikując...
                                    pole px współrzędnej poziomej nowego elementu
            if z2<>nil then z2^.lewa:=wsb; - jeśli następnik w wiązaniu głównym...
                                             istnieje, związ go z nowym elementem
            z1^.prawa:=wsb;  } związ bieżący element z nowym w wiązaniu głównym...
            z1^.kur_p:=wsb;  } i kursorowym
            Form2.Przesun(wsb,false) - przesuń w prawo pozostałe znaki,...
                     modyfikując pola współrzędnych poziomych tych elementów,...
                     które znajdują się na tym samym poziomie co bieżący element
        end else
```

Wartości współrzędnych pionowych dla pól `ly` i `py` nowego elementu zostają pozyskane z tych samych pól współrzędnych pionowych, które należą do bieżącego elementu, dostępnego przez wskaźnik `z1`, natomiast wartość współrzędnej poziomej `lx` jest wartością z pola `px` bieżącego elementu. Takie pozyskanie wartości współrzędnych od swego poprzednika powoduje, że nowy znak będzie widoczny na ekranie przy swoim poprzedniku z prawej strony i w tym samym rzędzie. Wiązania z elementami znajdującymi się nad lub pod nowym elementem nie należą do tej procedury, dlatego pola `kur_g` i `kur_d` zainicjowane są adresami pustymi. Gdy nowy element `lisc` został już poprawnie związany z elementami z lewej i prawej strony, zostaje mu nadana szerokość a elementy usytuowane z prawej jego strony, są przesunięte w prawo o szerokość nowego znaku. Odpowiedni schemat związania elementów ze sobą, gdy kursor ustawiony jest za znakiem bieżącym, przedstawia rysunek 42.

Gdy podczas wprowadzania nowego znaku, zmienna `stat_kur` ustawiona jest w pozycji `false`, co powoduje usytuowanie kursora przed znakiem bieżącym, sytuacja jest nieco inna, ponieważ bieżący element może jeszcze nie istnieć. Gdy jednak bieżący element, dostępny przez wskaźnik `z1` istnieje, będzie on związany z nowym elementem z lewej jego strony, przez co nowy znak umieszczony zostanie na ekranie przed znakiem wpisanym w pole `znak` bieżącego elementu. Tak jak w poprzednim przypadku, wartości współrzędnych pionowych `ly` i `py` zostaną skopiowane z pól współrzędnych bieżącego elementu, natomiast wartość współrzędnej poziomej `lx` pochodzi w tym przypadku z pola `lx` bieżącego elementu.

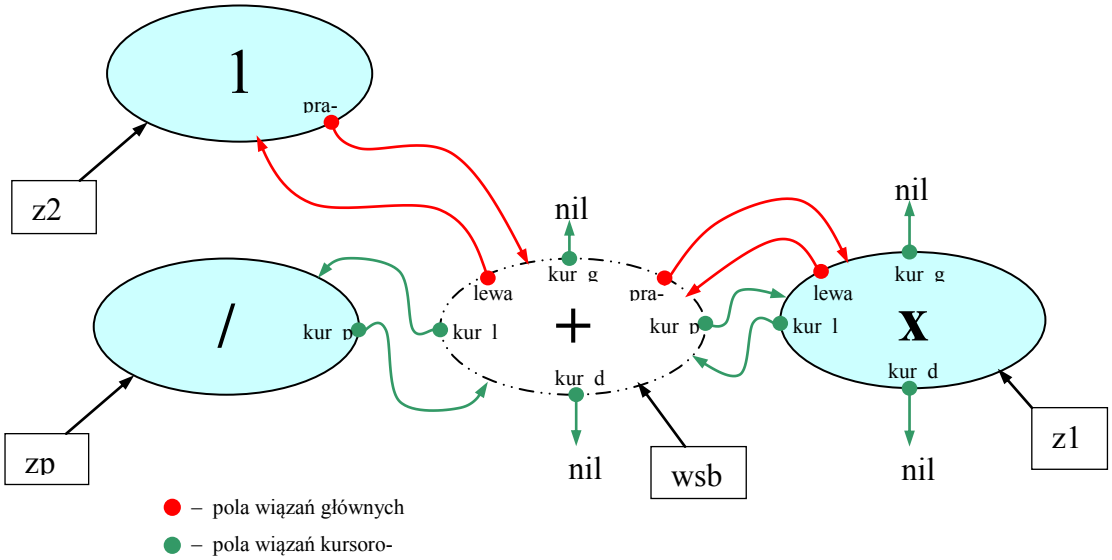


Rys. 42. Dowiązanie nowego elementu, dostępnego przez wskaźnik `wsb`, wstawiając go za elementem bieżącym, dostępnym przez wskaźnik `z1` i istniejący następnik (zp)

```
new(wsb); – utwórz w pamięci nowy element lisc wykorzystując główny wskaźnik...
            edytora
if stat_kur then – czy zmienna stat_kur ustawiona jest w pozycji true?
begin
     . . .
end else – jeśli nie, to ustawiona jest w pozycji false, czyli...
    begin
         if z1<>nil then – czy bieżący element istnieje?
         begin – jeśli tak, to...
              z2:=z1^.lewa; – do zmiennej z2 wczytaj adres poprzednika...
                              bieżącego elementu w wiązaniu głównym

              zp:=z1^.kur_l; – do zmiennej zp wczytaj adres poprzednika...
                               bieżącego elementu w wiązaniu kursorowym

              with wsb^ do
              begin
                   lewa:=z2;
                   prawa:=z1;
                   kur_l:=zp;
                   kur_g:=nil;
                   kur_d:=nil;
                   kur_p:=z1;
                   lx:=z1^.lx;
                   ly:=z1^.ly;
                   py:=z1^.py;
                   y1:=ly;
                   y2:=py;
                   znak:=wzn1
              end;
              z1^.kur_g:=nil;
              z1^.kur_d:=nil;
              Form3.Szerokosc(wsb);
              if z2<>nil then z2^.prawa:=wsb else wsp:=wsb;
```

```
            if zp<>nil then zp^.kur_p:=wsb;
            z1^.lewa:=wsb;
            z1^.kur_l:=wsb;
            stat_kur:=true;
            Form3.Przesun(wsb,false)
        end else
```

Schemat wiązania elementów ze sobą, gdy kursor ustawiony jest przed znakiem bieżącym, przedstawia rysunek 43.



Rys. 43. Wiązanie nowego elementu, dostępnego przez wskaźnik `wsb`, wstawiając go przed elementem bieżącym, dostępnym przez wskaźnik `z1`

Gdy bieżący element nie istnieje, czyli wskaźnikowi `z1` przypisany został adres pusty, oznacza to, że utworzony element `lisc` jest pierwszym elementem całej struktury. W tej sytuacji, wszystkie współrzędne dla pierwszego elementu są pozyskiwane ze zmiennych startowych `AX` i `AY` należących do bieżącego modułu, natomiast wysokość czcionki pozyskiwana jest w zmiennej modułu regulacji – `RZ`. Wszystkie pola wiązań zostają zainicjowane adresami pustymi.

```
new(wsb); – utwórz w pamięci pierwszy element lisc, wykorzystując główny...
            wskaźnik edytora.
if stat_kur then – czy zmienna stat_kur ustawiona jest w pozycji true?
begin
     . . .
end else – jeśli nie, to ustawiona jest w pozycji false, czyli...
    begin
         if z1<>nil then – czy istnieje element bieżący?
         begin
              . . .
         end else – jeśli nie, to ten element będzie pierwszy, czyli...
             begin
                  with wsb^ do
                  begin
                       lewa:=nil;
                       prawa:=nil;
                       kur_l:=nil;
```

```
                    kur_p:=nil;
                    kur_g:=nil;
                    kur_d:=nil;
                    lx:=AX; - wczytaj wartość współrzędnej poziomej...
                              ze zmiennej startowej AX
                    ly:=AY; - wczytaj wartość górnej współrzędnej...
                              pionowej ze zmiennej startowej AY
                    py:=ly+Form6.RZ; - wylicz dolną współrzędną...
                           pionową dodając do górnej współrzędnej...
                           wysokość znaku ze zmiennej modułu regulacji
                    y1:=ly;  } na etapie tworzenia nowego elementu,..
                    y2:=py;  } współrzędne rzeczywiste są równe...
                             } współrzędnym poziomu
                    znak:=wzn1 - w pole znak wpisz znak...
                                 wczytany z klawiatury
                end;
                Form3.Szerokosc(wsb);
                wsp:=wsb; - jest to pierwszy element, dlatego jego adres...
                            zostaje dodatkowo zapamiętany w zmiennej wsp
                stat_kur:=true - po wprowadzonym znaku, zmienna...
                 stat_kur ustawiona jest w pozycji true (kursor za znakiem)
            end;
        end;
```

Gdy nowy składnik edytora został utworzony i poprawnie związany z pozostałymi jego składnikami, pozostaje ustawienie zmiennej `stat_kur` w pozycji `true`, by kursor był ustawiony za wprowadzonym znakiem oraz odświeżenie całej struktury `lisc`. Od tej chwili nowy znak staje się bieżącym a składnik edytora, do którego ten znak został wpisany – bieżącym elementem, dostępnym przez wskaźnik `wsb`.

```
stat_kur:=true; - po wprowadzonym znaku, zmienna stat_kur ustawiona jest...
                  w pozycji true (kursor za znakiem)
wsb^.druk:=true; - nowy znak ma być widoczny na ekranie, dlatego wpisz w pole...
                   druk wartość true
wsb^.zazn:=false; - znak nie jest zaznaczony
Form3.Odswiez_funkcje - ułóż ponownie funkcję/pochodną w sposób...
                        matematyczny, uwzględniając nowy znak
```

XIV. 1.2. Czy nowy znak rozbije nazwę 'log'? Funkcja `czy_log`

Zadaniem funkcji jest zbadanie pól `znak`, należących do otoczenia bieżącego elementu `lisc`, czy wpisane w nie znaki nie tworzą funkcji „log". W pierwszej instrukcji wyboru `case` identyfikowany jest znak wpisany w pole `znak` bieżącego elementu, którego adres dostępny jest w parametrze. Jeśli znak ten należy do jednej z liter wspomnianej funkcji, zmiennej lokalnej `l5` wpisana zostaje liczba określająca pozycję tego znaku w nazwie funkcji. Na podstawie tej liczby, identyfikowane są pozostałe znaki wpisane w pola `znak` sąsiadujących elementów `lisc`. Jeśli identyfikacja wypadnie pomyślnie, funkcja zwraca wartość `true`, w przeciwnym przypadku zwraca ona domyślną wartość `false`.

Poza procedurą `Wstaw_znak`, funkcja wykorzystywana jest m.in. przez procedurę zdarzenia od klawiatury, w której wykrycie nazwy funkcji „log" podczas poruszania się kursorem, ustawia zmienną `stat_kur` w pozycji `true`, umożliwiając sprawdzenie jej w procedurze `Wstaw_znak`. Ponadto używana jest przez procedurę `Usun_znak` do wykrycia nazwy tej funkcji matematycznej przed jej usunięciem.

```
function czy_log(const odn5:lisc):boolean;
var n1:lisc;
    l5:byte;
    sa5:integer;
begin
     Result:=false; - domyślna, zwracana wartość przez funkcję
     if odn5=nil then exit; - jeśli dostarczony w parametrze adres jest pusty..
                              opuść funkcję, zwracając wartość false
     sa5:=ord(odn5^.znak); - odczytaj wartość znaku wpisanego w pole znak...
                             elementu otrzymanego w parametrze
     case sa5 of
          108:l5:=1; - gdy wartość znaku odpowiada literze 'l', przypisz zmiennej..
                       l5 liczbę jeden
          111:l5:=2; - gdy wartość znaku odpowiada literze 'o' – liczbę dwa
          103:l5:=3 - gdy wartość znaku odpowiada literze 'g' – liczbę trzy
          else l5:=0 - dla każdego innego znaku, przypisz zmiennej i5 liczbę zero
     end;
     if l5=1 then - czy w zmiennej i5 znajduje się liczba jeden?
     begin - jeśli tak, to...
          n1:=odn5^.kur_p; - do zmiennej n1 wczytaj adres następnika...
                     w wiązaniu kursorowym, względem elementu bieżącego
          if n1<>nil then - czy następny element lisc istnieje?
             if ord(n1^.znak)=111 then - jeśli tak, to czy wpisany w jego...
                                         pole znak jest literą 'o'?
             begin - jeśli tak, to...
                  n1:=n1^.kur_p; - pozyskaj w zmiennej n1 adres...
                                   następnika w wiązaniu kursorowym
                  if n1<>nil then - czy następnik istnieje?
                     if ord(n1^.znak)=103 then Result:=true;
                       jeśli tak, to jeśli wpisany w jego pole znak jest literą 'g',...
                       zwróć do zmiennej Result wartość true
             end;
     end;
     if l5=2 then - czy w zmiennej i5 znajduje się liczba dwa?
     begin - jeśli tak, to...
          n1:=odn5^.kur_l; - do zmiennej n1 wczytaj adres poprzednika...
                     w wiązaniu kursorowym, względem elementu bieżącego
```

```
                if n1<>nil then - czy poprzednik istnieje?
                   if ord(n1^.znak)=108 then - jeśli tak, to czy znak wpisany...
                                                w jego pole jest literą 'l'?
                   begin - jeśli tak, to...
                        n1:=odn5^.kur_p; - do zmiennej n1 wczytaj adres...
                                           następnika w wiązaniu kursorowym,...
                                           względem bieżącego elementu
                        if n1<>nil then - czy następnik istnieje?
                           if ord(n1^.znak)=103 then Result:=true;
                           jeśli tak, to jeśli znak wpisany w jego pole...
                           jest literą 'g', zwróć do zmiennej Result wartość true
                   end;
           end;
           if l5=3 then - czy w zmiennej i5 znajduje się liczba trzy?
           begin - jeśli tak, to...
                n1:=odn5^.kur_l; - do zmiennej n1 wczytaj adres poprzednika...
                          w wiązaniu kursorowym, względem bieżącego elementu
                if n1<>nil then - czy poprzednik istnieje?
                   if ord(n1^.znak)=111 then - jeśli tak, to czy znak wpisany...
                                               w jego pole jest literą 'o'?
                   begin - jeśli tak, to...
                        n1:=n1^.kur_l; - do zmiennej n1 wczytaj adres...
                                         poprzednika w wiązaniu kursorowym
                        if n1<>nil then - czy poprzednik istnieje?
                           if ord(n1^.znak)=108 then Result:=true;
                           jeśli tak, to jeśli znak wpisany w jego pole...
                           to litera 'l', zwróć do zmiennej Result wartość true
                   end;
           end;
    end;
```

XIV. 2. Przebudowanie struktury edytora po wprowadzeniu lub usunięciu znaku – procedura `Odswiez_funkcje`

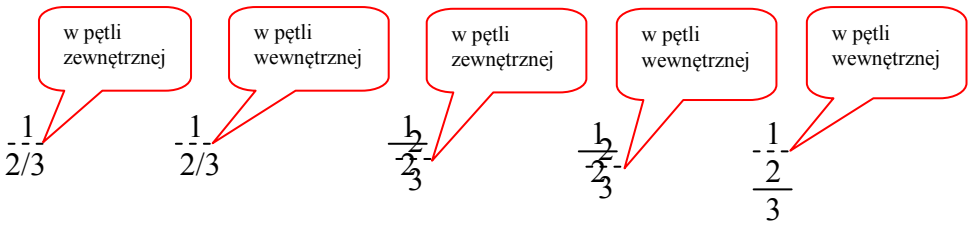
Jest to procedura najważniejsza dla edytora. Dzięki jej wewnętrznym procedurom możliwa jest pełna obsługa znaków akcji i nawiasów. Pojawienie się w funkcji znaków akcji wymaga wielokrotnego wywołania procedur odsługujących te znaki, przez co możliwe jest obsłużenie funkcji składającej się z ułamków piętrowych czy innych przypadków wielokrotnego użycia znaków akcji, co może mieć miejsce przy bardzo rozbudowanych funkcjach. By usprawnić mechanizm odświeżania funkcji, zadeklarowana została struktura, której wykorzystanie zdecydowanie ułatwia wspomniane czynności odświeżania. Jej deklaracja jest następująca:

```
type dzialanie=^pola;
     pola=record
                adres:lisc; - pole przeznaczone na adres elementu lisc...
                              z wpisanym znakiem akcji
                znak:integer; - pole przewidziane na wartość znaku akcji
                poprz:dzialanie - pole służące do związania z poprzednim...
                                  elementem dzialanie
     end;
var a1,a2:lisc;
```

```
p1,pn:dzialanie;
k:integer;
```

Struktura `dzialanie` posiada tylko jedno pole do związania ze swoim poprzednikiem – jest to więc lista jednokierunkowa. Do jej obsługi wystarczą dwa wskaźniki `p1` i `pn`.

Główna część procedury składa się z pętli `while..do`, w której identyfikowane są wszystkie znaki wpisane w pola `znak` elementów `lisc`. Odbywa się to przez przypisanie lokalnej zmiennej adresu pierwszego elementu `lisc` i wyszukiwania znaków akcji, czyli dzielenia, potęgowania, pierwiastkowania i... logarytmu o dowolnej podstawie. Zlokalizowanie logarytmu odbywa się nieco inaczej niż pozostałych znaków akcji, gdyż trzeba zidentyfikować trzy znaki składające się na nazwę tej funkcji, zatem gdy bieżący element `lisc` posiada wpisany znak litery ‘g’, w kolejnych instrukcjach warunkowych szukane są pozostałe znaki tej funkcji, wpisane w pola `znak` kolejnych poprzedników. Rozpoznanie wszystkich trzech znaków „log” powoduje przypisanie zmiennej `k` liczby 95 – liczba ta znajduje się w stałych znaków akcji. W kolejnej instrukcji warunkowej identyfikowany jest znak akcji i jeśli zostanie rozpoznany, w kolejnych instrukcjach pętli przydzielona zostaje pamięć dla nowego elementu `dzialanie`, w którego pole `znak` wpisana zostaje wartość znaku akcji, natomiast w polu `adres` – adres bieżącego elementu `lisc` z wpisanym znakiem akcji. Jeśli istnieje poprzednik nowo utworzonego elementu `dzialanie`, nowy element zostaje z nim związany przez zapamiętanie jego adresu w polu `poprz`. Gdy nowy element `dzialanie` jest już kompletny, na podstawie wartości znaku akcji rozpoznawanej w instrukcji wyboru `case`, wywołana jest właściwa procedura, odpowiednia do rozpoznanego znaku akcji. Po powrocie z procedury, uruchamiana jest wewnętrzna pętla `while..do`, w której na podstawie wartości znaków akcji, zapamiętanych w polach `znak` elementów `dzialanie`, wywoływane są te same procedury lecz w odwrotnej kolejności. Pozwala na to lista jednokierunkowa, której każdy element pamięta tylko adres swego poprzednika. Powtórne wywoływanie tych samych procedur wynika z faktu, że każda z nich układa tylko tę część funkcji, która należy do otoczenia danego znaku akcji. Jeśli np. mianownik ułamka składa się z kolejnego ułamka, bez powtórnego wywołania procedury `dzielenie`, ułamki nakładałyby się na siebie. Pokazuje to rysunek 44.



Rys. 44. Ukazanie faz układania ułamka piętrowego, będącego wynikiem wywołania procedury `dzielenie` dla wskazanych linią przerywaną znaków dzielenia, z zaznaczeniem pętli, w której procedura ta została wywołana

Z każdym krokiem pętli, szukana jest maksymalna wartość współrzędnej poziomej, pochodzącej z pól `px` analizowanych elementów `lisc`. Jej wartość jest każdorazowo aktualizowana w zmiennej `mx2`. Po opuszczeniu pętli, w zmiennej tej będzie znajdowała się maksymalna wartość tej współrzędnej, odnoszącej się do pełnej funkcji. Wykorzystana zostanie do ewentualnego powiększenia szerokości formy, gdyby okazało się, że funkcja nie zmieści się w standardowej jej szerokości.

```
begin - początek procedury
      a1:=wsp; - zmiennej a1 wpisz adres pierwszego elementu lisc
      p1:=nil; - wskaźnikowi p1 wpisz identyfikator adresu pustego
      pn:=p1;
      while a1<>nil do -pętla zewnętrzna. Wykonaj krok pętli,...
      begin              póki zmienna a1 nie ma adresu pustego
            k:=ord(a1^.znak); - odczytaj wartość znaku wpisanego w pole znak...
                              analizowanego elementu lisc
            if k=103 then - czy wartość znaku odpowiada literze 'g'?
            begin - jeśli tak, to...
                  a2:=a1^.kur_l; - pozyskaj adres poprzednika w wiązaniu kursorowym
                  if a2<>nil then - czy poprzednik istnieje?
                     if ord(a2^.znak)=111 then - jeśli tak, to czy wartość znaku...
                                    wpisanego w jego pole znak odpowiada literze 'o'?
                     begin - jeśli tak, to...
                           a2:=a2^.kur_l; - pozyskaj adres poprzednika...
                                          w wiązaniu kursorowym
                           if a2<>nil then - czy poprzednik istnieje?
                              if ord(a2^.znak)=108 then k:=95;
                                 jeśli tak, to jeśli wartość znaku wpisanego...
                                 w jego pole znak odpowiada literze 'l',...
                                 to wpisz do zmiennej k wartość 95
                     end;
            end;
            if k in dzed then - czy wartość w zmiennej k należy do...
                              wartości znaków akcji?
            begin - jeśli tak, to...
                  pn:=p1; - w zmiennej pn zapamiętaj adres dotychczasowego...
                            elementu dzialanie
                  new(p1); - utwórz w pamięci nowy element dzialanie
                  p1^.poprz:=pn; - w polu poprz zapamiętaj adres poprzednika
                  p1^.znak:=k; - w polu znak zapamiętaj wartość znaku akcji
                  p1^.adres:=a1; - w polu adres zapamiętaj adres elementu lisc
                                   aktualnie analizowanego w pętli
                  case k of
                        95:potelog(a1,false); - wywołaj procedurę potelog...
                                              dla logarytmu o dowolnej podstawie
                        47:dzielenie(a1); - wywołaj procedurę dzielenie...
                                          dla znaku dzielenia
                        94:potelog(a1,true); - wywołaj procedurę potelog...
                                             dla znaku potęgowania
                        92:pierwiastek(a1) - wywołaj procedurę pierwiastek...
                                           dla znaku pierwiastkowania
                  end;
                  pn:=p1; - przechowaj we wskaźniku pn adres...
                            ostatniego elementu dzialanie

                  while p1<>nil do -pętla wewnętrzna. wykonaj krok pętli,...
                  begin              póki wskaźnik p1 nie ma adresu pustego
                        nawiasy; - wywołaj procedurę nawiasy
                        k:=p1^.znak; - z pola znak bieżącego elementu...
                                       dzialanie odczytaj wartość znaku akcji
                        a2:=p1^.adres; - z pola adres odczytaj adres elementu lisc
```

```
                    case k of
                         95:potelog(a2,false);
                         47:dzielenie(a2);
                         94:potelog(a2,true);
                         92:pierwiastek(a2)
                    end;
                    p1:=p1^.poprz - pozyskaj adres poprzedniego...
                                    elementu dzialanie
               end;
               p1:=pn - do wskaźnika p1 wczytaj adres ostatniego elementu...
                        dzialanie, zapamiętanego we wskaźniku pn
          end;
          if a1^.px>mx2 then mx2:=a1^.px; - jeśli wartość współrzędnej...
                    poziomej w polu px jest większa od wartości w zmiennej mx2,...
                    wpisz nową wartość współrzędnej do tej zmiennej
          a1:=a1^.prawa - pozyskaj adres następnego elementu lisc
     end;
```

Po opuszczeniu pętli `while..do` a tym samym ukończeniu procesu układania funkcji w sposób matematyczny, pamięć zajęta przez elementy `dzialanie`, musi zostać uwolniona.

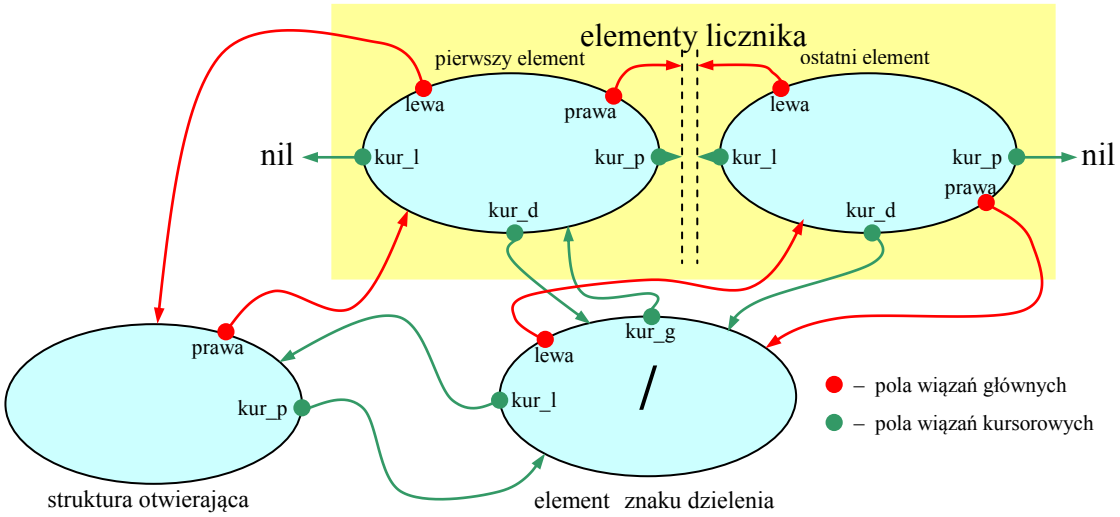
```
     p1:=pn; - wskaźnikowi p1 wpisz adres ostatniego elementu dzialanie
     while p1<>nil do - wykonaj krok pętli, póki wskaźnik p1 nie ma adresu pustego
     begin
          p1:=p1^.poprz; - we wskaźniku p1 pozyskaj adres poprzednika
          dispose(pn); - uwolnij pamięć, zajmowaną przez element,...
                         wskazywany przez wskaźnik pn
          pn:=p1 - wskaźnikowi pn przypisz adres poprzednika
     end;
```

Mając na względzie fakt, że w wyniku analizy wszystkich elementów `lisc`, mógł nie zostać rozpoznany ani jeden znak akcji, a co za tym idzie, mogła zostać pominięta procedura `nawiasy`, tuż przed uwidocznieniem funkcji na ekranie, procedura ta musi być bezwzględnie wywołana, by wszystkie ewentualne znaki nawiasów zostały poprawnie uporządkowane. Poza nawiasami i wstawieniem funkcji na formie przez procedurę `Wyswietl`, sprawdzana jest maksymalna szerokość funkcji. Jeśli okaże się, że funkcja nie zmieści się w standardowej szerokości formy, jej szerokość zostaje powiększona względem sumy maksymalnej współrzędnej poziomej funkcji zapamiętanej w zmiennej `mx2` i naddatku zawartego w stałej `AXX`.

```
     if wsp<>nil then - czy wskaźnik na pierwszy element lisc nie jest pusty?
     begin - jeśli nie jest pusty, to...
          nawiasy; - wywołaj procedurę nawiasy
          if (mx2+AXX)>SZER2 then Form2.Width:=mx2+AXX; - jeśli suma...
                         wartości w zmiennej mx2 i stałej AXX jest większa od...
                         szerokości formy, zapamiętanej w zmiennej SZER2,...
                         ustaw szerokość formy względem tej sumy
          Form2.Wyswietl - wstaw funkcję na formie
     end;
end; - koniec procedury
```

XIV. 2.1. Układanie ułamka – procedura `dzielenie`

Zadaniem procedury jest ułożenie ułamka, którego znak dzielenia znajduje się w elemencie `lisc` dostępnym w procedurze w postaci parametru o nazwie `akcja`. Parametr ten zabezpieczony został przed przypadkową modyfikacją, tzn. przed przypisaniem mu adresu innego elementu `lisc`, poprzedzając jego nazwę słowem kluczowym `const`. Procedura rozpoczyna swą pracę od sprawdzenia, czy licznik ułamka został już wyznaczony podczas możliwych, wcześniejszych wywołań procedury. Sprawdzenie to polega na porównaniu adresów wpisanych w pola `lewa` oraz `kur_l` elementu wskazywanemu przez parametr `akcja` – jeśli w pola te nie wpisano adresów pustych oraz gdy adresy te są takie same, oznacza to, że licznik ułamka nie został jeszcze wyznaczony. Czynności związane z wydzieleniem licznika oraz określeniem struktury otwierającej przedstawia rysunek 45.



Rys. 45. Struktura logiczna wiązania elementów licznika ułamka i struktury otwierającej z elementem znaku dzielenia

Pierwszą czynnością związaną z wyznaczaniem licznika jest wyszukanie elementu granicznego, czyli możliwego, pierwszego elementu licznika. Polega to na wyszukaniu spośród kolejnych poprzedników w wiązaniu kursorowym takiego elementu, który nie ma swego poprzednika w wiązaniu kursorowym. Wyszukiwanie rozpoczyna się od elementu ze znakiem dzielenia i przeprowadzone jest w pętli `while..do`. Znaleziony element będzie wykorzystany w warunku opuszczenia kolejnej pętli wyznaczającej licznik.

```
procedure dzielenie(const akcja:lisc);
var ad1,ad2,ad3,ad4,apl:lisc;
    n1,k1,l1,m1,kr1:integer;
brak,licz:boolean;ad2:=nil;

     function srodek –fragment nagłówka funkcji wewnętrznej

begin
     ad2:=nil; – zainicjuj wskaźnik ad2, przeznaczony na adres…
                 pierwszego elementu licznika adresem pustym
     ad3:=akcja; – przypisz wskaźnikowi ad3 adres elementu ze znakiem dzielenia
     kr1:=ad3^.lx; – zapamiętaj pozycję współrzędnej poziomej elementu znaku dzielenia
```

```
ad1:=ad3^.lewa; - wskaźnikowi ad1 przypisz adres poprzednika elementu…
                  znaku dzielenia w wiązaniu głównym
ad4:=ad3^.kur_l; - wskaźnikowi ad4 przypisz adres poprzednika elementu...
                   znaku dzielenia w wiązaniu kursorowym
brak:=false;
if ad4<>nil then - czy istnieje poprzednik elementu znaku dzielenia...
                   w wiązaniu kursorowym?
   if ad4=ad1 then - jeśli tak, to czy adresy elementów w wiązaniu kursorowym...
                     i głównym są takie same?
   begin - jeśli tak, to obydwa wskaźniki wskazują na ten sam element, co oznacza,...
           że licznik nie został jeszcze wyznaczony, więc…
        ad4:=nil; - przypisz wskaźnikowi ad4 identyfikator adresu pustego
        ad2:=ad1; - zachowaj adres poprzednika w osobnym wskaźniku
        while ad2<>nil do - wykonaj krok pętli, póki wskaźnik ad2...
                            nie ma adresu pustego
        begin
             ad4:=ad2; - we wskaźniku ad4 zachowaj adres zawarty...
                         we wskaźniku ad2
             ad2:=ad2^.kur_l - we wskaźniku ad2 pozyskaj adres...
                               poprzednika w wiązaniu kursorowym
        end;
```

Po wyznaczeniu elementu granicznego uruchamiana jest pętla `repeat..until`, w której wyznaczony będzie licznik ułamka. Rozpoczęcie każdego kroku pętli rozpoczyna się do pozyskania adresu poprzedniego elementu analizowanego w pętli w wiązaniu głównym oraz sczytania znaku wpisanego w jego pole `znak`, zapamiętując jego wartość w zmiennej liczbowej `k1`. Jeśli jego adres nie jest równy adresowi elementu granicznego, pętla jest kontynuowana. Z każdym krokiem pętli zliczane są nawiasy w ten sposób, że jeśli sczytany znak należy do znaków nawiasów zamykających, licznik nawiasów, czyli zmienna `n1`, zwiększa swoją wartość o jeden, natomiast sczytanie znaku należącego do nawiasów otwierających, zmniejsza jej wartość o jeden. Dla liczby nawiasów mniejszej od zera, pętla jest bezwzględnie przerywana, gdyż wykryty nawias otwierający musi znaleźć się przed kreską ułamkową. Gdy liczba nawiasów jest równa zero oraz gdy sczytany znak należy do znaków działania, czyli dodawania, odejmowania lub mnożenia, pętla jest przerwana instrukcją `break`, gdyż znak tego typu powinien znaleźć się przed kreską ułamkową. W tym miejscu warto zaznaczyć jeden ważny aspekt przerwania pętli – może się zdarzyć, że znak działania lub nawias otwierający, został wpisany w pole `znak` struktury otwierającej, należącej do znaku potęgowania lub kolejnego znaku dzielenia, znajdującego się w liczniku układanego ułamka. Przeniesienie tego znaku przed kreskę ułamkową, a co za tym idzie, zakwalifikowania tego elementu jako swojej struktury otwierającej, doprowadziłoby do rozbicia otoczenia znaku akcji znajdującego się w liczniku. By zachować otoczenie znaku akcji w niezmienionym stanie, tworzona jest w pamięci dodatkowa struktura otwierająca dla tego znaku, która w swym polu `znak` będzie miała wpisany znak spacji. Kolejnym aspektem wartym uwagi jest wczytanie do zmiennej sterującej pętlą, czyli do wskaźnika `ad1`, adresu poprzedniego elementu w wiązaniu kursorowym, pod warunkiem, że poprzednik ten należy do znaków akcji, czyli dzielenia, potęgowania, pierwiastkowania bądź też ma wpisany znak litery ‘g’. Taki przeskok analizowanych elementów w pętli pozwoli na pominięcie niekompletnego otoczenia znaku akcji, pozwalając na zachowanie tych elementów w wyznaczanym liczniku.

Jeśli w wyniku powyższych instrukcji pętla nie została przerwana, do wskaźnika `ad2` wczytany zostaje adres elementu aktualnie analizowanego w pętli – jest to pewny element licznika. Jednocześnie, w zmiennej liczbowej `kr1` zapamiętana zostaje najmniejsza wartość

współrzędnej poziomej należącej do znaku licznika z jego lewej strony. Wartość ta będzie wykorzystana do zwymiarowania ułamka.

```
if ad4<>nil then - czy poprzednik elementu znaku dzielenia, w wiązaniu kursorowym,...
                   nie jest pusty?
begin - jeśli nie, to...
     n1:=0; - wyzeruj licznik nawiasów
     ad1:=ad3; - zainicjuj zmienną sterującą pętlą adresem elementu ze znakiem dzielenia
     repeat
          ad1:=ad1^.lewa; - do wskaźnika ad1 wczytaj adres poprzednika...
                            w wiązaniu głównym elementu analizowanego w pętli
          k1:=ord(ad1^.znak); - zapamiętaj w zmiennej k1 wartość znaku …
                                zawartego w polu znak elementu aktualnie analizowanego
          if k1 in zam then inc(n1); - jeśli wartość w zmiennej k1…
                                       odpowiada wartościom nawiasów zamykających,...
                                       zwiększ wartość w zmiennej n1 o jeden
          if k1 in otw then dec(n1); - jeśli wartość w zmiennej k1…
                                       odpowiada wartościom nawiasów otwierających,...
                                       zmniejsz wartość w zmiennej n1 o jeden
          if n1<=0 then - czy wartość w zmiennej n1 jest mniejsza lub równa zero?
          begin - jeśli tak, to...
               if n1<0 then break; - jeśli wartość w zmiennej n1 jest…
                                     mniejsza od zera, opuść pętlę
               if k1 in dzia then break; - gdy analizowany w pętli
                                           element ma wpisany znak działania '*', '+', '– '…
                                           opuść pętlę
               apl:=ad1^.kur_l; - jeśli pętla nie została opuszczona, wczytaj do...
                                  wskaźnika apl adres poprzednika w wiązaniu…
                                  kursorowym elementu aktualnie analizowanego
               if apl<>nil then - czy poprzednik ten istnieje?
               begin - jeśli tak, to...
                    if ord(apl^.znak) in dzed then ad1:=apl
                    else if apl^.znak='g' then ad1:=apl –jeśli...
                        znak wpisany w pole znak poprzednika, należy do.znaków..
                        akcji lub ma wpisany znak litery 'g', przypisz zmiennej...
                        sterującej pętlą – wskaźnikowi ad1 adres jego poprzednika
               end;
               if (ad1^.znak='^') or (ad1^.znak='/') then - czy ...
                bieżący element pętli ma wpisany znak potęgowania lub dzielenia?
                (tylko te znaki akcji posiadają strukturę otwierającą)
               begin - jeśli tak, to...
                    apl:=ad1^.kur_l; - do wskaźnika apl wczytaj adres...
                                       poprzednika w wiązaniu kursorowym…
                                       elementu wskazywanemu przez…
                                       wskaźnik ad1
                    if ord(apl^.znak) in dzia then brak:=true;
                    jeśli w polu znak poprzednika znajduje się znak działania,...
                    ustaw zmienną brak w pozycji true
                    if ord(apl^.znak) in otw then brak:=true;
                    jeśli w pole znak wpisano znak nawiasu otwierającego,...
                    ustaw zmienną brak w pozycji true
                    if brak then - czy zmienna brak została ustawiona...
                                   w pozycji true?
```

```
                        begin - jeśli tak, to...
                              new(ad1); - utwórz w pamięci nową strukturę...
                                          otwierającą dla elementu posiadającego…
                                          wpisany znak akcji,…
                                          by nawias otwierający i …
                                          znak działania znalazły się przed…
                                          kreską ułamkową
                              with ad1^ do - związ nową strukturę…
                                             otwierającą tak,…
                                             by z prawej jej strony znalazł się element
                                             z wpisanym znakiem akcji a z lewej,…
                                             jego dotychczasowa struktura otwierająca..
                                             wskazywana przez wskaźnik ap1…
                              begin
                                   lewa:=apl;
                                   prawa:=apl^.prawa;
                                   kur_l:=nil;
                                   kur_p:=apl^.kur_p;
                                   kur_g:=apl^.kur_g;
                                   kur_d:=nil;
                                   lx:=apl^.px;
                                   px:=lx;
                                   ly:=apl^.ly;
                                   py:=apl^.py;
                                   y1:=ly;
                                   y2:=py;
                                   znak:=' ';
                                   druk:=true
                                   zazn:=false
                              end;
                              apl^.prawa:=ad1;
                              apl^.kur_p:=nil;
                              apl^.kur_g:=nil;
                              apl:=ad1^.prawa;
                              apl^.lewa:=ad1;
                              apl:=ad1^.kur_p;
                              apl^.kur_l:=ad1;
                              brak:=false;
                        end;
                    end;
               end;
               ad2:=ad1; - we wskaźniku ad2 zapamięuj adres bieżącego elementu...
                           analizowanego w pętli (pewny element licznika)
               if kr1>ad2^.lx then kr1:=ad2^.lx; - w zmiennej kr1...
                                   zapamiętaj minimalną wartość współrzędnej...
                                   poziomej, usytuowanej z lewej strony znaku
          until ad1=ad4; - opuść pętlę, gdy wskaźnik ad1 uzyska adres taki,...
                           jaki zawarty jest we wskaźniku ad4 (element graniczny licznika)
     end;
```

Po opuszczeniu pętli `repeat..until`, wskaźnik `ad2` powinien wskazywać na pierwszy element licznika. Może się jednak zdarzyć, że pętla zostanie opuszczona wcześniej, nim wskaźnik ten zdoła przyjąć adres pierwszego elementu analizowanego w pętli, czyli od wskaźnika `ad1`. Jeśli tak się stanie, brakujący i jedyny element licznika musi zostać utworzony, by otoczenie ułamka było kompletne. Podobnie, gdy wskaźnik `ad1` będzie zawierał adres pusty bądź element dostępny przez ten wskaźnik nie będzie związany wiązaniem kursorowym z pierwszym elementem licznika – w obu przypadkach zmienna logiczna `brak` zostaje usta-

wiona w pozycji `true`. Wykrycie tej wartości w instrukcji warunkowej spowoduje utworzenie struktury otwierającej dla składanego ułamka. Jeśli struktura otwierająca istnieje, można ją związać wiązaniem kursorowym z elementem znaku dzielenia, zgodnie ze schematem zamieszczonym na rysunku 45.

```
if ad2<>nil then – czy licznik ułamka został wyznaczony?
begin – jeśli tak, to...
     if ad1=ad2 then ad1:=ad1^.lewa; – gdy wskaźniki ad1 i ad2 wskazują...
                    na ten sam element, pozyskaj we wskaźniku ad1 adres poprzednika...
                    w wiązaniu głównym

     if ad1=nil then brak:=true – jeśli wskaźnik ad1 posiada adres pusty, to...
                                  ustaw zmienną logiczną brak w pozycji true

     else – w przeciwnym przypadku...
         if ad1^.kur_p<>ad2 then brak:=true – jeśli element wskazywany...
                        przez wskaźnik ad1 nie jest związany wiązaniem kursorowym...
                        z pierwszym elementem licznika, element ten nie należy do...
                        tego ułamka, więc ustaw zmienną logiczną brak w pozycji true

         else – w przeciwnym przypadku...
         begin
                                                  { związż wzajemnie wiązaniem kursorowym...
               ad3^.kur_l:=ad1;                   { element – strukturę otwierającą tego ułamka,...
               ad1^.kur_p:=ad3;                   { dostępną przez wskaźnik ad1 z elementem...
                                                  { znaku dzielenia, dostępnym przez wskaźnik ad3
         end
end else – jeśli licznik ułamka nie został wyznaczony, to...
    if ad1<>nil then – czy wskaźnik ad1 nie ma adresu pustego?
       if ad1^.kur_p<>ad3 then brak:=true – jeśli nie, to gdy nie jest związany...
            kursorowo z elementem znaku dzielenia, ustaw zmienną brak w pozycji true
```

Jeśli adresy zawarte we wskaźnikach `ad1` i `ad4`, jeszcze przed uruchomieniem pętli `repeat..until` różnią się, oznacza to, że licznik został już wyznaczony a struktura otwierająca dostępna przez wskaźnika `ad4` została związana z pozostałymi elementami ułamka, co mogło mieć miejsce podczas wcześniejszych wywołań procedury. W takiej sytuacji procedura musi czuwać nad ewentualnym dopisaniem do licznika znaku działania, czyli dodawania lub odejmowania. Jeśli taki znak został przez użytkownika dopisany a licznik nie posiada wymaganego nawiasu otwierającego na jego początku, nowy element `lisc` z wpisanym znakiem nawiasu zostaje utworzony i związany tak, że staje się pierwszym elementem licznika.

```
. . .
ad1:=ad3^.lewa; – wskaźnikowi ad1 przypisz adres poprzednika w wiązaniu głównym…
                  elementu znaku dzielenia

ad4:=ad3^.kur_l; – wskaźnikowi ad4 przypisz adres poprzednika w wiązaniu…
                   kursorowym elementu znaku dzielenia
. . .
if ad4<>nil then – czy istnieje poprzednik elementu znaku dzielenia...
                   w wiązaniu kursorowym?

   if ad4=ad1 then – jeśli tak, to czy adresy elementów w wiązaniu kursorowym...
                     i głównym są takie same?

   begin
        . . . (pętla repeat..until wyznaczająca licznik)

   end else – jeśli adresy nie są takie same, to...

   begin
```

```
if ord(ad4^.znak) in dzed then brak:=true; - gdy znak ...
      wpisany w pole znak elementu wskazywanego przez wskaźnik ad4,...
      należy do znaków akcji, ustaw zmienną brak w pozycji true

while ad1<>nil do - wykonaj krok pętli, póki wskaźnik ad1...
                    nie ma adresu pustego
begin
     if ad1=ad4 then break; - jeżeli wskaźnik ad1 przyjmie adres...
          struktury otwierającej, dostępnej we wskaźniku ad4, opuść pętlę

      ad2:=ad1; - we wskaźniku ad2 zapamiętuj aktualny i pewny...
                  adres elementu licznika

     if kr1>ad2^.lx then kr1:=ad2^.lx; - jeśli dotychczasowa…
                                         wartość w zmiennej kr jest większa niż ta,…
                                         zawarta w polu lx analizowanego w pętli…
                                         elementu licznika, wpisz do zmiennej kr…
                                         wartość z tego pola

     ad1:=ad1^.lewa - pozyskaj adres poprzednika w wiązaniu głównym…
                      elementu analizowanego w pętli
end;
if ord(ad2^.znak) in otw then licz:=true else licz:=false;
                    jeśli pierwszym znakiem licznika jest nawias otwierający,...
                    ustaw zmienną logiczną licz w pozycji true,...
                    w przeciwnym przypadku ustaw ją w pozycji false

if not licz then
begin
     ad4:=ad2; - zachowaj adres pierwszego elementu licznika w zmiennej ad4
     n1:=0; - wyzeruj licznik nawiasów
     repeat
         if ord(ad4^.znak) in otw then inc(n1); - jeśli w pole..
                                                  znak, analizowanego elementu, wpisano...
                                                  znak nawiasu otwierającego, zwiększ ...
                                                  wartość w zmiennej n1 o jeden

         if ord(ad4^.znak) in zam then dec(n1); - jeśli w pole..
                                                  znak wpisano znak nawiasu zamykającego,...
                                                  zmniejsz wartość w zmiennej n1 o jeden

         if n1<=0 then - czy liczba nawiasów jest mniejsza...
                         lub równa zero?

            if ord(ad4^.znak) in dzia then - jeśli tak, to czy..
                znak wpisany w pole znak należy do znaków działania?

            begin - jeśli tak, to...
                licz:=true; - ustaw zmienną licz w pozycji true
                break - opuść pętlę
            end;
            ad4:=ad4^.prawa - pozyskaj adres następnika...
                              w wiązaniu głównym…
                              elementu analizowanego w pętli

     until ad4=ad3; - opuść pętlę, gdy wskaźnik ad4 uzyska adres...
                      elementu znaku dzielenia

     if licz then - czy zmienna licz została ustawiona w pozycji true?
     begin - jeśli tak, to...
         ad4:=ad2^.lewa; - do zmiennej ad4 wpisz adres...
                           struktury otwierającej
         new(apl); - utwórz nowy element lisc – pierwszy element licznika
         with apl^ do
        begin
```

```
                    lewa:=ad4;  } związ wzajemnie wiązaniem głównym pierwszy…
                    prawa:=ad2; } element licznika i strukturę otwierającą
                    kur_l:=nil;
                    kur_p:=ad2; - zapamiętaj w polu kur_p adres…
                                  następnego elementu licznika
                    kur_g:=nil;
                    kur_d:=ad3; - zapamiętaj w polu kur_d adres elementu…
                                  znaku dzielenia
                    lx:=ad2^.lx; - do pola lx wpisz wartość współrzędnej…
                                   poziomej dotychczasowego, pierwszego…
                                   elementu licznika
                    px:=lx; do wyliczenia szerokości znaku nawiasu przez …
                            procedurę Szerokosc, współrzędna pozioma…
                            z prawej strony znaku musi być równa współrzędnej…
                            z lewej strony
                    ly:=ad2^.ly; } przepisz wartości współrzędnych pionowych …
                    py:=ad2^.py; } poziomu od swego następnika
                    y1:=ly; } współrzędne pionowe rzeczywiste są równe …
                    y2:=py; } współrzędnym poziomu
                    znak:='('; - wpisz znak nawiasu otwierającego w pole znak
                    druk:=true - znak ma być widoczny
                    zazn:=false - znak nie jest zaznaczony
                end;
                Form3.Szerokosc(apl); - wylicz szerokość nawiasu,...
                                        modyfikując pole px nowego elementu
                Form3.Przesun(apl,false); - przesuń pozostałe znaki licznika...
                                            w prawo o szerokość nowego elementu
                ad2^.lewa:=apl;
                ad2^.kur_l:=apl;
                if ad4<>nil then ad4^.prawa:=apl;
                ad2:=apl - przypisz wskaźnikowi ad2 adres nowego elementu
            end;
        end;
    end
else brak:=true; - jeśli nie istnieje poprzednik elementu znaku dzielenia w wiązaniu...
                   kursorowym, tzn. gdy wskaźnik ad4 posiada adres pusty, ustaw...
                   zmienną brak w pozycji true
```

Wykrycie w zmiennej logicznej `brak` wartości `true` powoduje utworzenie struktury otwierającej we wskaźniku `ad1`, przedtem adres elementu wskazywanego przez ten wskaźnik zostaje zachowany we wskaźniku `ad4`, by podczas wiązania nowo utworzonego składnika z pozostałymi elementami edytora zachować ciągłość listy. Jeśli wskaźnik `ad4` został zainicjowany adresem pustym, oznacza to, że utworzona struktura otwierająca jest pierwszym składnikiem listy, dlatego niezbędne jest zachowanie tego adresu w głównym wskaźniku edytora `wsp`. Gdy element spoza ułamka istnieje a znak zawarty w jego polu `znak` znajduje się na tym samym poziomie, co znak dzielenia wpisany w elemencie znaku dzielenia, zostaje on związany ze strukturą otwierającą także wiązaniem kursorowym, umożliwiając bezpośrednie poruszanie się kursorem po znakach funkcji znajdujących się obok siebie.

```
if brak then - czy zmienna logiczna brak została ustawiona w pozycji true?
begin - jeśli tak, to...
     ad4:=ad1; - zachowaj w zmiennej ad4 adres elementu spoza tego ułamka
     new(ad1); - utwórz w pamięci element lisc (struktura otwierająca)
```

```
        if ad4=nil then wsp:=ad1 - jeśli element spoza ułamka nie istnieje, zachowaj…
                    adres nowo utworzonej struktury otwierającej we wskaźniku edytora wsp
        else - w przeciwnym przypadku
        begin
             ad4^.prawa:=ad1; - zwiąż prawostronnie wiązaniem głównym element…
                                spoza ułamka (ad4) ze strukturą otwierającą (ad1)
           if ad4^.ly=ad3^.ly then licz:=true - jeśli współrzędne pionowe…
                poziomu element spoza ułamka i element znaku dzielenia są sobie równe,…
                zaznacz potrzebę związania kursorowego elementu spoza ułamka…
                ze strukturą otwierającą
        end;
        ad3^.kur_l:=ad1;
        with ad1^ do
        begin
             prawa:=ad2; - w pole prawa wpisz adres pierwszego elementu licznika
             lewa:=ad4; - w pole lewa wpisz adres elementu spoza otoczenia ułamka
             kur_g:=nil;
             kur_d:=nil;
             if licz then czy zmienna licz została ustawiona w pozycji true?
             begin jeśli tak, to…
                  ad4^.kur_p:=ad1; } związ wzajemnie wiązaniem kursorowym…
                  kur_l:=ad4       } element spoza ułamka ze strukturą otwierającą
             end else kur_l:=nil; - w przeciwnym przypadku zainicjuj pole kur_l…
                                    struktury otwierającej adresem pustym
             kur_p:=ad3; - w pole kur_p wpisz adres element znaku dzielenia
             lx:=kr1; - wpisz w pole lx wartość współrzędnej poziomej…
                        pochodzącej od współrzędnej poziomej najbardziej…
                        wysuniętego w lewo elementu licznika i zapamiętanej…
                        w zmiennej kr1
             ly:=ad3^.ly; } pozyskaj wartości współrzędnych pionowych poziomu…
            py:=ad3^.py   } z pól ly i py elementu znaku dzielenia
             px:=lx; - (końcowa współrzędna pozioma jest taka sama...
                        jak początkowa, ponieważ szerokość znaku spacji...
                        wpisana w pole znak ma być równa zero)
             y1:=ly;
             y2:=py;
             znak:=' ';
             druk:=true
             zazn:=false;
        end;
   end;
```

Obecność adresu elementu `lisc` we wskaźniku `ad2` tuż po opuszczeniu pętli wyznaczającej licznik wskazuje na poprawne wyznaczenie licznika. W takiej sytuacji licznik należy zamknąć z lewej i prawej strony. Polega to na wpisaniu adresów pustych w pola `kur_l` pierwszego elementu i `kur_p` ostatniego elementu licznika. Jeśli licznik zaopatrzony został w skrajną parę nawiasów, za pośrednictwem procedury `ukryj_nawiasy`, nawiasy te zostają ukryte. Pozostaje jeszcze związanie elementów licznika z elementem znaku dzielenia tak, by możliwe było poruszanie się za pomocą klawiszy kierunkowych klawiatury pomiędzy kreską ułamkową a elementami licznika. Wiązanie to odbywa się w pętli `reeat..until`, w której każdy element licznika jest sprawdzany przez funkcję `wiazanie`, która zwraca jednoznaczną odpowiedź logiczną, zezwalającą lub nie na wpisanie w pole `kur_d` kolejnego elementu licznika adresu elementu znaku dzielenia. Z kolei wpisanie adresu elementu licznika

w pole kur_g elementu znaku dzielenia polega na takim wyborze elementu, który w swym polu kur_d zawiera już adres elementu znaku dzielenia oraz znak ten nie jest ukryty. Gdy wybór zostanie dokonany, zmienna logiczna brak zostaje ustawiona w pozycji true, co uchroni przed modyfikacją pola kur_g adresem innego elementu. Z każdym krokiem pętli, szukana jest maksymalna współrzędna pionowa, pochodząca z pól y2 elementów licznika. Wartość ta, zapamiętana zostaje w zmiennej l1, która będzie wykorzystana podczas wymiarowania ułamka.

```
if ad2<>nil then - czy wskaźnik ad2 nie ma adresu pustego?
begin - jeśli nie, to wskaźnik wskazuje na pierwszy element licznika, czyli...
     ad3^.lx:=ad1^.px; ustaw współrzędną poziomą z lewej strony kreski ułamkowej...
                na podstawie współrzędnej poziomej z prawej strony struktury otwierającej
     ad2^.lewa:=ad1; - związ wiązaniem głównym pierwszy element licznika...
                       ze strukturą otwierającą
     ad2^.kur_l:=nil; - zamknij pierwszy elementu licznika z lewej strony
     ad4:=ad3^.lewa; - przypisz wskaźnikowi ad4 adres ostatniego elementu licznika
     ad4^.kur_p:=nil; - zamknij ostatni element licznika z prawej strony
     ukryj_nawiasy(ad2,ad4); - ukryj skrajną parę nawiasów
     ad4:=ad2; - przypisz wskaźnikowi ad4 adres pierwszego elementu licznika
     brak:=false; - ustaw zmienną logiczną brak w pozycji false
     l1:=0; - przypisz zmiennej l1 wartość początkową równą zero
     repeat
          if not wiazanie(ad4,ad3,false) then ad4^.kur_d:=ad3;
          jeśli funkcja wiazanie zwróci wartość false,...
          wpisz w pole kur_d elementu licznika wskazanego przez wskaźnik ad4...
          adres elementu znaku dzielenia
          if not brak then - czy zmienna brak jest ustawiona w pozycji false?
             if ad4^.druk then - jeśli tak, to czy znak zawarty w elemencie...
                         wskazywanym przez wskaźnik ad4 nie jest ukryty?
                         (nie wolno wiązać elementów ukrytych z kreską ułamkową)
               if ad4^.kur_d=ad3 then - jeśli znak nie jest ukryty, to czy...
                           ten element licznika zawiera w swym polu kur_d...
                           adres elementu znaku dzielenia?
                begin - jeśli tak, to...
                     ad3^.kur_g:=ad4; - wpisz w pole kur_g elementu...
                                        znaku dzielenia adres elementu licznika
                     brak:=true - ustaw zmienną brak w pozycji true
                end;
                n1:=ad4^.y2; - zmiennej n1 wpisz wartość współrzędnej pionowej...
                               z pola y2 elementu licznika analizowanego w pętli
                if l1<n1 then l1:=n1; - jeśli wartość w zmiennej l1 jest...
                                        mniejsza od wartości w zmiennej n1, do...
                                        zmiennej l1 wpisz wartość ze zmiennej n1
                ad4:=ad4^.prawa - pozyskaj adres następnika w wiązaniu głównym
     until ad4=ad3; - opuść pętlę, gdy wskaźnik ad4 uzyska adres...
                      elementu znaku dzielenia
end else
```

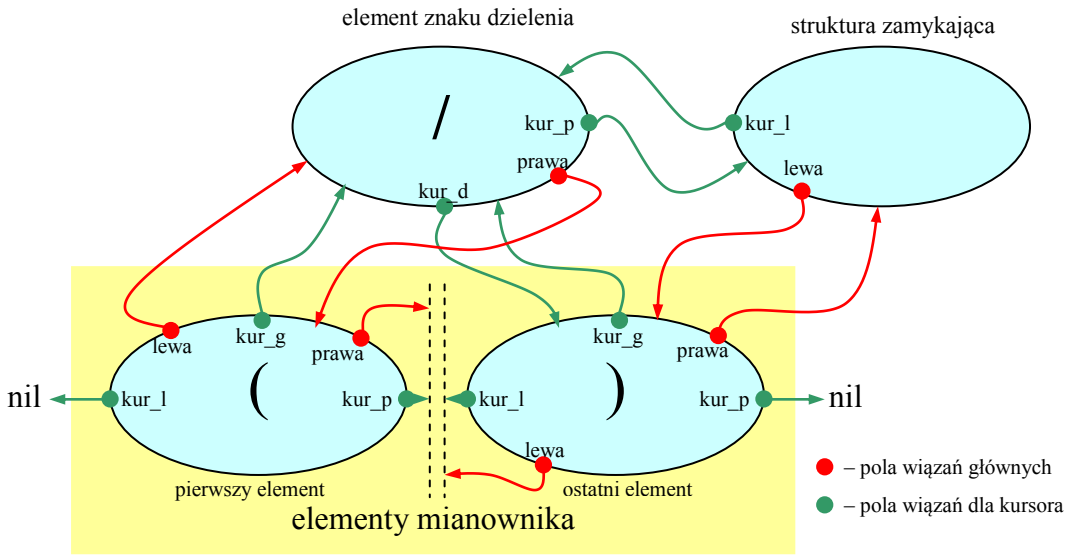
Po opuszczeniu pętli, w zmiennej l1 będzie znajdowała się wartość największej współrzędnej pionowej, która odpowiada najniżej położonemu znakowi licznika na ekranie.

Gdy wskaźnik ad2 będzie miał wpisany adres pusty, jedyny element licznika z wpisanym znakiem cyfry jeden, musi zostać utworzony.

```
if ad2<>nil then - czy wskaźnik ad2 nie ma adresu pustego?
begin
     . . .
end else - jeśli ma adres pusty, to...
    begin
         new(ad2); - utwórz w pamięci element lisc (jedyny element licznika)
         ad1^.prawa:=ad2; - wpisz w pole prawa struktury otwierającej adres…
                            utworzonego elementu licznika
         ad3^.lewa:=ad2; - wpisz w pole lewa elementu znaku dzielenia adres …
                           utworzonego elementu licznika
         ad3^.kur_g:=ad2; - w pole kur_g elementu znaku dzielenia wpisz adres…
                            utworzonego elementu licznika
         with ad2^ do
         begin
              . . .
              lx:=ad3^.lx;   } przypisz współrzędne poziome i pionowe...
              ly:=ad3^.ly;   } nowemu elementowi, na podstawie...
              py:=ad3^.py;   } współrzędnych elementu znaku dzielenia
              y1:=ly;
              y2:=py;
              znak:='1';
              . . .
         end;
         Form2.Szerokosc(ad2); - wylicz szerokość znaku, modyfikując pole px...
                                 nowego elementu
         l1:=ad2^.y2; - zmiennej l1 przypisz wartość współrzędnej pionowej...
                        jedynego elementu licznika
    end;
```

Po wyznaczeniu licznika i struktury otwierającej, pora na mianownik i strukturę zamykającą. Nim procedura rozpocznie wyznaczanie mianownika, adres pierwszego elementu licznika zostaje zapamiętany we wskaźniku `apl`, by dotychczasowy wskaźnik `ad2` można było ponownie wykorzystać.

Wyznaczanie mianownika rozpoczyna się od zbadania następników elementu znaku dzielenia w wiązaniu głównym i kursorowym. Jeśli adresy obu następników są takie same, oznacza to, że następnikiem w obu wiązaniach jest ten sam element, a to oznacza, że mianownik nie został jeszcze wyznaczony. Kolejnym warunkiem zezwalającym na wyznaczanie mianownika jest obecność znaku nawiasu otwierającego wpisanego w pole `znak` składnika związanego z elementem znaku dzielenia z jego prawej strony. Brak tego znaku w następniku dyskwalifikuje go jako pierwszy element mianownika, dlatego musi on zostać utworzony. Sposób wiązania elementów mianownika i struktury zamykającej z elementem znaku dzielenia przedstawia poniższy rysunek 46.



Rys. 46. Struktura logiczna wiązania elementów mianownika ułamka i struktury zamykającej z elementem znaku dzielenia

Wyznaczenie mianownika poprzedza znalezienie struktury granicznej. Polega to, podobnie jak w liczniku, na przeszukiwaniu w pętli `while..do` kolejnych następników w wiązaniu kursorowym i znalezieniu takiego elementu, który nie ma swego następnika w wiązaniu kursorowym. Adres tego elementu będzie jednym z warunków opuszczenia pętli wyznaczającej mianownik.

```
apl:=ad2; - zachowaj adres pierwszego elementu licznika we wskaźnikuj apl
ad1:=ad3^.prawa; - do wskaźnika ad1 wpisz adres następnika w wiązaniu głównym..
                   elementu znaku dzielenia

ad4:=ad3^.kur_p; - do wskaźnika ad4 wpisz adres następnika w wiązaniu kursorowym…
                   elementu znaku dzielenia

ad2:=nil; - zainicjuj wskaźnik ad2 identyfikatorem adresu pustego
brak:=false; - zainicjuj zmienną logiczną, informującą o istnieniu elementu za ułamkiem,...
               wartością false

licz:=false; - zmienną logiczną informującą o braku struktury zamykającej,...
               ustaw w pozycji false

n1:=0; - wyzeruj licznik nawiasów
if ad4<>nil then - czy następnik w wiązaniu kursorowym istnieje?
   if ad4=ad1 then - jeśli tak, to czy adresy następników: w wiązaniu głównym...
                     i kursorowym, są takie same?

      if ord(ad1^.znak) in otw then - jeśli tak, to czy znak wpisany w pole...
                                      znak następnika, należy do znaków nawiasów otwierających?

      begin - jeśli tak, to...
            ad4:=nil; - zainicjuj wskaźnik ad4 identyfikatorem adresu pustego
            ad2:=ad1; - zachowaj adres następnika we wskaźniku ad2
            while ad2<>nil do - wykonaj krok pętli, póki wskaźnik ad2...
                                nie ma adresu pustego
            begin
                  ad4:=ad2; - zachowaj adres kolejnego elementu…
                              we wskaźniku ad4

                  ad2:=ad2^.kur_p - we wskaźniku ad2 pozyskaj adres następnika...
                                    w wiązaniu kursorowym
```

```
            end;
```

Po wyznaczeniu struktury granicznej i zapamiętaniu jej we wskaźniku ad4, uruchamiana jest pętla repeat..until, w której odbywa się wyznaczanie mianownika. W pętli zliczane są nawiasy, jednak w odwróconym warunku ich rozpoznawania niż miało to miejsce w pętli wyznaczającej licznik. Oznacza to, że w sytuacji rozpoznania w analizowanym elemencie znaku nawiasu otwierającego, wartość w zmiennej n1 jest zwiększana o jeden; natomiast po rozpoznaniu znaku nawiasu zamykającego, wartość ta jest zmniejszana o jeden. Rozpoznanie znaku akcji (dzielenia, potęgowania, pierwiastkowania), powoduje przypisanie zmiennej sterującej pętlą (ad1) adresu następnika w wiązaniu kursorowym, co pozwala na pominięcie elementów należących do otoczenia wykrytego znaku akcji. By znak zawarty w tym następniku mógł zostać sprawdzony nim zmienna sterująca pętlą pozyska adres następnika w wiązaniu głównym, następne instrukcje w pętli muszą zostać pominięte instrukcją continue, co spowoduje rozpoczęcie kroku pętli od początku. Z każdym krokiem pętli adres elementu aktualnie analizowanego zostaje zachowany we wskaźniku ad2 – wskazuje on na pewny element mianownika. Pętla kończy swoją pracę, gdy zmienna n1 osiągnie wartość mniejszą lub równą zero lub gdy zmienna sterująca pętlą, czyli wskaźnik ad1, uzyska adres pusty, bądź gdy adres zawarty we wskaźniku ad2 będzie równy adresowi struktury granicznej.

```
repeat
      k1:=ord(ad1^.znak); - zapamiętaj w zmiennej k1 wartość znaku wpisanego...
                            w pole znak analizowanego elementu
      if k1 in otw then inc(n1); - jeśli wartość znaku zawiera się w tablicy...
                                   otw, zwiększ wartość w zmiennej n1 o jeden
      if n1<0 then break; - jeśli liczba nawiasów jest mniejsza od zera, opuść pętlę
      if k1 in zam then dec(n1); - jeśli wartość znaku zawiera się w tablicy...
                                   zam, zmniejsz wartość w zmiennej n1 o jeden
      if k1 in dzed then - czy wartość znaku zawiera się w tablicy dzed?
      begin - jeśli tak, to...
            ad1:=ad1^.kur_p; - do zmiennej ad1 wczytaj adres następnika...
                               w wiązaniu kursorowym
            continue - pomiń dalsze instrukcje w pętli i rozpocznij nowy jej krok
      end;
      ad2:=ad1; - zachowaj adres elementu mianownika we wskaźniku ad2
      ad1:=ad1^.prawa - we wskaźniku ad1 pozyskaj adres następnika...
                        w wiązaniu głównym
until (ad1=nil) or (n1=0) or (ad2=ad4); - opuść pętlę, gdy wskaźnik ad1...
                  uzyska adres pusty lub gdy licznik nawiasów osiągnie wartość zero,…
                  lub gdy do wskaźnika ad2 wpisany został adres struktury granicznej
```

Po poprawnym wyznaczeniu mianownika, wskaźnik ad2 będzie wskazywał na ostatni element mianownika. Nie jest możliwe, by wskaźnik ten po opuszczeniu pętli posiadał adres pusty, gdyż obecność nawiasu otwierającego w następniku elementu znaku dzielenia umożliwia uruchomienie tej pętli, zatem następnik ten jest pewnym składnikiem mianownika. Jeśli adres zawarty we wskaźniku ad2 należy do ostatniego składnika listy lisc, co może mieć miejsce podczas sekwencyjnego wprowadzania znaków z klawiatury, wskaźnik ad1 będzie posiadał adres pusty, co oznacza, że brakuje elementu zamykającego ułamek. W tej sytuacji zmienna logiczna brak zostaje ustawiona w pozycji true, zaznaczając w ten sposób potrzebę utworzenia struktury zamykającej. Jeśli jednak wskaźnik ad1 posiada adres składnika

lisc i jest on związany wiązaniem kursorowym z lewej strony z ostatnim elementem mianownika, składnik ten wiązany jest kursorowo z elementem znaku dzielenia, ustanawiając go strukturą zamykającą. Może się zdarzyć, że element dostępny przez wskaźnik ad1 nie będzie związany kursorowo z ostatnim elementem mianownika lecz z innym – w takiej sytuacji związanie to musi pozostać nienaruszone, gdyż jego rozbicie mogłoby spowodować nieprzewidziane skutki podczas dalszego układania funkcji przez pozostałe procedury modułu. Oczywiście w tej sytuacji element ten nie może należeć do otoczenia składanego ułamka, dlatego zmienna logiczna brak zostaje ustawiona w pozycji true.

```
if ad1=nil then licz:=true - jeśli wskaźnik ad1 posiada adres pusty, to...
                             zaznacz brak struktury zamykającej

else - w przeciwnym przypadku...
    if ad1^.kur_l<>ad2 then brak:=true - jeśli element dostępny przez wskaźnik..
          ad1 nie jest związany wiązaniem kursorowym z ostatnim elementem mianownika,…
          ustaw zmienną logiczną brak w pozycji true

    else - w przeciwnym przypadku...
    begin
         ad3^.kur_p:=ad1; { związ wzajemnie wiązaniem kursorowym...
         ad1^.kur_l:=ad3;   strukturę zamykającą tego ułamka (ad1),...
    end                     z elementem znaku dzielenia (ad3)
```

Opisane wyżej działania wykonują się tylko wtedy, gdy spełnione zostały wszystkie warunki pozwalające na wyznaczenie mianownika. Nie spełnienie choć jednego z nich pozostawia we wskaźniku ad2 adres pusty, będący informacją o braku mianownika.

Jeżeli następniki elementu znaku dzielenia w wiązaniu głównym i kursorowym różnią się, oznacza to, że mianownik został już zbudowany podczas poprzedniego wywołania procedury. Świadczą o tym dwa różne adresy elementów lisc zawarte we wskaźnikach ad1 i ad4. W takiej sytuacji uruchomiona zostaje pętla while..do, w której – poza zliczaniem nawiasów – wyszukiwany jest ostatni element mianownika. Warunkiem opuszczenia pętli jest uzyskanie przez zmienną sterującą ad1 adresu struktury zamykającej, dostępnej przez wskaźnik ad4.

```
ad1:=ad3^.prawa; - do zmiennej ad1 wpisz adres następnika elementu znaku dzielenia...
                   w wiązaniu głównym
ad4:=ad3^.kur_p; - do zmiennej ad4 wpisz adres następnika elementu znaku dzielenia,...
                   w wiązaniu kursorowym
. . .
if ad4<>nil then - czy wskaźnik ad4 nie ma adresu pustego?
   if ad4=ad1 then - jeśli nie ma adresu pustego, oznacza to, że wskaźnik ad1...
    także nie ma adresu pustego. Czy zatem obydwa wskaźniki wskazują na ten sam element?
      if ord(ad1^.znak) in otw then - jeśli tak, to czy element ten ma wpisany...
                                      znak nawiasu otwierającego?
      begin
           . . .
      end else - jeśli nie, to...
          if ord(ad1^.znak) in dzed then - czy element następnika ma...
                                           wpisany znak akcji?
           begin - jeśli tak, to...
                licz:=true; - zaznacz brak struktury zamykającej
                brak:=true - zaznacz obecność elementu spoza ułamka
           end else
   else - jeśli adresy we wskaźnikach ad1 i ad4 różnią się, to...
   begin
```

```
        if ord(ad4^.znak) in dzed then - czy następnik w wiązaniu...
                                        kursorowym ma wpisany znak akcji?
        begin - jeśli tak, to...
              licz:=true; - zaznacz brak struktury zamykającej
              brak:=true - zaznacz obecność elementu spoza ułamka, który należy...
                           związać ze strukturą zamykającą wiązaniem kursorowym
        end;
        while ad1<>nil do - wykonaj krok pętli, póki zmienna ad1 nie ma...
                            adresu pustego
        begin
              if ad1=ad4 then break; - jeśli zmienna ad1 uzyska adres struktury...
                                       zamykającej, opuść pętlę
              k1:=ord(ad1^.znak); - wczytaj do zmiennej k1 wartość znaku,...
                                    wpisanego w pole znak analizowanego elementu
              if k1 in otw then inc(n1); - jeśli wartość znaku zawiera się...
                                           w tablicy otw, zwiększ wartość w zmiennej n1 o jeden
              if k1 in zam then dec(n1); jeśli wartość znaku zawiera się...
                                         w tablicy zam, zmniejsz wartość w zmiennej n1 o jeden
              ad2:=ad1; - zachowaj adres element mianownika w zmiennej ad2
              ad1:=ad1^.prawa - w zmiennej ad1 pozyskaj adres następnika...
                                w wiązaniu głównym
        end
   end
else licz:=true; - jeśli wskaźnik ad4 ma wpisany adres pusty, ustaw zmienną...
                   licz w pozycji true
```

Po opuszczeniu pętli `while..do`, wskaźnik `ad2` będzie zawierał adres ostatniego elementu mianownika. Zliczanie nawiasów w pętli może wydawać się bezcelowe, jednak wartość zachowana w liczniku nawiasów, czyli w zmiennej `n1`, będzie wykorzystana w dalszej części procedury.

Wykrycie wartości `true` w zmiennej logicznej `licz` spowoduje utworzenie w pamięci struktury zamykającej i związanie jej z otoczeniem ułamka i ze składnikiem należącym do dalszej części funkcji. Tuż przed utworzeniem w pamięci struktury zamykającej, we wskaźniku `ad4` zachowany zostaje adres elementu spoza ułamka. Jeśli element ten istnieje oraz jego pozycja w pionie jest taka sama jak elementu znaku dzielenia, jest on związany z nowo utworzoną strukturą zamykającą także wiązaniem kursorowym.

```
if licz then - czy zmienna logiczna licz jest ustawiona w pozycji true?
begin - jeśli tak, to...
      ad4:=ad1; - zachowaj w zmiennej ad4 możliwy adres elementu spoza ułamka
      new(ad1); - utwórz w pamięci nowy element lisc (struktura zamykająca)
      if ad2<>nil then ad2^.prawa:=ad1; - jeśli mianownik istnieje,...
       związ wiązaniem głównym ostatni element mianownika z nowym elementem
      if ad4<>nil then ad4^.lewa:=ad1; - jeśli element spoza ułamka istnieje,...
                                         związ go wiązaniem głównym z nowym elementem
      ad3^.kur_p:=ad1; - związ wiązaniem kursorowym element znaku dzielenia...
                         z nowym elementem
      with ad1^ do
      begin
            prawa:=ad4;
            lewa:=ad2;
            if brak then - czy zmienna brak została ustawiona w pozycji true?
            begin - jeśli tak, to...
```

```
                kur_p:=ad4;          } związ wiązaniem kursorowym element ...
                ad4^.kur_l:=ad1      } spoza ułamka z nowym elementem
            end else kur_p:=nil; - w przeciwnym przypadku, wpisz w pole...
                                   kur_p nowego elementu identyfikator adresu pustego
            kur_l:=ad3;
            kur_g:=nil;
            kur_d:=nil;
            lx:=ad3^.px;
            px:=lx;
            ly:=ad3^.ly;
            py:=ad3^.py;
            y1:=ly;
            y2:=py;
            znak:=' ';
            druk:=true
        end;
end;
```

Wykrycie we wskaźniku `ad2` adresu pustego świadczącego o braku mianownika, powoduje utworzenie w pamięci jedynego elementu mianownika z wpisanym w pole `znak` nawiasem otwierającym.

Skoro mianownik jest już obecny, pozostaje zamknięcie go z lewej i prawej strony. Odbywa się to w taki sam sposób, w jaki zamknięty został licznik, a więc przez wpisanie adresu pustego w pole `kur_l` pierwszego i `kur_p` ostatniego elementu mianownika, wyznaczając w ten sposób granicę w poruszaniu się kursorem w obrębie mianownika.

Jeśli wprowadzony z klawiatury nawias zamykający okaże się ostatnim znakiem mianownika, ważne jest, by we wskaźniku edytora `wsb` znalazł się adres struktury zamykającej. Umożliwi to użytkownikowi wprowadzanie znaków już poza kompletnym ułamkiem bez dodatkowego przemieszczenia kursora za kreskę ułamkową. Informacji o zaistniałej sytuacji dostarcza równość adresów zawartych zarówno w ostatnim elemencie mianownika, jak i we wskaźniku edytora `wsb`. Gdy okaże się, że oba adresy są identyczne – przy dodatkowym warunku, że licznik nawiasów w mianowniku, czyli zmienna `n1`, jest równa zero – wskaźnik edytora `wsb` przyjmuje adres struktury zamykającej, czyli znaku znajdującego się za kreską ułamkową, zaś zmienna logiczna `stat_kur` ustawiona zostaje w pozycji `false`, powodując tym samym pojawienie się kursora przed znakiem.

```
if ad2=nil then - czy wskaźnik ad2 ma wpisany adres pusty?
begin - jeśli tak, to...
     new(ad2); - utwórz w pamięci nowy element lisc (mianownik)
     . . .
     if ad3=wsb then wsb:=ad2; - jeżeli w głównym wskaźniku edytora znajduje się...
                     adres elementu znaku dzielenia, wpisz do wskaźnika edytora – wsb,...
                     adres struktury zamykającej
     Form2.Szerokosc(ad2) - wylicz szerokość elementu mianownika,...
                            modyfikując jego pole px
end else w przeciwnym przypadku...
    begin
         ad2^.kur_p:=nil; - zamknij ostatni element mianownika z prawej strony
         ad4:=ad3^.prawa; - w zmiennej ad4 wpisz adres pierwszego elementu...
                            mianownika
         ad4^.kur_l:=nil; - zamknij pierwszy element mianownika z lewej strony
         if n1=0 then - czy liczba nawiasów mianownika jest równa zero?
```

```
                if ad2=wsb then - jeśli tak, to czy wskaźnik edytora – wsb...
                                  posiada adres ostatniego elementu mianownika?
                begin - jeśli tak, to...
                     wsb:=ad1; - wpisz wskaźnikowi wsb adres struktury zamykającej
                     stat_kur:=false - ustaw zmienną edytora – stat_kur,...
                              w pozycji false, ustawiającą kursor przed znakiem
                end
        end;
```

Gdy mianownik jest obecny w ułamku, pozostało związanie jego elementów z elementem znaku dzielenia, umożliwiając poruszanie się kursorem między mianownikiem a kreską ułamkową. Sama zasada wiązania jest porównywalna z tą, zastosowaną przy wiązaniu elementów licznika z elementem znaku dzielenia. Różnica polega na przekazaniu w parametrze funkcji `wiazanie` wartości `true`, zmieniającej zasady badania otrzymanego elementu mianownika, pod kątem możliwości związania tego elementu z kreską ułamkową. Ponadto, w pętli `repeat..until`, szukana jest minimalna wartość współrzędnej pionowej, pochodząca z pól `y1` elementów mianownika, która zapamiętana w zmiennej liczbowej `m1` będzie wykorzystana do zwymiarowania ułamka.

```
ad4:=ad3^.prawa; - w zmiennej ad4 pozyskaj adres pierwszego elementu mianownika
ukryj_nawiasy(ad4,ad2); - ukryj skrajną parę nawiasów mianownika
ad4:=ad2; - do zmiennej ad4 wpisz adres ostatniego elementu mianownika
brak:=false; - ustaw zmienną logiczną brak w pozycji false
m1:=maxint; - do zmiennej m1 wpisz startową wartość równą maksymalnej wartości...
              typu integer
repeat
     if not wiazanie(ad4,ad3,true) then ad4^.kur_g:=ad3; - jeśli...
                                  funkcja wiazanie zwróciła wartość false,...
                                  zwiąż element znaku dzielenia z elementem...
                                  mianownika, wiązaniem kursorowym kur_g
     if not brak then - czy zmienna brak jest ustawiona w pozycji false?
        if ad4^.druk then - jeśli tak, to czy element nie ma ukrytego znaku?
           if ad4^.kur_g=ad3 then - jeśli znak nie jest ukryty, to czy element...
                                    jest już związany z elementem...
                                    znaku dzielenia polem kur_g?
           begin - jeśli tak, to...
                ad3^.kur_d:=ad4; - zwiąż element mianownika z elementem...
                                   znaku dzielenia, polem kur_d
                brak:=true - zablokuj możliwość wiązania z innym...
                             elementem mianownika, ustawiając zmienną...
                             brak w pozycji true
           end;
           n1:=ad4^.y1; - do zmiennej n1 wczytaj wartość współrzędnej pionowej...
                          z pola y1 analizowanego elementu mianownika
           if m1>n1 then m1:=n1; - jeżeli w zmiennej m1 wpisana została...
                                   większa wartość liczbowa niż w zmiennej n1,...
                                   wpisz do zmiennej m1 wartość ze zmiennej n1
           ad4:=ad4^.lewa - w zmiennej ad4 pozyskaj adres poprzednika...
                            w wiązaniu głównym
until ad4=ad3; - opuść pętlę, gdy zmienna ad4 uzyska adres elementu znaku dzielenia
```

Ułamek jest już kompletny, więc należy go zwymiarować. Polega to na takim ułożeniu licznika i mianownika względem kreski ułamkowej, by ułamek wyglądał na ekranie matema-

tycznie. Skoro kreska ułamkowa jest elementem odniesienia, względem której ustalane jest położenie licznika i mianownika, w pierwszej kolejności korygowane są rzeczywiste współrzędne elementu znaku dzielenia. Polega to na wyliczeniu takiej wartości, która będzie zawierała się w połowie między górną współrzędną pionową `ly` a dolną `py` poziomu, na którym znajduje się znak. Różnica między tą wartością a rzeczywistą współrzędną znaku dzielenia, zawartą w polu `y1` lub `y2`, wskazuje na potrzebę skorygowania tych pól o wskazaną różnicę.

```
k1:=ad3^.ly+trunc((ad3^.py-ad3^.ly)/2)-ad3^.y1; - w zmiennej k1 wylicz ...
                                różnicę między wymaganą wartością współrzędnej rzeczywistej...
                                elementu znaku dzielenia a dotychczasową
if k1<>0 then - czy wartość w zmiennej k1jest różna od zera?
begin - jeśli tak, to...
     ad3^.y1:=ad3^.ly+k1; - wpisz w pole y1 sumę wartości górnej, współrzędnej...
                                pionowej poziomu i wyliczonej różnicy w zmiennej k1
     ad3^.y2:=ad3^.y1 - wpisz w pole y2 wartość z pola y1...
                                (kreska ułamkowa nie ma wysokości)
end;
```

Gdy współrzędne pionowe rzeczywiste elementu znaku dzielenia są poprawne, można już wyliczyć wartości, o jakie należy skorygować współrzędne pionowe elementów licznika i mianownika, by ich odległości od kreski ułamkowej były poprawne. Możliwość tą zapewniają zmienne `l1` i `m1`, przechowujące wartości współrzędnych pionowych tych elementów, które są najbardziej wysunięte w dół – w przypadku licznika lub w górę – w przypadku mianownika. Wielkość korekcji jest wynikiem różnicy między tymi wartościami a sumą rzeczywistej współrzędnej pionowej elementu znaku dzielenia oraz regulowanego odstępu od kreski ułamkowej, pochodzącego ze zmiennej globalnej `KR` bloku regulacji.

```
l1:=ad3^.y1-Form6.KR-l1; - wylicz wielkość korekcji współrzędnych pionowych licznika
m1:=ad3^.y1+Form6.KR-m1; - wylicz wielkość korekcji współrzędnych pionowych...
                                mianownika
if m1<>0 then - czy zmienna m1 jest różna od zera?
begin - jeśli tak, to skoryguj elementy mianownika
     ad4:=ad3; - przypisz zmiennej ad4 adres elementu znaku dzielenia
     repeat
          ad4:=ad4^.prawa; - w zmiennej ad4 pozyskaj adres następnika...
                                w wiązaniu głównym
          with ad4^ do
          begin
               ly:=ly+m1;
               py:=py+m1;   } skoryguj współrzędne pionowe elementu...
               y1:=y1+m1;   } mianownika o wielkość korekcji
               y2:=y2+m1
          end;
     until ad4=ad2; - opuść pętlę po skorygowaniu ostatniego elementu mianownika
end;
if l1<>0 then - czy zmienna l1 jest różna od zera?
begin - jeśli tak, to skoryguj elementy licznika
     ad4:=ap1; - do zmiennej ad4 wpisz adres pierwszego elementu licznika
```

```
        repeat
              with ad4^ do
              begin
                    ly:=ly+l1;
                    py:=py+l1;   } skoryguj współrzędne pionowe elementu...
                    y1:=y1+l1;     licznika o wielkość korekcji
                    y2:=y2+l1
              end;
              ad4:=ad4^.prawa - w zmiennej ad4 pozyskaj adres następnika...
                                w wiązaniu głównym
        until ad4=ad3; - opuść pętlę, gdy zmienna ad4 uzyska adres...
                         elementu znaku dzielenia
  end;
```

Gdy współrzędne pionowe zostały ustawione, pora na współrzędne poziome. Za pomocą funkcji `dlugosc`, zmiennym `l1` i `m1` przypisane zostają długości odpowiednio licznika i mianownika. Wybierając spośród nich większą wartość, wyliczana jest końcowa współrzędna pozioma `px` elementu znaku dzielenia, która jest sumą wybranej, większej długości i początkowej współrzędnej `lx`. W ten sposób ustalona zostaje długość kreski ułamkowej.

```
  l1:=dlugosc(apl,ad3); - wylicz i zapamiętaj w zmiennej l1 długość licznika
  ad1:=ad3^.prawa; - wpisz do zmiennej ad1 adres pierwszego elementu mianownika
  ad4:=ad3^.kur_p; - do zmiennej ad4 wpisz adres struktury zamykającej
  m1:=dlugosc(ad1,ad4); - wylicz i zapamiętaj w zmiennej m1 długość mianownika
  if l1>m1 then k1:=ad3^.lx+l1 - jeśli długość licznika jest większa od mianownika,...
                                 na podstawie długości licznika wylicz w zmiennej...
                                 k1 końcową współrzędną poziomą kreski ułamkowej
    else k1:=ad3^.lx+m1; - w przeciwnym przypadku, na podstawie długości mianownika...
                           wylicz w zmiennej k1 końcową współrzędną kreski ułamkowej
  if ad3^.px<>k1 then ad3^.px:=k1; - jeśli wartość w polu px elementu znaku...
    dzielenia jest inna niż wartość w zmiennej k1, wpisz w pole px wartość ze zmiennej k1
```

Jeśli długość kreski ułamkowej uległa zmianie, koniecznie należy przesunąć elementy znajdujące się za kreską o różnicę między końcową współrzędną `px` elementu znaku dzielenia i początkową współrzędną `lx` struktury zamykającej. W pierwszej kolejności korygowane są pola współrzędnych poziomych struktury zamykającej, a następnie, za pomocą procedury `Przesun`, pozostałe elementy, związane ze sobą wiązaniami kursorowymi.

```
  ad1:=ad3^.kur_p; - do zmiennej ad1 wpisz adres struktury zamykającej
  n1:=k1-ad1^.lx; - wylicz różnicę między wartością w zmiennej k1 i współrzędną...
                    początkową z pola lx struktury zamykającej
  if n1<>0 then - czy wyliczona różnica w zmiennej n1 jest różna od zera?
  begin - jeśli tak, to...
        m1:=ad1^.px-ad1^.lx; - wylicz szerokość znaku zawartego...
                               w strukturze zamykającej
        ad1^.lx:=ad1^.lx+n1; - skoryguj pole lx struktury zamykającej...
                               o wyliczoną różnicę
        ad1^.px:=ad1^.lx+m1; - wpisz w pole px struktury zamykającej wartość...
                               będącą sumą skorygowanego pola lx i szerokości znaku
```

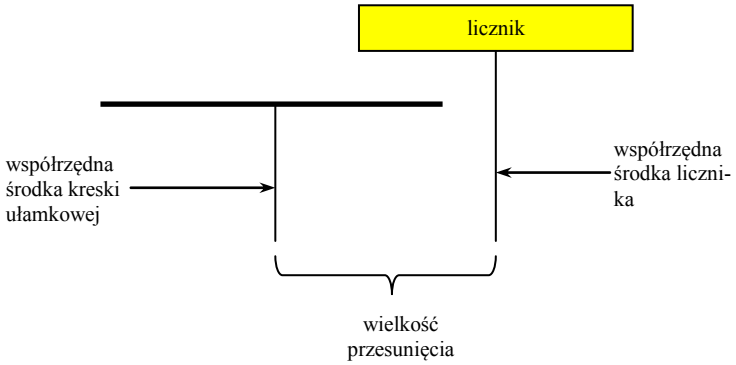
```
        Form2.Przesun(ad1,true) - przesuń pozostałe elementy względem...
                                  poprawionych współrzędnych poziomych...
                                  struktury zamykającej, zawartej we wskaźniku ad1
    end;
```

Ułamek jest już prawie zwymiarowany. Pozostało jeszcze wyśrodkowanie licznika lub mianownika względem kreski ułamkowej. Długość kreski ułamkowej jest równa dłuższemu elementowi ułamka, czyli licznikowi albo mianownikowi, dlatego tylko krótszy element może wymagać środkowania. Wielkość, o jaką należy przesunąć licznik lub mianownik, dostarcza wewnętrzna funkcja opisywanej procedury – `srodek`. Na podstawie tej wielkości, pola współrzędnych poziomych elementów licznika lub mianownika, są korygowane w oddzielnych pętlach.

```
    m1:=srodek(apl,ad3,ad2,true); - wylicz i zapamiętaj w zmiennej m1 wielkość...
                                    potrzebną do środkowania mianownika względem kreski ułamkowej
    if m1<>0 then - czy wartość w zmiennej m1 jest różna od zera?
    begin - jeśli tak, to wyśrodkuj mianownik ułamka...
        ad1:=ad3; - wpisz do zmiennej ad1 adres elementu znaku dzielenia
        repeat
            ad1:=ad1^.prawa; - w zmiennej ad1 pozyskaj adres następnika...
                               w wiązaniu głównym
            ad1^.lx:=ad1^.lx+m1; - skoryguj współrzędną poziomą z lewej...
                                   strony znaku o wartość w zmiennej m1
            ad1^.px:=ad1^.px+m1 - skoryguj współrzędną poziomą w prawej...
                                  strony znaku o wartość w zmiennej m1
        until ad1=ad2; - opuść pętlę, po skorygowaniu ostatniego elementu mianownika
    end;
    m1:=srodek(apl,ad3,ad2,false); - wylicz i zapamiętaj w zmiennej m1 wielkość...
                                     potrzebną do środkowania licznika względem kreski ułamkowej
    if m1<>0 then - czy wartość w zmiennej m1 jest różna od zera?
    repeat - jeśli tak, to rozpocznij pętlę korygującą współrzędne poziome elementów licznika
        apl^.lx:=apl^.lx+m1; - (jak wyżej)
        apl^.px:=apl^.px+m1; - (jak wyżej)
        apl:=apl^.prawa - w zmiennej apl pozyskaj adres następnika w wiązaniu głównym
    until apl=ad3; - opuść pętlę, gdy zmienna apl uzyska adres elementu znaku dzielenia
```

XIV. 2.1.1. Środkowanie licznika/mianownika względem kreski ułamkowej – funkcja `srodek`

Zadaniem funkcji jest wyliczenie wielkości przesunięcia w poziomie elementów licznika lub mianownika tak, by po wyświetleniu ułamka na ekranie, licznik i mianownik były wyśrodkowane względem kreski ułamkowej. Funkcja otrzymuje w parametrach adresy: pierwszego elementu licznika, elementu znaku dzielenia i ostatniego elementu mianownika. Czwarty parametr logiczny podpowiada funkcji, dla której części ułamka ma wyliczyć wielkość przesunięcia: wartość `true` dotyczy mianownika, natomiast `false` – licznika. Na początku funkcja wylicza wartości współrzędnych, które znajdują się dokładnie w połowie między najmniejszą a największą współrzędną poziomą badanych części ułamka. Różnica między dwiema wyliczonymi wartościami jest szukaną wielkością przesunięcia, zwracaną przez funkcję (rysunek 47).



Rys. 47. Zasada wyliczania wielkości przesunięcia licznika w celu wyśrodkowania go względem kreski ułamkowej

```
function srodek(const pocz,srod,kon2:lisc;
                liczmia:boolean):integer;
var b1:lisc;
    ka,kb,kd:integer;
begin
     kd:=srod^.lx+trunc((srod^.px-srod^.lx)/2); - wylicz...
          i zapamiętaj w zmiennej kd współrzędną środka kreski ułamkowej

     if liczmia then - czy parametr liczmia ma wartość true?
     begin - jeśli tak, to wyśrodkuj mianownik
          b1:=srod; - wpisz do zmiennej b1 adres elementu znaku dzielenia
          ka:=maxint; - wpisz do zmiennej ka maksymalną wartość...
                        dla typu integer, będącą wartością startową...
                        dla szukanej minimalnej współrzędnej poziomej

          kb:=0; - wpisz do zmiennej kb zero będącą wartością startową w...
                   szukaniu maksymalnej współrzędnej poziomej

          while b1<>kon2 do - wykonaj krok pętli, póki zmiennej b1...
               nie zostanie wpisany adres ostatniego elementu mianownika
          begin
               b1:=b1^.prawa; - pozyskaj w zmiennej b1 adres następnika...
                                w wiązaniu głównym

               if ka>b1^.lx then ka:=b1^.lx; - jeśli wartość...
                                 w zmiennej ka jest większa od współrzędnej...
                                 zawartej w polu lx badanego elementu,...
                                 wpisz do tej zmiennej wartość współrzędnej...
                                 z pola lx tego elementu

               if kb<b1^.px then kb:=b1^.px - jeśli wartość...
                                 w zmiennej kb jest mniejsza od współrzędnej...
                                 zawartej w polu px badanego elementu,...
                                 wpisz do tej zmiennej wartość współrzędnej...
                                 z pola px tego elementu
          end;
          kb:=ka+trunc((kb-ka)/2); - wylicz i zapamiętaj w zmiennej...
                                     kb współrzędną środka mianownika

          Result:=kd-kb - wylicz różnicę między współrzędnymi środków...
                          i zwróć ją za pośrednictwem zmiennej Result

     end else - jeśli parametr liczmia nie ma wartości true,...
                to wyśrodkuj licznik
```

```
            begin
                 b1:=pocz; - wpisz do zmiennej b1 adres pierwszego...
                             elementu licznika
                 ka:=maxint; - (jak wyżej)
                 kb:=0; - (jak wyżej)
                 while b1<>srod do - wykonaj krok pętli, póki zmienna...
                                     b1 nie uzyska adresu elementu znaku dzielenia
                 begin
                      if ka>b1^.lx then ka:=b1^.lx; - (jak wyżej)
                      if kb<b1^.px then kb:=b1^.px; - (jak wyżej)
                      b1:=b1^.prawa - (jak wyżej)
                 end;
                 kb:=ka+trunc((kb-ka)/2); - wylicz i zapamiętaj...
                                            w zmiennej kb współrzędną środka licznika
                 Result:=kd-kb - wylicz różnicę między współrzędnymi...
                                 środków i zwróć ją za pośrednictwem zmiennej Result
            end;
end;
```

XIV. 2.2. Ułożenie pierwiastka – procedura `pierwiastek`

Zadaniem procedury jest złożenie kompletnego pierwiastka z elementów związanych z otrzymanym w parametrze elementem `lisc` o nazwie `adres`, który ma wpisany znak pierwiastkowania. Poza poprawnym ułożeniem funkcji podpierwiastkowej, procedura ma także za zadanie tak zmodyfikować współrzędne elementu z wpisanym znakiem pierwiastkowania, by na ich podstawie można było odwzorować symbol pierwiastka.

Na początku procedura musi zbadać, czy istnieje element stopnia pierwiastka, a jeśli tak, to czy stopień ten składa się tylko z jednej cyfry. Badanie polega na porównaniu ze sobą adresów elementów związanych z elementem znaku pierwiastkowania, z prawej strony wiązaniem głównym i kursorowym. Jeśli uzyskane adresy należą do elementów `lisc` oraz gdy różnią się, oznacza to, że istnieje element stopnia pierwiastka, który należy dokładnie sprawdzić. Jeśli do tego elementu dowiązane zostały kolejne elementy wiązaniem kursorowym z lewej lub prawej strony, muszą one zostać usunięte.

```
procedure pierwiastek(const adres:lisc);
var pi1,pi2,pi3,pi4,pom1:lisc;
    k2,s1,s2,n1:integer;
    brak,brak1:boolean;
begin
     brak:=false;
     pom1:=nil;
     pi1:=adres^.prawa; - do zmiennej pi1 wpisz adres następnika elementu...
                          znaku pierwiastka, w wiązaniu głównym
     pi3:=adres^.kur_p; - do zmiennej pi3 wpisz adres następnika elementu...
                          znaku pierwiastka, w wiązaniu kursorowym
     if pi1<>nil then - czy wskaźnik pi1 nie ma adresu pustego?
        if pi3<>nil then - jeśli nie, to czy wskaźnik pi3 nie ma adresu pustego?
           if pi1<>pi3 then - jeśli nie, to czy adresy te różnią się?
           begin - jeśli tak, to wskaźnik pi1 wskazuje na element...
                   stopnia pierwiastka
                pi2:=pi1^.kur_l; - do zmiennej pi2 wczytaj adres...
                                   poprzednika elementu stopnia pierwiastka,...
                                   w wiązaniu kursorowym
```

```
                    if pi2<>nil then - czy wskaźnik pi2 nie ma adresu pustego?
                    begin - jeśli nie, to...
                         pi1^.kur_l:=nil; - wpisz do pola kur_l elementu...
                                  stopnia pierwiastka identyfikator adresu pustego
                         pi1^.lewa:=adres;   { związ wzajemnie wiązaniem...
                         adres^.prawa:=pi1;  { głównym elementy stopnia...
                                             { pierwiastka i znaku pierwiastka
                         if wsb=pi2 then wsb:=pi1; - jeśli wskaźniki:...
                                  edytora – wsb i pi2, posiadają ten sam adres,...
                                  wpisz do wskaźnika edytora adres elementu...
                                  stopnia pierwiastka
                         dispose(pi2) - uwolnij pamięć zajmowaną przez...
                                   element wskazywany przez wskaźnik pi2
                    end;
                    pi2:=pi1^.kur_p; - do zmiennej pi2 wczytaj adres...
                                  następnika elementu stopnia pierwiastka...
                                  w wiązaniu kursorowym
                    if pi2<>nil then - czy wskaźnik pi2 nie ma adresu pustego?
                    begin - jeśli nie, to...
                         pi3:=pi2^.prawa; - do zmiennej pi3 wpisz adres...
                                pierwszego elementu funkcji podpierwiastkowej
                         pi1^.prawa:=pi3; - związ wiązaniem głównym...
                                  z prawej strony, element wskazywany przez...
                                  wskaźnik pi3 z elementem stopnia pierwiastka
                         if pi3<>nil then pi3^.lewa:=pi1; jeżeli...
                                     adres we wskaźniku pi3 nie jest pusty,...
                                     związ wiązaniem głównym z lewej strony,...
                                     element stopnia pierwiastka z elementem...
                                     wskazywanym przez wskaźnik pi3
                         pi1^.kur_p:=nil; - zamknij element stopnia...
                           pierwiastka z prawej strony, wpisując w jego pole...
                           kur_p identyfikator adresu pustego
                         if wsb=pi2 then wsb:=pi1; - jeśli wskaźnik...
                            edytora wsb i wskaźnik pi2, mają taki sam adres,...
                            wpisz do wskaźnika edytora adres elementu...
                            stopnia pierwiastka
                         dispose(pi2) - uwolnij pamięć zajmowana przez...
                           element, którego adres widnieje we wskaźniku pi2
                    end;
                                                 { związ wzajemnie element stopnia...
                    pi1^.kur_d:=adres;           { pierwiastka i znaku pierwiastka ..
                    adres^.kur_g:=pi1            { wiązaniem kursorowym...
                                                 { kur_g i kur_d.
               end else
```

Nie spełnienie choć jednego z powyższych warunków ustawia zmienną logiczną `brak` w pozycji `true`, która niesie informację o braku elementu stopnia pierwiastka. Brak tego elementu uniemożliwiłby późniejsze dopisanie przez użytkownika stopnia pierwiastka, dlatego należy go utworzyć.

```
if pi1<>nil then
   if pi3<>nil then
      if pi1<>pi3 then
      begin
           . . .
      end else - gdy następniki elementu znaku pierwiastka, w wiązaniu głównym...
                 i kursorowym są takie same, to...
          begin
               pom1:=pi1; - zapamiętaj adres następnika w wiązaniu głównym...
                            jest to pierwszy element funkcji podpierwiastkowej
               brak:=true - ustaw zmienną brak w pozycji true...
                            informującą o braku elementu stopnia pierwiastka
          end
   else
   begin
        pom1:=pi1; - (jak wyżej)
        brak:=true - (jak wyżej)
   end
else brak:=true; - ustaw zmienną brak w pozycji true informującą o braku...
                   elementu stopnia pierwiastka
```

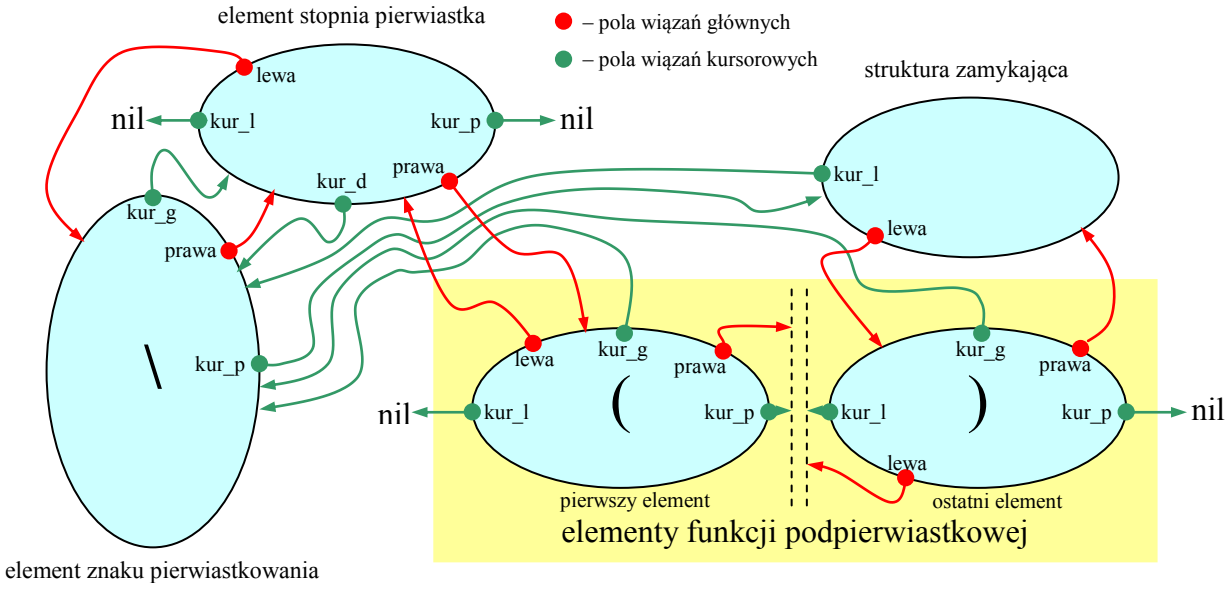
Wykrycie w zmiennej `brak` wartości `true`, powoduje utworzenie w pamięci elementu stopnia pierwiastka.

```
if brak then - czy zmienna brak została ustawiona w pozycji true?
begin - jeśli tak, to...
     new(pi1); - utwórz w pamięci składnik lisc – – element stopnia pierwiastka
     with pi1^ do
     begin
          lewa:=adres; - w pole lewa wpisz adres elementu znaku pierwiastka
          prawa:=pom1; - w pole prawa wpisz adres pierwszego elementu...
                         funkcji podpierwiastkowej (adres ten może być pusty)
          kur_g:=adres^.kur_g; - w pole kur_g wpisz adres elementu, który...
                 może się znajdować nad pierwiastkiem (adres ten może być pusty)
          kur_d:=adres; - w pole kur_d wpisz adres elementu znaku pierwiastka
          kur_l:=nil; - zamknij element stopnia pierwiastka z lewej strony
          kur_p:=nil; - jak wyżej, tylko z prawej strony
          lx:=adres^.lx; - element stopnia pierwiastka ma wstępną pozycję...
                           poziomą równą pozycji elementu znaku pierwiastka
          px:=lx; - póki pole znak nie ma wpisanego znaku cyfry,...
                    obie współrzędne poziome mają taką samą wartość
          znak:=' '; - w pole znak wpisz spację
          druk:=true; - znak ma być widoczny, by można było ustawić się...
                        kursorem w polu stopnia pierwiastka
          zazn:=false - znak nie jest zaznaczony
     end;
     adres^.prawa:=pi1; } związ nowy element z elementem...
     adres^.kur_g:=pi1; } znaku pierwiastka
end else
```

Gdy element stopnia pierwiastka został już utworzony podczas poprzedniego wywołania procedury, znak wprowadzony w jego pole `znak` musi zostać zweryfikowany. Dozwolone są jedynie cyfry w przedziale od 3 do 9 – każdy inny znak musi zostać zastąpiony spacją.

```
if brak then - czy zmienna brak została ustawiona w pozycji true?
begin
     . . .
end else - jeśli nie, element stopnia pierwiastka istnieje, więc...
    if pi1^.lx<>pi1^.px then - czy współrzędne poziome różnią się?
    begin  jeśli tak, to w polu znak nie ma już znaku spacji, więc...
          k2:=ord(pi1^.znak); - odczytaj zawartość pola znak jako wartość
          if (k2<51) or (k2>57) then - czy wartość w zmiennej k2...
               nie zawiera się w zakresie odpowiadającym cyfrom od 3 do 9?
          begin - jeśli warunek został spełniony, to...
               pi1^.znak:=' '; - zastąp niewłaściwy znak spacją
               pi1^.px:=pi1^.lx - zredukuj szerokość znaku do zera
          end;
    end;
```

Dalsze budowanie pierwiastka i wiązanie ze sobą jego elementów, przebiega dokładnie tak, jak pokazuje to rysunek 48.



Rys. 48. Struktura logiczna wiązania elementów pierwiastka

Wyznaczenie funkcji podpierwiastkowej oraz struktury zamykającej polega na sprawdzeniu, czy istnieje następnik elementu znaku pierwiastkowania w wiązaniu kursorowym oraz następnik elementu stopnia pierwiastka w wiązaniu głównym. Obecność pierwszego następnika oznacza także istnienie drugiego, co pozwala ograniczyć liczbę instrukcji warunkowych. Skoro istnieją oba następniki oraz gdy są to dwa różne elementy struktury `lisc`, uruchomienie procesu wyznaczania elementów dla funkcji podpierwiastkowej uzależnione jest jeszcze od obecności znaku nawiasu otwierającego w następniku elementu stopnia pierwiastka. Podobnie jak podczas wyznaczania mianownika w procedurze `dzielenie`, wyznaczenie funkcji podpierwiastkowej rozpoczyna się od znalezienia struktury granicznej, czyli takiego elementu, który nie będzie miał swojego następnika w wiązaniu kursorowym. Gdy struktura ta zostanie znaleziona, następuje uruchomienie pętli `repeat..until`, w której odbywa się szukanie ostatniego elementu funkcji podpierwiastkowej. Jej praca polega na identyfikowaniu każdego znaku wpisanego w pole `znak` kolejnego elementu związanego z poprzednim wiązaniem głównym. Jeśli sczytany znak jest nawiasem otwierających, wartość zmiennej `n1`

zwiększona zostaje o jeden; natomiast, gdy jest on nawiasem zamykającym, wartość tej zmiennej zmniejszana jest o jeden. Dla liczby nawiasów mniejszej od zera, pętla jest natychmiast przerywana instrukcją `break`. Tuż przed wczytaniem do zmiennej sterującej pętlą adresu następnego elementu, adres dotychczasowego elementu zostaje zapamiętany w osobnej zmiennej – jest to pewny element funkcji podpierwiastkowej.

Gdyby okazało się, że element analizowany w pętli ma wpisany znak akcji, do zmiennej sterującej pętlą wczytany zostaje adres następnika w wiązaniu kursorowym, pomijając w ten sposób elementy należące do otoczenia wykrytego znaku akcji. By znak w owym następniku mógł zostać w pętli sprawdzony nim wskaźnik sterujący pętlą przyjmie adres następnego elementu, pozostałe instrukcje w pętli zostają pominięte przez zastosowaną instrukcję `continue`, pozwalając na kontynuowanie pętli z niezmienionym adresem we wskaźniku `pi3`.

Opuszczenie pętli nastąpi po uzyskaniu przez zmienną sterującą adresu struktury granicznej bądź adresu pustego lub gdy licznik nawiasów, czyli zmienna `n1`, będzie miała wartość równą zero.

```
pi3:=pi1^.prawa; - wpisz do zmiennej pi3 adres następnika elementu stopnia...
                   pierwiastka w wiązaniu głównym
pi4:=adres^.kur_p; - wpisz do zmiennej pi4 adres następnika elementu znaku...
                     pierwiastkowania w wiązaniu kursorowym
pi2:=nil;
brak:=false;
brak1:=false;
n1:=0; - wyzeruj licznik nawiasów
if pi4<>nil then - czy wskaźnik pi4 nie ma adresu pustego?
   if pi4=pi3 then - jeśli nie, to czy wskaźniki pi3 i pi4 wskazują...
                     na ten sam element?
      if ord(pi3^.znak) in otw then - jeśli tak, to czy w polu znak...
              następnika stopnia pierwiastka znajduje się nawias otwierający?
      begin - jeśli tak, to...
            pi4:=nil; - zainicjuj zmienną pi4 adresem pustym
            pi2:=pi3; - wpisz do zmiennej pi2 adres następnika elementu...
                        stopnia pierwiastka w wiązaniu głównym
            pi1^.prawa:=pi2; } związ wzajemnie wiązaniem głównym element...
            pi2^.lewa:=pi1;  } stopnia pierwiastka i pierwszy element funkcji...
                               podpierwiastkowej
            pi2^.kur_l:=nil; - wskaźnik pi2 wskazuje na pierwszy element...
                               funkcji podpierwiastkowej, więc wpisz w pole kur_l...
                               identyfikator adresu pustego, zamykając ten element...
                               z lewej strony
            while pi2<>nil do - wykonaj krok pętli, póki wskaźnik pi2...
                                nie ma adresu pustego
            begin
                  pi4:=pi2; - wpisz do zmiennej pi4 adres elementu...
                              wskazywanego przez wskaźnik pi2
                  pi2:=pi2^.kur_p - we wskaźniku pi2 pozyskaj adres...
                                    następnika w wiązaniu kursorowym
            end;
```

```
repeat
      k2:=ord(pi3^.znak); - odczytaj znak wpisany w pole...
                            znak analizowanego elementu i zapamiętaj...
                            jego wartość w zmiennej k2

      if k2 in dzed then - czy odczytany znak należy do...
                           znaków akcji?

      begin jeśli tak, to...
           pi3:=pi3^.kur_p; - wczytaj do wskaźnika pi3 adres...
                              następnika w wiązaniu kursorowym

           continue - pomiń następne instrukcje w pętli i
kontynuuj...
                      pracę pętli od pierwszej instrukcji
      end;

      if k2 in otw then inc(n1); - jeśli odczytany znak...
                                   należy do znaków nawiasów otwierających,...
                                   zwiększ wartość w zmiennej n1 o jeden

      if k2 in zam then dec(n1); - jeśli odczytany znak...
                                   należy do znaków nawiasów zamykających,...
                                   zmniejsz wartość w zmiennej n1 o jeden

      if n1<0 then break; - gdy liczba nawiasów jest...
                            mniejsza od zera, opuść pętlę

      pi2:=pi3; - zapamiętaj adres elementu funkcji...
                  podpierwiastkowej w zmiennej pi2

      pi3:=pi3^.prawa - w zmiennej pi3 pozyskaj adres...
                        następnika w wiązaniu głównym

until (pi3=nil) or (n1=0) or (pi2=pi4); opuść...
                                        pętlę, gdy adres we wskaźniku pi3 jest pusty lub...
                                        gdy liczba nawiasów jest równa zero, lub...
                                        gdy wskaźnik pi3 uzyska adres struktury granicznej,...
                                        dostępnej przez wskaźnik pi4
```

Po opuszczeniu pętli, wskaźnik `pi2` powinien wskazywać na ostatni element funkcji podpierwiastkowej. Gdy wskaźnik ten wskazuje na element struktury `lisc` oraz gdy element ten ma swego następnika w wiązaniu kursorowym, następnik ten staje się strukturą zamykającą dla układanego pierwiastka. Nie spełnienie tych warunków ustawia zmienną logiczną `brak1` w pozycji `true`, będącą informacją dla procedury, że elementu `lisc` jako struktury zamykającej nie udało się znaleźć, dlatego musi on zostać utworzony.

```
if pi3<>nil then - czy wskaźnik pi3, po opuszczeniu pętli, nie ma adresu pustego?
   if pi2<>nil then - jeśli nie, to czy istnieje ostatni element...
                      funkcji podpierwiastkowej?

      if pi2^.kur_p<>nil then - jeśli istnieje, to czy istnieje także jego...
                                następnik w wiązaniu kursorowym?
      begin - jeśli tak, to...
           pi4:=pi2^.kur_p; - wczytaj do zmiennej pi4 adres następnika...
                              w wiązaniu kursorowym

           pi4^.kur_l:=adres; } element wskazywany przez wskaźnik...
           adres^.kur_p:=pi4; } pi4 jest strukturą zamykającą. Zwiąż...
                               } go z elementem znaku pierwiastkowania

           pi3:=pi4 - zapamiętaj jego adres we wskaźniku pi3
      end else - jeśli nie, to...
          begin
```

```
                pi4:=pi2^.prawa; - we wskaźniku pi4 zapamiętaj adres...
                                   elementu spoza pierwiastka

                brak1:=true - zaznacz brak struktury zamykającej
            end
    else
    if adres^.kur_p=pi3 then pi4:=pi3 else - jeśli następnik elementu...
        znaku pierwiastkowania, w wiązaniu kursorowym oraz element dostępny przez...
        wskaźnik pi3, to ten sam element, jest on strukturą zamykającą, więc...
        zapamiętaj jego adres we wskaźniku pi4

    begin - jeśli oba elementy różnią się, to...
          pi4:=adres^.prawa; - we wskaźniku pi4 zapamiętaj adres elementu...
                               spoza pierwiastka

          brak1:=true - zaznacz brak struktury zamykającej
    end
else jeśli po opuszczeniu pętli, wskaźnik pi3 ma adres pusty, to...
begin
     pi4:=nil; - brak elementu spoza pierwiastka, wpisz do wskaźnika pi4...
                 identyfikator adresu pustego

     brak1:=true - zaznacz brak struktury zamykającej
end
```

Może się zdarzyć, że mimo istnienia następnika elementu stopnia pierwiastka w wiązaniu głównym, nie można wyznaczyć funkcji podpierwiastkowej. Sytuacja taka może mieć miejsce wtedy, gdy następnik ten nie będzie miał wpisanego znaku nawiasu otwierającego. Wówczas to ten następnik staje się strukturą zamykającą pierwiastek, natomiast brak jedynego elementu funkcji podpierwiastkowej zostaje uzupełniony zaraz po wykryciu w instrukcji warunkowej obecności adresu pustego we wskaźniku `pi2`.

```
if ord(pi3^.znak) in otw then - czy następnik elementu stopnia pierwiastka,...
                                w wiązaniu głównym, ma wpisany znak nawiasu otwierającego?
begin
     . . .
end else - jeśli nie, to...
    if ord(pi3^.znak) in dzed then brak1:=true jeśli następnik...
           elementu stopnia pierwiastka zawiera w swym polu znak znak akcji,...
           to ustaw zmienną brak1 w pozycji true – brak struktury zamykającej
    else
```

Gdy adresy elementów `lisc`, dostępne przez wskaźniki `pi3` i `pi4`, różnią się, funkcja podpierwiastkowa została już określona podczas poprzedniego wywołania procedury. Pozostaje zdobycie adresu ostatniego elementu tej funkcji oraz związanie wiązaniem głównym pierwszego jej elementu z elementem stopnia pierwiastka. Odbywa się to w pętli `while..do`, której opuszczenie nastąpi po uzyskaniu przez zmienna sterującą pętlą adresu struktury zamykającej, dostępnej we wskaźniku `pi4`.

```
if pi4<>nil then - czy następnik elementu znaku pierwiastkowania...
                   w wiązaniu kursorowym istnieje?
   if pi4=pi3 then - jeśli tak, to czy następnik ten oraz następnik elementu...
                     stopnia pierwiastka w wiązaniu głównym, to ten sam element?
   . . .
   else - jeśli oba następniki to dwa różne elementy, to...
   while pi3<>nil do - wykonaj krok pętli, póki wskaźnik pi3...
                       nie ma adresu pustego
   begin
```

```
            if pi3=pi4 then break; - jeśli wskaźnik pi3 uzyskał adres elementu...
                                     wskazywanego przez wskaźnik pi4, opuść pętlę
            k2:=ord(pi3^.znak); - odczytaj znak wpisany w pole znak...
                                  elementu wskazywanego przez wskaźnik pi3...
                                  i zapamiętaj jego wartość w zmiennej liczbowej k2
            if k2 in otw then inc(n1); - jeśli odczytany znak jest nawiasem...
                                         otwierającym, zwiększ wartość w zmiennej n1 o jeden
            if k2 in zam then dec(n1); - jeśli odczytany znak jest nawiasem...
                                         zamykającym, zmniejsz wartość w zmiennej n1 o jeden
            if pi2=nil then - czy wskaźnik pi2 posiada jeszcze adres pusty?
            begin - jeśli tak, to element dostępny przez wskaźnik pi3 jest...
                    pierwszym elementem funkcji podpierwiastkowej, więc...
                 pi1^.prawa:=pi3; } związ wzajemnie ten element z elementem...
                 pi3^.lewa:=pi1   } stopnia pierwiastka wiązaniem głównym
            end;
            pi2:=pi3; - zachowaj adres tego elementu we wskaźniku pi2
            pi3:=pi3^.prawa - we wskaźniku pi3 pozyskaj adres następnika...
                              w wiązaniu głównym
        end
```

Brak następnika elementu znaku pierwiastkowania w wiązaniu kursorowym wskazuje jednocześnie na brak struktury zamykającej, dlatego w tej sytuacji zmienna logiczna `brak1` musi zostać ustawiona w pozycji `true` by brakujący element mógł zostać uzupełniony. Możliwy adres następnika elementu stopnia pierwiastka w wiązaniu głównym zostaje zachowany we wskaźniku `pi4` – element wskazywany przez ten wskaźnik może należeć do dalszej części funkcji, który należy związać z pierwiastkiem.

```
if pi4<>nil then - czy następnik elementu znaku pierwiastkowania...
                   w wiązaniu kursorowym istnieje?
. . .
else - jeśli nie, to...
begin
     pi4:=pi1^.prawa; - zachowaj we wskaźniku pi4 adres elementu...
                        spoza pierwiastka
     brak1:=true - zaznacz brak struktury zamykającej
end;
```

Wykrycie w zmiennej logicznej `brak1` wartości `true` powoduje utworzenie w pamięci nowego elementu `lisc` – struktury zamykającej. Warto zaznaczyć, że gdy wskaźnik `pi4` nie posiada adresu pustego, wykryty został element spoza otoczenia pierwiastka, dlatego zostaje on związany ze strukturą zamykającą wiązaniem głównym, a jeśli znak wpisany w jego pole `znak` znajduje się na tym samym poziomie co znak pierwiastkowania, także wiązaniem kursorowym, umożliwiając w ten sposób swobodne poruszanie się kursorem między tymi znakami.

```
if brak1 then - czy zmienna brak1 jest ustawiona w pozycji true?
begin - jeśli tak, to...
     brak:=false; - zainicjuj zmienną brak wartością false
     new(pi3); - utwórz w pamięci strukturę zamykającą
     if pi2<>nil then pi2^.prawa:=pi3; - jeśli istnieje funkcja...
          podpierwiastkowa, związ jej ostatni element z nowym elementem...
          wiązaniem głównym z prawej strony
     if pi4<>nil then - czy istnieje element spoza otoczenia pierwiastka?
```

```
        begin - jeśli tak, to...
              pi4^.lewa:=pi3; - wpisz w jego pole lewa adres struktury zamykającej
              if pi4^.ly=adres^.ly then brak:=true - jeśli znak spoza...
            otoczenia pierwiastka i znak pierwiastkowania są na tym samym poziomie,...
            ustaw zmienną brak w pozycji true
        end;
        adres^.kur_p:=pi3; - wpisz w pole kur_p elementu znaku...
         pierwiastkowania adres nowego elementu – struktury zamykajacej
        with pi3^ do
        begin
              prawa:=pi4; - związ wiązaniem głównym z prawej strony...
                            element spoza pierwiastka
              lewa:=pi2; - związ wiązaniem głównym z lewej strony...
                           ostatni element funkcji podpierwiastkowej...
                           (jeśli nie istnieje funkcja podpierwiastkowa,...
                           w pole lewa wpisany zostaje adres pusty)
              if brak then - czy zmienna logiczna brak została ustawiona...
                             w pozycji true?
              begin - jeśli tak, to...
                    kur_p:=pi4;      } związ wzajemnie wiązaniem kursorowym...
                    pi4^.kur_l:=pi3; } element spoza pierwiastka z nowym elementem
              end else kur_p:=nil; - w przeciwnym przypadku, w pole kur_p...
                                     wpisz identyfikator adresu pustego
              kur_l:=adres; - związ element znaku pierwiastkowania z nowym...
                              elementem wiązaniem kursorowym z lewej strony
              kur_g:=nil;
              kur_d:=nil;
              lx:=adres^.lx;
              px:=lx;
              ly:=adres^.ly;  } wpisz w pola współrzędnych wartości współrzędnych..
              py:=adres^.py;  } pochodzących z elementu znaku pierwiastkowania
              y1:=ly;
              y2:=py;
              znak:=' '; - w pole znak wpisz spację
              druk:=true - ustaw widoczność znaku
              zazn:=false - znak nie jest zaznaczony
        end;
end;
```

Gdy wskaźnik `pi2` posiada adres pusty, oznacza to brak przynajmniej jednego elementu należącego do funkcji podpierwiastkowej, dlatego brakujący składnik zostaje utworzony i związany z lewej strony wiązaniem głównym z elementem stopnia pierwiastka a z prawej strony ze strukturą zamykającą. Spośród wiązań kursorowych, tylko w pole `kur_g` wpisany zostaje adres elementu znaku pierwiastkowania, pozostałe pola zainicjowane zostają adresami pustymi. Wartości pól współrzędnych pochodzą z pól współrzędnych elementu znaku pierwiastkowania. W pole `znak` wpisany zostaje nawias otwierający.

```
if pi2=nil then - czy wskaźnik pi2 posiada adres pusty?
begin - jeśli tak, to...
      new(pi2); - utwórz w pamięci nowy składnik lisc – element funkcji...
                  podpierwiastkowej
      . . .
      if adres=wsb then wsb:=pi2; - jeśli główny wskaźnik edytora wsb,...
                                    posiada adres elementu znaku pierwiastkowania,...
                                    wpisz mu adres nowego elementu
```

```
        n1:=1; - wpisz do zmiennej n1 liczbę jeden (zliczenie nawiasu otwierającego)
        Form2.Szerokosc(pi2) - zmodyfikuj współrzędną poziomą px stosownie do...
         wyliczonej przez procedurę Szerokosc, szerokości znaku nawiasu otwierającego
end else pi2^.kur_p:=nil; - jeśli wskaźnik pi2 posiada adres elementu lisc,..
                            wpisz w pole kur_p identyfikator adresu pustego,...
                            zamykając funkcję podpierwiastkową z prawej strony
```

Pewnym ułatwieniem dla użytkownika jest ustawienie kursora na elemencie struktury zamykającej po wprowadzeniu przez niego znaku nawiasu zamykającego na końcu funkcji podpierwiastkowej. Pozwala to na pominięcie dodatkowej czynności związanej z ustawieniem kursora za pierwiastkiem – warunkiem jest zerowa wartość licznika nawiasów.

```
if n1=0 then - czy liczba nawiasów otwierających i zamykających jest zgodna?
   if pi2=wsb then - jeśli tak, to czy wskaźnik edytora wsb posiada adres...
                              ostatniego elementu funkcji podpierwiastkowej?
   begin - jeśli tak, to...
        wsb:=pi3; - wpisz do wskaźnika wsb adres struktury zamykającej
        stat_kur:=false - ustaw kursor z lewej strony znaku
   end;
```

Skoro funkcja podpierwiastkowa na pewno istnieje, można już związać jej elementy z elementem znaku pierwiastkowania w ten sposób, by można było poruszać się kursorem w górę i w dół pomiędzy znakami funkcji podpierwiastkowej a znakiem pierwiastkowania. Wiązanie to odbywa się w pętli `repeat..until`, w której dodatkowo szukane są: minimalna i maksymalna współrzędna pionowa wśród elementów funkcji podpierwiastkowej – wartości te będą niezbędne do wymiarowania pierwiastka. Sama idea wiązania jest porównywalna z wiązaniem elementu kreski ułamkowej z elementami mianownika w procedurze `dzielenie`. Tu również informacji logicznej o możliwości związania elementu znaku pierwiastkowania ze wskazanym elementem w pętli dostarcza funkcja `wiazania`.

Nim pętla zostanie uruchomiona, skrajne nawiasy funkcji podpierwiastkowej zostają ukryte, oczywiście w celach estetycznych.

```
pi4:=pi1^.prawa; - do zmiennej pi4 wpisz adres pierwszego elementu funkcji...
                              podpierwiastkowej
ukryj_nawiasy(pi4,pi2); - ukryj skrajną parę nawiasów należących do funkcji...
                                    podpierwiastkowej
brak:=false; - ustaw zmienną brak w pozycji false
s1:=maxint; - do zmiennej s1 wczytaj maksymalną wartość dla typu integer
s2:=-maxint; - do zmiennej s2 wczytaj minimalną wartość dla typu integer
repeat
     if not wiazanie(pi4,adres,true) then pi4^.kur_g:=adres;
      jeśli funkcja wiazania zwróciła wartość false, związ element...
      znaku pierwiastkowania z elementem wskazywanym przez...
      wskaźnik pi4, polem kur_g
     if not brak then - czy zmienna brak jest ustawiona w pozycji false?
        if pi4^.druk then - jeśli tak, to czy znak wpisany w pole znak...
                elementu wskazywanego przez wskaźnik pi4, nie jest ukryty?
           if pi4^.kur_g=adres then - jeśli znak nie jest ukryty, to czy...
                                      element ten jest już związany z elementem...
                                      znaku pierwiastkowania?
           begin - jeśli jest związany, to...
                adres^.kur_d:=pi4; - związ ten element z elementem...
                                     znaku pierwiastkowania, polem kur_d
```

```
            brak:=true - ustaw zmienną brak w pozycji true
        end;
        n1:=pi4^.y1; - wpisz do zmiennej n1 wartość górnej...
                       współrzędnej pionowej, z pola y1

        if s1>n1 then s1:=n1; - jeśli liczba zawarta w zmiennej s1 jest...
                                większa od odczytanej współrzędnej, wpisz do...
                                zmiennej s1 wartość tej współrzędnej

        n1:=pi4^.y2; - wpisz do zmiennej n1 wartość dolnej...
                       współrzędnej pionowej, z pola y2 elementu funkcji...
                       podpierwiastkowej, aktualnie analizowanej w pętli

        if s2<n1 then s2:=n1; - jeśli liczba zawarta w zmiennej s2 jest...
                                mniejsza od odczytanej współrzędnej, wpisz...
                                do zmiennej s2 wartość tej współrzędnej

        pi4:=pi4^.prawa - w zmiennej pi4 pozyskaj adres następnika...
                          w wiązaniu głównym
until pi4=pi3; - opuść pętlę, gdy wskaźnik pi4 uzyska adres struktury zamykającej
```

Po opuszczeniu pętli, zmienne s1 i s2 powinny posiadać wartości skrajnych współrzędnych pionowych, odnoszących się do funkcji podpierwiastkowej. Na ich podstawie wyliczona zostaje wysokość znaku pierwiastkowania, która jest sumą wysokości funkcji podpierwiastkowej, czyli różnicy wartości zawartych w zmiennych s1 i s2, i minimalnej odległości znaku od kreski pierwiastkowej, czyli wartości znajdującej się w zmiennej globalnej KR bloku regulacji. Jeśli wyliczona wysokość różni się od bieżącej wysokości znaku pierwiastkowania, znak ten zostaje dopasowany do wyliczonej wartości. Polega to na określeniu różnicy wysokości znaku pierwiastkowania z wyliczoną oraz symetrycznej zmianie wartości współrzędnych pionowych rzeczywistych elementu znaku pierwiastkowania, pozostawiając niezmienioną pozycję tego znaku względem sąsiadujących znaków z lewej i prawej strony. Zastosowana regulacja pozwala zachować niezmieniony poziom znaku pierwiastkowania przy możliwych zmianach zawartości funkcji podpierwiastkowej.

```
k2:=adres^.y2-adres^.y1; - wylicz wysokość rzeczywistą elementu znaku...
                           pierwiastkowania i zapamiętaj ją w zmiennej k2

n1:=(s2-s1)+Form6.KR; - wylicz i zapamiętaj w zmiennej n1 wysokość rzeczywistą...
                        funkcji podpierwiastkowej, uwzględniając odległość od kreski pierwiastkowej

if k2<>n1 then - czy obydwie odległości różnią się?
begin - jeśli tak, to...
     n1:=n1-k2; - wylicz różnicę obu wysokości i zapamiętaj ją w zmiennej n1
     k2:=trunc(n1/2); - wylicz połowę tej różnicy i zapamiętaj ją w zmiennej k2
     n1:=adres^.y1-k2; - wylicz nową wartość górnej współrzędnej...
                         dla znaku pierwiastka

     k2:=adres^.y2+k2; - wylicz nową wartość dolnej współrzędnej...
                         dla znaku pierwiastka

     adres^.y1:=n1; - wpisz w pole y1 elementu znaku pierwiastkowania...
                      wartość nowej współrzędnej
```

```
        adres^.y2:=k2 - wpisz w pole y2 elementu znaku pierwiastkowania...
                        wartość nowej współrzędnej
end;
```

Mimo dopasowania znaku pierwiastka do funkcji podpierwiastkowej, funkcja ta może wymagać niewielkiego dopasowania do górnej współrzędnej znaku pierwiastka, dlatego powtórnie wyliczana jest różnica między minimalną współrzędną funkcji podpierwiastkowej, zapamiętanej w zmiennej `s1` oraz sumą górnej współrzędnej znaku pierwiastkowania `y1` i odległości od kreski pierwiastkowej. Ewentualna różnica powoduje uruchomienie pętli `repeat..until`, w której poprawiane są pola współrzędnych pionowych wszystkich elementów funkcji podpierwiastkowej. Nowa, maksymalna współrzędna pionowa badanych elementów funkcji, zawarta w zmiennej `s2`, jest jednocześnie nową współrzędna `y2` elementu znaku pierwiastkowania.

```
s1:=adres^.y1+Form6.KR-s1; - wylicz różnicę, między górną współrzędną...
                             elementu znaku pierwiastkowania a sumą...
                             minimalnej współrzędnej funkcji...
                             podpierwiastkowej i odległości od...
                             kreski pierwiastkowej

pi2:=pi1^.prawa; - wpisz do zmiennej pi2 adres pierwszego elementu...
                   funkcji podpierwiastkowej

if s1<>0 then - czy wyliczona różnica jest różna od zera?
begin - jeśli tak, to...
     s2:=-maxint; - wpisz do zmiennej s2 minimalną wartość dla typu integer
     pi4:=pi2; - zachowaj w zmiennej pi4 adres pierwszego elementu...
                 funkcji podpierwiastkowej
     repeat
          with pi4^ do
          begin
               ly:=ly+s1;
               py:=py+s1;   } zmodyfikuj współrzędne pionowe badanego...
               y1:=y1+s1;   } elementu w pętli o wyliczoną różnicę
               y2:=y2+s1
          end;
          if pi4^.y2>s2 then s2:=pi4^.y2; - jeśli wartość dolnej...
                             współrzędnej z pola y2 badanego elementu...
                             jest większa od wartości w zmiennej s2,...
                             wpisz do tej zmiennej wartość współrzędnej

          pi4:=pi4^.prawa; - w zmiennej pi4 pozyskaj adres następnika...
                             w wiązaniu głównym

     until (pi4=pi3) or (pi4=nil); - opuść pętlę, gdy zmienna pi4...
                                     uzyska adres struktury zamykającej lub pusty

     adres^.y2:=s2 - wpisz do pola y2 elementu znaku pierwiastkowania...
                     wartość nowej współrzędnej
end;
```

Element znaku pierwiastkowania oraz składniki należące do funkcji podpierwiastkowej mają już poprawne współrzędne pionowe. Można już zwymiarować element stopnia pierwiastka – jego dolna współrzędna pionowa równa jest różnicy dolnej współrzędnej rzeczywistej znaku pierwiastkowania i połowy jego wysokości (zmienna `s2`) oraz odległości od kreski ułamkowej, której wartość znajduje się w zmiennej `KR` należącej do modułu regulacji, zaś górna współrzędna równa jest różnicy jego dolnej współrzędnej oraz połowy wysokości znaków, czyli wartości znajdującej się w zmiennej globalnej `RZ` modułu regulacji. Wysokość

stopnia pierwiastka jest więc niezależna od wysokości znaku pierwiastkowania. Współrzędne poziome elementu stopnia pierwiastka są wynikiem różnicy przesunięcia początkowej współrzędnej `lx` elementu znaku pierwiastkowania o 1/8 jego wysokości oraz sumy dotychczasowej wartości współrzędnej `lx` elementu stopnia pierwiastka i ¼ jego szerokości.

```
s2:=trunc((adres^.y2- adres^.y1)/2); - wylicz połowę wysokości znaku...
                                       pierwiastkowania – wynik zapamiętaj w zmiennej s2

with pi1^ do - instrukcja wiążąca dla elementu stopnia pierwiastka
begin
     lx:=adres^.lx; - wpisz do pola lx wartość z pola lx elementu...
                      znaku pierwiastkowania

     ly:=adres^.y1; - wpisz do pola ly wartość z pola y1 elementu...
                      znaku pierwiastkowania

     py:=adres^.y2-s2-Form6.KR; - wpisz do pola py współrzędną pionową,...
                                  znajdująca się w połowie między współrzędnymi...
                                  y1 a y2 elementu znaku pierwiastkowania,...
                                  pomniejszoną o odległość znaku...
                                  od kreski pierwiastkowej

     y1:=ly; } współrzędne rzeczywiste y1 i y2 są identyczne...
     y2:=py  } jak współrzędne poziomu ly i py
end;
Form2.Szerokosc(pi1); - zmodyfikuj współrzędną poziomą px elementu stopnia...
                        pierwiastka na podstawie szerokości znaku wpisanego w pole znak

n1:=adres^.lx+trunc(s2/4); - wylicz sumę współrzędnej lx elementu znaku...
                             pierwiastkowania i ¼ połowy jego wysokości

n1:=n1-(pi1^.lx+trunc((pi1^.px-pi1^.lx)/4)); - wylicz różnicę...
                               między dotychczasową współrzędną poziomą lx...
                               elementu stopnia pierwiastka a wymaganą

if n1<>0 then - czy wyliczona różnica jest różna od zera?
begin - jeśli tak, to...
     pi1^.lx:=pi1^.lx+n1; } zmodyfikuj współrzędne poziome elementu...
     pi1^.px:=pi1^.px+n1  } stopnia pierwiastka o wyliczoną różnicę
end;
```

Ważną częścią wymiarowania pierwiastka jest wyliczenie współrzędnej poziomej, od której będzie rozpoczynała się funkcja podpierwiastkowa. Jest ona równa sumie współrzędnej poziomej zawartej w polu `lx` elementu znaku pierwiastkowania i połowie wysokości znaku pierwiastka, która zapamiętana została w zmiennej `s2`. Różnica między wyliczoną wartością a współrzędną, zawartą w polu `lx` pierwszego elementu funkcji podpierwiastkowej, wskazuje na potrzebę przesunięcia elementów tej funkcji o wyliczoną różnicę.

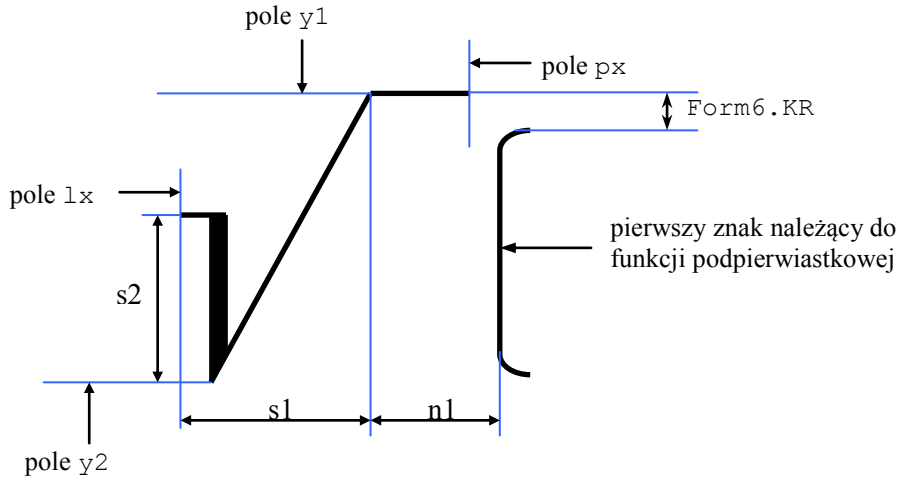
```
s1:=adres^.lx+s2; - zapamiętaj w zmiennej s1 sumę współrzędnej poziomej,...
  pochodzącej z pola lx elementu znaku pierwiastkowania i połowy jego wysokości

n1:=s1-pi2^.lx; - wylicz różnicę między wymaganą wartością współrzędnej...
 a rzeczywistą, zawartą w polu lx pierwszego elementu funkcji podpierwiastkowej

if n1<>0 then - czy wyliczona różnica jest różna od zera?
begin - jeśli tak, to...
     k2:=pi2^.px-pi2^.lx; - wylicz szerokość znaku zawartego w pierwszym...
                            elemencie funkcji podpierwiastkowej i zapamiętaj ją w zmiennej k2

     pi2^.lx:=pi2^.lx+n1; - zmodyfikuj pole lx o różnicę zawartą w zmiennej n1
     pi2^.px:=pi2^.lx+k2; - wpisz w pole px sumę wartości poprawionego pola...
                            lx i szerokości znaku
```

```
        Form2.Przesun(pi2,true) - przesuń w poziomie pozostałe elementy funkcji...
                                  podpierwiastkowej względem poprawionego, pierwszego elementu tej funkcji
end;
```

Funkcja podpierwiastkowa jest już na właściwym miejscu, można już określić szerokość znaku pierwiastkowania. Polega to na wyliczeniu końcowej wartości współrzędnej poziomej dla tego znaku, która jest sumą początkowej współrzędnej poziomej funkcji podpierwiastkowej, zapamiętanej w zmiennej `s1` oraz szerokości tej funkcji. Wyliczona suma zostaje wpisana w pole `px` elementu znaku pierwiastkowania.

```
adres^.px:=s1+dlugosc(pi2,pi3); - wpisz w pole px elementu znaku...
                                  pierwiastkowania sumę wartości w zmiennej s1 oraz długości funkcji podpierwiastkowej
```

Sposób wykonania obliczeń modyfikujących współrzędne elementu znaku pierwiastkowania może wydać się mało czytelny. Pomocnym w zrozumieniu tych obliczeń jest rysunek 49, wyjaśniający znaczenie pól współrzędnych elementu znaku pierwiastkowania w odniesieniu do samego znaku oraz zmiennych zastosowanych przy obliczeniach.



Rys. 49. Zasada wymiarowania znaku pierwiastkowania

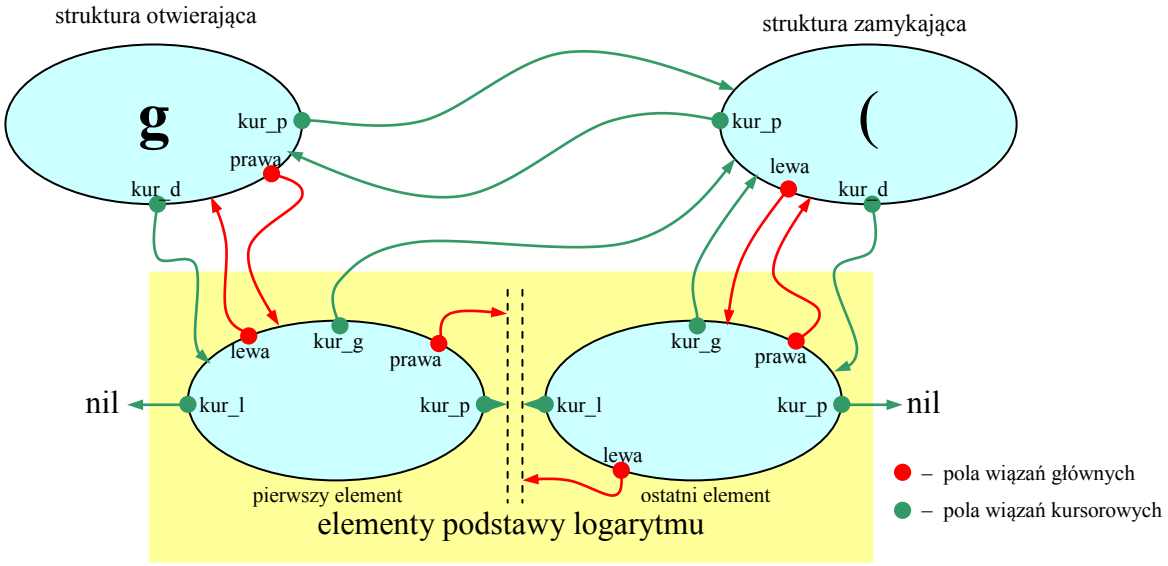
Znak pierwiastka oraz jego funkcja podpierwiastkowa są już zwymiarowane. Pozostało jeszcze dopasowanie do znaku pierwiastka struktury zamykającej i pozostałych elementów znajdujących się za tą strukturą, związanych ze sobą wiązaniem kursorowym. W tym celu porównywane są ze sobą dwie wartości współrzędnych poziomych: pole `px` elementu znaku pierwiastkowania oraz `lx` struktury zamykającej. Jeśli obie współrzędne różnią się, współrzędne poziome struktury zamykającej i pozostałych składników listy, związanych wiązaniem kursorowym, zostają zmodyfikowane o tę różnicę.

```
pi3^.kur_l:=adres;  } związ wzajemnie wiązaniem kursorowym element znaku...
adres^.kur_p:=pi3;  } pierwiastkowania i strukturę zamykającą
```

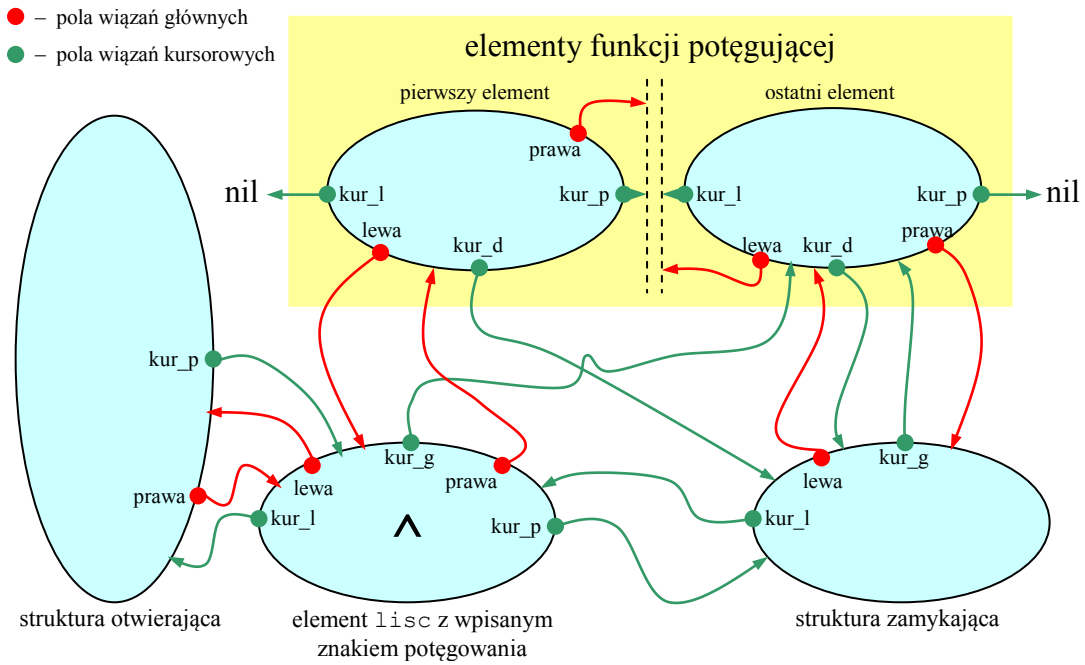
```
s1:=adres^.px-pi3^.lx; - wylicz w zmiennej s1 różnicę między współrzędnymi:...
                         px elementu znaku pierwiastkowania a lx struktury zamykającej
if s1<>0 then - czy wartość w zmiennej s1 jest różna od zera?
begin - jeśli tak, to...
      s2:=pi3^.px-pi3^.lx; - wylicz szerokość znaku struktury zamykającej
      pi3^.lx:=pi3^.lx+s1; - zmodyfikuj pole lx o różnicę zawartą w zmiennej s2
      pi3^.px:=pi3^.lx+s2; - wpisz w pole px sumę wartości poprawionego pola...
                             lx i szerokości znaku

      Form2.Przesun(pi3,true) - przesuń w poziomie pozostałe składniki listy...
                                względem elementu, dostępnego przez wskaźnik pi3
end;
```

XIV. 2.3. Ułożenie funkcji potęgującej lub logarytmu – procedura `potelog`

Procedura ta zajmuje się przypadkiem potęgowania oraz logarytmu o dowolnej podstawie. Zadanie jej polega na prawidłowym wyznaczeniu otoczenia elementów związanych z powyższymi przypadkami i właściwym ich związaniu, które musi być zgodne z poniższymi schematami (rysunki 50-51).



Rys. 50. Struktura logiczna wiązania elementów logarytmu o dowolnej podstawie



Rys. 51. Struktura logiczna wiązania elementów funkcji potęgującej

O tym, jakim przypadkiem procedura ma się zająć, informuje ją drugi dostarczony parametr o nazwie `tryb4` – jeśli jego wartość wynosi `true`, procedura zajmuje się potęgowaniem, w przeciwnym przypadku – logarytmem. Wybór podwójnej roli procedury wynika ze zbieżności wielu elementów kodu dla obu przypadków. Są jednak instrukcje, które należą tylko do potęgowania – jedną z nich jest kontrola obecności struktury otwierającej – jeśli brakuje tej struktury, musi ona zostać utworzona i odpowiednio związana zarówno z bieżącym elementem wskazywanym przez wskaźnik `adres4`, jak i ewentualnym jego poprzednikiem.

```
procedure potelog(const adres4:lisc;tryb4:boolean);
const liter3=[44,46,48..57,101,112]; tablica stałych składająca się z...
                    wartości znaków przecinka, kropki, cyfr oraz liter ‘e’ i ‘π’
var pl1,pl2,pl3,pl4,pom3:lisc;
    brak_l,brak_p:boolean;
    po1,po2,po3:integer;
begin
     po2:=0;
     if tryb4 then – czy parametr tryb4 jest ustawiony w pozycji true, która...
                     sugeruje, że rozpatrywanym znakiem jest potęgowanie?

     begin – jeśli tak, to...
          if adres4^.druk then – czy znak potęgowania nie został ukryty?
          begin – jeśli nie został ukryty, to...
               adres4^.druk:=false; – ukryj znak potęgowania przez...
                                      ustawienie pola druk w pozycji false

               adres4^.px:=adres4^.lx – zredukuj szerokość znaku do zera
          end;
          po2:=adres4^.lx; – zapamiętaj początkową pozycję znaku w osi X...
                             będzie to początkowa pozycja funkcji potęgującej

          brak_l:=false;
```

```
                    pl1:=adres4^.kur_l; - wczytaj adres poprzednika ...
                                          w wiązaniu kursorowym
                    if pl1=nil then brak_l:=true - jeśli poprzednik nie istnieje...
                                   ustaw zmienną logiczną brak_l w pozycji true
                    else - jeśli poprzednik istnieje, to...
                    begin
                         po1:=ord(pl1^.znak); - odczytaj wartość znaku poprzednika
                         if po1 in zam then - czy znak ten należy do tablicy znaków...
                                              nawiasów zamykających?
                            if po1<>93 then - jeśli tak, to czy nie jest to znak nawiasu...
                                              kwadratowego?
                            begin - jeśli nie jest to nawias kwadratowy, to...
                                 po1:=trunc((pl1^.px-pl1^.lx)/4); - wylicz...
                                                korektę przesunięcia równą ¼ szerokości...
                                                nawiasu okrągłego lub sześciennego
                                 po2:=po2-po1 - skoryguj początkową pozycję funkcji
...
                                                potęgującej o wyliczoną korektę
                            end
                    end;
                    if brak_l then - czy brakuje struktury otwierającej?
                    begin - jeśli tak, to...
                         pl2:=adres4^.lewa; - we wskaźniku pl2 zapamiętaj adres ...
                                 poprzednika w wiązaniu głównym bieżącego elementu
                         new(pl3); - utwórz w pamięci nowy element lisc...
                                     (struktura otwierająca)
                         if pl2<>nil then pl2^.prawa:=pl3 else wsp:=pl3;
                            jeśli poprzednik bieżącej struktury w wiązaniu głównym istnieje...
                            zwiąż go z nowoutworzonym elementem, w przeciwnym...
                            przypadku nowy element jest pierwszym,...
                            dlatego zapamiętaj jego adres w zmiennej wsp
                         adres4^.lewa:=pl3;
                         adres4^.kur_l:=pl3;
                         with pl3^ do
                         begin
                              lewa:=pl2;
                              kur_l:=nil;
                              prawa:=adres4;
                              kur_p:=adres4;
                              kur_g:=nil;
                              kur_d:=nil;
                              lx:=adres4^.lx;
                              px:=lx;
                              ly:=adres4^.ly;
                              py:=adres4^.py;
                              y1:=ly;
                              y2:=py;
                              znak:=' ';
                              druk:=true
                         end;
                    end;
               end;
```

Po uzupełnieniu ewentualnego braku struktury otwierającej, procedurę czeka dość trudne zadanie polegające na określeniu funkcji potęgującej lub podstawy logarytmu. Zadanie to wykonane zostanie w pętli `while..do`, ale przedtem należy określić, czy struktura zamyka-

jąca została już utworzona oraz czy istnieje następnik bieżącego elementu lisc, którego adres istnieje w zmiennej adres4. Jeśli następnik istnieje, to czy istnieje także struktura zamykająca. Informację o istnieniu struktury zamykającej udzieli wynik porównania dwóch składników lisc związanych z bieżącym elementem w wiązaniu kursorowym i głównym. Jeśli obydwa elementy mają ten sam adres, oznacza to, że struktura zamykająca nie została jeszcze utworzona, dlatego by ją utworzyć, zmienna brak_p zostaje ustawiona w pozycji true – jest to informacja dla dalszej części programu, w której znajdują się instrukcje tworzenia brakujących składników lisc.

```
brak_l:=false;
brak_p:=false; - zainicjowanie zmiennej logicznej brak_p wartością false
pl1:=nil; - zmienna pomocnicza pl1 zainicjowana wartością pustą
pl2:=adres4^.prawa; - pozyskanie adresu następnika w wiązaniu głównym
pl4:=adres4^.kur_p; - pozyskanie adresu następnika w wiązaniu kursorowym
if pl4<>nil then - czy następnik w wiązaniu kursorowym istnieje?
begin - jeśli tak, to...
     if pl4=pl2 then - czy adresy następników: w wiązaniu kursorowym...
                       i głównym wskazują, że to ten sam element?
     begin - jeśli tak, to...
          pl1:=pl4; - przypisz zmiennej ad1 adres następnika
          brak_p:=true; - ustaw zmienną brak_p w pozycji true...
          informującej o braku elementu funkcji potęgującej lub podstawy logarytmu
     end else
```

Jeśli jednak adresy obu elementów różnią się, struktura zamykająca, której adres znajduje się we wskaźniku pl4, na pewno istnieje. W tej sytuacji konieczne jest wyznaczenie funkcji potęgującej lub podstawy logarytmu. Obecność struktury zamykającej daje pewność, że obecny jest także następnik w wiązaniu głównym, co oznacza, że wskaźnik pl2 nie posiada adresu pustego. Przed uruchomieniem pętli, adres następnika zostaje przekazany wskaźnikowi pl1, a po jego przepisaniu, wskaźnik pl2 zostaje zainicjowany adresem pustym. Po wyzerowaniu licznika nawiasów, czyli zmiennej po1, pętla while..do zostaje uruchomiona. Pierwszą instrukcją w pętli jest sczytanie znaku z elementu, na który aktualnie wskazuje wskaźnik pl1. Odczytany znak zostaje przekonwertowany do wartości w kodzie ASCII, co ułatwia identyfikowanie tych znaków.

Jeśli procedura ma zajmować się logarytmem, każdy sczytany znak jest porównywany z tablicą stałą o nazwie liter3. Jeśli aktualnie analizowany znak nie zawiera się w tej tablicy, musi zostać usunięty. Odbywa się to w ten sposób, że gdy jest to jedyny element podstawy logarytmu, znak wpisany w jego pole znak zostaje zastąpiony spacją a szerokość znaku zredukowana jest do zera. Gdy nie jest to jedyny znak podstawy logarytmu, elementy przed i za elementem wskazywanym przez wskaźnik pl1, są ze sobą związane a on sam zostaje usunięty z pamięci. Wyjątkiem jest znak nawiasu otwierającego, gdy nie istnieje następnik w wiązaniu kursorowym. W takiej sytuacji pętla zostaje opuszczona instrukcją break, by nawias ten znalazł się poza podstawą logarytmu. By pętla mogła być kontynuowana bezbłędnie, wskaźnik pl1 oraz główny wskaźnik edytora – wsb, otrzymują nowe adresy od poprzednika lub następnika usuniętego elementu.

W pętli zliczane są nawiasy w ten sposób, że jeśli obecnie analizowany składnik lisc posiada wpisany znak nawiasu otwierającego, zmienna po1 zwiększa swoją wartość o jeden, natomiast dla znaku nawiasu zamykającego, zmniejsza ją o jeden. Jeśli wykryty zostanie znak nawiasu otwierającego i nie jest to pierwszy element funkcji potęgującej, o czym informuje niepusty adres poprzednika we wskaźniku pl2, sprawdzany jest znak tego poprzednika – jeśli zawiera się w tablicy stałej o nazwie liter3, zawierającej wartości znaków cyfr oraz

gdy poprzedni element jest związany z bieżącym elementem w pętli wiązaniem kursorowym, następuje opuszczenie pętli. Opisany zestaw warunków pozwala na zachowanie w funkcji trygonometrycznej potęgowania umieszczonego między jej rdzeniem a pierwszym nawiasem otwierającym. Brak związania kursorowego z poprzednikiem może wskazywać na kompletne wyznaczenie i właściwe związanie elementów potęgowania podczas poprzedniego wywołania procedury, dlatego nie spełnienie tego warunku umożliwia kontynuowanie pętli. Jeśli licznik nawiasów będzie miał wartość mniejszą lub równą zero, wykrycie elementu z wpisanym znakiem mnożenia powoduje opuszczenie pętli. Z kolei wykrycie znaku dzielenia, przy dodatkowym warunku, że poprzednik w wiązaniu głównym oraz w wiązaniu kursorowym mają taki sam adres, także spowoduje opuszczenie pętli. Element ze znakiem dzielenia staje się wówczas strukturą zamykającą dla bieżącego elementu ze znakiem potęgowania. By procedura `dzielenie` mogła prawidłowo wyznaczyć licznik ułamka, przed elementem ze znakiem dzielenia, musi znaleźć się jego poprzednik, a więc dodatkowy element, który musi być z nim związany zarówno wiązaniem głównym, jak i kursorowym. Brak dodatkowego elementu spowodowałby niepoprawne wyznaczenie licznika ułamka. Gdyby natomiast, po wykryciu znaku dzielenia, pominięty został drugi warunek, podczas cyklicznych wywołań omawianej procedury dochodziłoby do każdorazowego lokowania w pamięci nowego elementu, doprowadzając tym do załamanie się programu. Wykrycie znaku dodawania lub odejmowania – przy dodatkowym warunku, że poprzedni element, dostępny poprzez wskaźnik `pl2` nie posiada znaku mnożenia, dzielenia lub potęgowania – powoduje opuszczenie pętli. To z kolei nie pozwala na zachowanie w funkcji potęgującej nie zamkniętego w nawiasie znaku działania, wyjątkiem jest zachowanie znaku odejmowania lub dodawania rozpoczynającego np. funkcję potęgującą. Gdy licznik nawiasów będzie miał wartość mniejszą od zera, pętla musi zostać bezwzględnie opuszczona, gdyż zidentyfikowany w pętli znak nawiasu zamykającego na pewno nie należy do obecnie analizowanej funkcji potęgującej. Tuż przed wczytaniem do zmiennej sterującej pętlą adresu następnika, do wskaźnika `pl2` wczytany zostaje adres bieżącego składnika `lisc`, który analizowany był w pętli.

Jeżeli pętla nie zostanie opuszczona instrukcją `break`, będzie się wykonywać tak długo, aż do zmiennej sterującej pętlą, czyli do wskaźnika `pl1`, zostanie wczytany adres struktury zamykającej, dostępnej we wskaźniku `pl4`.

```
if pl4<>nil then - czy istnieje następnik w wiązaniu kursorowym elementu...
                   wskazywanego przez parametr o nazwie adres4?
   if pl4=pl2 then - jeśli tak, to czy następnik ten jest równy następnikowi...
                     parametru adres4 w wiązaniu głównym?
   begin - jeśli tak, to...
        pl1:=pl4; - przypisz adres następnika wskaźnikowi pl1
        brak_p:=true - zaznacz brak funkcji potęgującej lub podstawy logarytmu
   end else - jeśli następniki: w wiązaniu głównym i kursorowym, są różne, to...
   begin
        pl1:=pl2; - przepisz adres następnika do zmiennej sterującej pętlą
        pl2:=nil; - wskaźnikowi, który po opuszczeniu pętli będzie zawierał adres...
                    ostatniego elementu funkcji potęgującej/podstawy logarytmu,...
                    przypisz na początek adres pusty
        po1:=0; - wyzeruj licznik nawiasów
        while pl1<>pl4 do wykonaj krok pętli, póki wskaźnik pl1 nie ma adresu pl4
        begin
             po3:=ord(pl1^.znak); - odczytaj wartość znaku z bieżącego...
                                    elementu lisc
             if not tryb4 then - czy wartość false parametru tryb4 nakazuje...
                                 procedurze zajmować się logarytmem?
```

```
if po3 in liter3 then - jeśli tak, to czy wartość sczytanego...
                        znaku zawiera się w tablicy stałej liter3?

else - jeśli nie, to...
begin
     pl3:=pl1^.kur_l; - odczytaj adres poprzednika…
                        w wiązaniu kursorowym

     pom3:=pl1^.kur_p; - odczytaj adres następnika…
                         w wiązaniu kursorowym

     if pl3<>nil then - czy poprzednik istnieje?
        if pom3<>nil then - jeśli tak, to czy następnik istnieje?
        begin - jeśli tak, to...
             pl1^.px:=pl1^.lx; - zredukuj bieżącemu...
                                 elementowi szerokość znaku do zera

             Form3.Przesun(pl1,true); - dosuń...
                                        znaki usytuowane na prawo...
                                        w miejsce usuwanego znaku

             pl3^.prawa:=pom3;
             pom3^.lewa:=pl3;      } związ elementy
             pl3^.kur_p:=pom3;     } ze sobą
             pom3^.kur_l:=pl3;
             brak_p:=true - zaznacz potrzebę usunięcia...
                            elementu

        end else - gdy następnik nie istnieje, to...
            if po3=40 then break -jeśli wartość bieżącego...
                                  znaku wskazuje na nawias otwierający,...
                                  opuść pętlę

            else - dla każdego, innego znaku niż...
                   nawias otwierający...
            begin
                 pl3^.prawa:=pl4;
                 pl4^.lewa:=pl3;     } związ poprzednika...
                 pl3^.kur_p:=nil;    } z elementem struktury...
                 pl3^.kur_g:=pl4;    } zamykającej
                 pl4^.kur_d:=pl3;
                 brak_p:=true - zaznacz potrzebę…
                                usunięcia elementu

            end
     else - gdy poprzednik nie istnieje, to...
     if pom3<>nil then - czy istnieje następnik?
     begin - jeśli tak, to...
          pl1^.px:=pl1^.lx; - zredukuj bieżącemu...
                              elementowi szerokość znaku do zera

          Form3.Przesun(pl1,true); - dosuń...
                                     elementy usytuowane na prawo...
                                     w miejsce usuwanego znaku

          pom3^.lewa:=adres4;
          adres4^.prawa:=pom3;     } związ następnika...
          adres4^.kur_d:=pom3;     } z elementem struktury...
          pom3^.kur_g:=adres4;     } otwierającej
          pom3^.kur_l:=nil;
          brak_p:=true - zaznacz potrzebę usunięcia...
                         elementu

     end else - jeśli nie istnieją: poprzednik i następnik, to...
         if po3=40 then break - jeśli wartość bieżącego...
```

```
                                    znaku wskazuje na nawias otwierający,...
                                    opuść pętlę
              else - dla każdego, innego znaku niż nawias otwierający...
              begin
                   pl1^.znak:=' '; - wpisz do pola bieżącego...
                                     elementu znak spacji
                   pl1^.px:=pl1^.lx - zredukuj szerokość…
                                      znaku do zera
                   po3:=32; - do zmiennej po3 wpisz wartość w ...
                              kodzie ASCII znaku spacji
              end;
         if brak_p then - czy zmienna logiczna została ustawiona...
                          w pozycji true?
         begin - jeśli tak, to...
              dispose(pl1); - uwolnij pamięć zajmowaną przez...
                              element pl1
              if pl3<>nil then pl1:=pl3
              else pl1:=pom3; - przypisz wskaźnikowi pl1…
                        adres poprzednika lub gdy nie istnieje…
                       – następnika
              wsb:=pl1; - przypisz adres pl1...
                          głównemu wskaźnikowi edytora
              po3:=ord(pl1^.znak); do zmiennej po3 wczytaj...
                                   wartość znaku z elementu pl1
              brak_p:=false - przywróć zmiennej logicznej ...
         end;                 poprzednią wartość
   end;
if po3 in otw then - czy odczytany znak zawiera się w tablicy otw ...
                     zawierającej wartości znaków nawiasów otwierających?
begin - jeżeli tak, to...
     if pl2<>nil then - czy wskaźnik pl2 nie ma już adresu...
                        pustego?
        if ord(pl2^.znak) in liter3 then
                          jeśli tak, to czy...
                          poprzedni element lisc ma wpisany znak...
                          zawierający się w tablicy liter3?
           if pl1^.kur_l=pl2 then break;
                          jeśli tak, to jeśli...
                          element bieżący jest związany kursorowo...
                           z poprzednikiem, opuść pętlę
     inc(po1) - zwiększ wartość w liczniku nawiasów o jeden
end;
if po3 in zam then dec(po1); - jeśli odczytany znak zawiera się...
                       w tablicy zam zawierającej wartości znaków...
                       nawiasów zamykających, zmniejsz wartość...
                       w liczniku nawiasów o jeden
if po1<=0 then - czy licznik nawiasów ma wartość mniejszą lub równą...
                 zero?
begin - jeśli tak, to...
     if pl2<>nil then - czy wskaźnik pl2 nie ma adresu pustego?
     begin - jeśli tak, to...
          if po3=42 then break; - jeśli odczytanym znakiem...
                                  jest mnożenie, opuść pętlę
```

```
                    if po3=47 then - czy odczytany znak to dzielenie?
                        if pl2=pl1^.kur_l then break; jeśli tak, to...
                          jeśli poprzednik elementu pl1 w wiązaniu głównym...
                          i jego poprzednik w wiązaniu kursorowym...
                          mają ten sam adres, opuść pętlę
                    if (po3=43) or (po3=45) then - czy odczytanym...
                              znakiem jest dodawanie lub odejmowanie?
                    begin - jeśli tak, to..
                         po1:=ord(pl2^.znak); - odczytaj wartość…
                                                znaku poprzednika
                         if (po1<>42) and (po1<>47)
                              and (po1<>94) - jeśli w poprzednim...
                                     elemencie lisc nie ma znaku...
                                     mnożenia, dzielenia ani potęgowania, to...
                         then break - opuść pętlę
                         else po1:=0 - w przeciwnym przypadku, przypisz...
                          zmiennej po1 wartość zerową, by kontynuować pętlę...
                          przy zerowej wartości licznika nawiasów
                    end;
               end;
               if po1<0 then break; - jeśli licznik nawiasów ma wartość...
                                      mniejszą od zera, bezwzględnie opuść pętlę
          end;
          pl2:=pl1; - zapamiętaj adres bieżącego elementu lisc we…
                      wskaźniku pl2
          pl1:=pl1^.prawa - do zmiennej sterującej pętlą wczytaj adres…
                            następnika elementu lisc w wiązaniu głównym
end; - zakończenie pętli while..do
```

Po opuszczeniu pętli, we wskaźniku `pl2` będzie znajdował się adres ostatniego elementu należącego do funkcji potęgującej lub podstawy logarytmu, natomiast we wskaźniku `pl1` – adres składnika `lisc`, który może być nową strukturą zamykającą. By móc rozważać taką ewentualność, należy zbadać, czy określone w pętli elementy funkcji potęgującej należą w całości do otoczenia bieżącego elementu, czyli tego, wskazywanego przez parametr `adres4`. Pomocnym okazał się sposób wiązania składników `lisc` funkcji potęgującej w ten sposób, że pierwszy element tej funkcji jest z lewej strony zamknięty w wiązaniu kursorowym, dzięki czemu można uzyskać adres elementu z wpisanym znakiem potęgowania, będącego poprzednikiem pierwszego elementu funkcji potęgującej w wiązaniu głównym. Jeśli element ten będzie miał taki sam adres jak element dostępny przez wskaźnik `adres4` oraz nowa struktura zamykająca nie została jeszcze poprawnie związana, konieczna jest dalsza analiza.

```
pl2^.kur_p:=nil; - ostatni element funkcji potęgującej/podstawy logarytmu, zamknij...
                   z prawej strony w wiązaniu kursorowym,...
                   przypisując polu kur_p tego elementu adres pusty
pom3:=pl1; - wpisz wskaźnikowi pomocniczemu pom3 adres elementu, który już...
             nie należy do funkcji potęgującej/podstawy logarytmu
repeat
      pl3:=pom3; - zapamiętaj adres ze wskaźnika pom3
      pom3:=pom3^.kur_l - uzyskaj adres poprzednika w wiązaniu kursorowym
until pom3=nil; - opuść pętlę, gdy poprzednik nie istnieje
pl3:=pl3^.lewa; - pozyskaj adres poprzednika w wiązaniu głównym
if (pl3=adres4) and (pl1<>pl4) then - jeśli uzyskany adres we wskaźniku...
```

pl3 jest taki sam jak adres we wskaźniku adres4 (to ten sam element)...
oraz czy adres zawarty we wskaźniku pl1 jest inny niż we wskaźniku pl4...
wskazującym na dotychczasową strukturą zamykającą, to...

W kolejnych instrukcjach warunkowych ustalana jest potrzeba utworzenia dodatkowego składnika `lisc`. Odbywa się to na podstawie znaku wpisanego w pole elementu dostępnego przez wskaźnik `pl1` oraz w pole jego poprzednika w wiązaniu kursorowym. Jeśli w pierwszym przypadku jest to znak dzielenia, dodatkowy element jest konieczny, ponieważ dzięki niemu, wszystkie elementy funkcji potęgującej znajdą się w liczniku ułamka. W drugim przypadku, gdy poprzednik ma wpisany znak akcji (dzielenia, potęgowania, pierwiastkowania), element dostępny przez wskaźnik `pl1` byłby dla poprzednika strukturą zamykającą, dlatego i w tym przypadku dodatkowy element jest nieodzowny. W obu przypadkach, sposób związania nowego elementu z już istniejącymi będzie się różnił, w zależności od tego, jaki tym znaku akcji wpisany został w pole `znak` elementu wskazywanego przez wskaźnik `pl1`, dlatego do rozwiania wszelkich wątpliwości wykorzystana została zmienna liczbowa `po1`, która po spełnieniu jednego z warunków przyjmie wartość jeden lub dwa.

```
if (pl3=adres4) and (pl1<>pl4) then
begin
     po1:=0; - zainicjowanie zmiennej po1 wartością zero
     pl3:=pl1^.kur_l; - pozyskanie adresu poprzednika elementu pl1...
                        w wiązaniu kursorowym

     if pl3<>nil then - czy poprzednik istnieje?
        if ord(pl3^.znak) in dzed then - jeśli tak, to czy znak...
                        wpisany w jego pole należy do znaków akcji?

           if pl3<>adres4 then po1:=1; - jeśli tak, to jeśli poprzednik...
                                         ten jest różny od elementu bieżącego,...
                                         wpisz do zmiennej po1 liczbę jeden

     if pl1^.znak='/' then po1:=2; - jeśli znak wpisany w pole elementu...
                        pl1 jest dzieleniem, wpisz do zmiennej po1 liczbę dwa
```

W kolejnej instrukcji warunkowej, gdy wartość w zmiennej `po1` jest większa od zera, lokowany jest w pamięci nowy składnik `lisc`.

```
if (pl3=adres4) and (pl1<>pl4) then
begin
     . . .
     if po1>0 then - czy zmienna po1 ma wartość większą od zera?
     begin - jeśli tak, to...
          new(pom3); - utwórz w pamięci nowy element lisc
          with pom3^ do
          begin                  } związ nowy element w wiązaniach...
               lewa:=pl2;        } głównych tak, by znalazł się między...
               prawa:=pl1;       } elementem pl2 i pl1
               kur_d:=pl1^.kur_d;
```

```
                lx:=pl1^.lx;
                px:=lx;          } jego współrzędne poziome i pionowe...
                ly:=pl1^.ly;     } mają być takie, jak elementu dostępnego...
                py:=pl1^.py;     } przez wskaźnik pl1
                y1:=ly;
                y2:=py;
                znak:=' '; - wpisz znak spacji
                druk:=true - znak ma być widoczny na ekranie
                    (znak spacji będzie zastąpiony innym znakiem,...
                     więc zaznaczenie tego pola jest niejako na wyrost)
            end;
            pl2^.prawa:=pom3;
            pl1^.lewa:=pom3;
            if po1=1 then - czy zmienna po1 ma wpisaną liczbę jeden?
            begin - jeśli tak, to...
                 pom3^.kur_l:=pl3;          } nowy element jest...
                 pom3^.kur_p:=nil;          } strukturą zamykającą...
                 pom3^.kur_g:=pl1^.kur_g;   } elementu pl3
                 pl3^.kur_p:=pom3
            end else - jeśli nie, zmienna po1 ma wpisaną wartość dwa, czyli...
                begin
                     pom3^.kur_l:=adres4;
                     pom3^.kur_p:=pl1;      } nowy element jest strukturą...
                     pom3^.kur_g:=pl2;      } zamykającą elementu...
                     pl1^.kur_l:=pom3;      } adres4
                     if pl1^.kur_g=pl2 then pl1^.kur_g:=nil;
                     pl1:=pom3
                end;
end;
```

Niezależnie od tego, czy nowy składnik `lisc` został ulokowany w pamięci czy nie, nowy układ funkcji potęgującej musi zostać uformowany. Polega to na dopasowaniu rozmiarów znaków wpisanych w składniki `lisc`, począwszy od elementu wskazanego wskaźnikiem `pl1` a skończywszy na dotychczasowej jeszcze strukturze zamykającej, dostępnej przez wskaźnik `pl4`. Dopasowaniem wielkości znaków względem rozmiarów elementu bieżącego, dostępnego przez wskaźnik `adres4`, zajmuje się procedura `Skalowanie`. Procedura otrzymuje w parametrach trzy elementy: adres bieżącego elementu `adres4` – element odniesienia, wartość logiczną `false` nakazującą procedurze przywrócenie rozmiarów sprzed zmniejszenia oraz adres elementu od którego ma nastąpić przywracanie rozmiarów. Gdyby w elemencie dostępnym przez wskaźnik `pl1` był wpisany znak działania, konieczne jest jego dodatkowe poszerzenie przez procedurę `Szerokosc`. Teraz już można związać bieżący element `adres4` z nową strukturą zamykającą.

```
Form3.Skalowanie(adres4,false,pl1); - przywróć rozmiary sprzed...
                                      zmniejszenia

Form3.Szerokosc(pl1); - dodatkowe poszerz znak przez skorygowanie...
                        współrzędnych poziomych elementu pl1

adres4^.kur_p:=pl1; - związ kursorowo bieżący element z nową...
                      strukturą zamykającą

pl1^.kur_l:=adres4; - związ nową strukturę zamykającą z elementem bieżącym
pl1^.kur_g:=pl2; - związ kursorowo nową strukturę zamykającą ...
                   z ostatnim elementem funkcji potęgującej/podstawą logarytmu
```

```
pl4^.kur_g:=nil; - była struktura zamykająca nie może być już związana...
                   z ostatnim elementem funkcji potęgującej/podstawą logarytmu...
                   dlatego w pole kur_g wpisz adres pusty
```

Po przywróceniu rozmiarów, a więc po skorygowaniu współrzędnych poziomych i pionowych elementom, które przestały należeć do funkcji potęgującej czy podstawy logarytmu, trzeba je jeszcze odpowiednio ułożyć na ekranie tak, by były ulokowane w jednej linii z bieżącym elementem, a więc tym, wskazywanym przez wskaźnik `adres4`. Do określenia wielkości przesunięcia elementów w pionie wystarczy wyliczyć różnicę między polem `ly` bieżącego elementu i tego samego pola należącego do elementu dostępnego przez wskaźnik `pl1`. Jeśli wyliczona liczba będzie różna od zera, uruchamiana zostaje pętla `repeat..until`, w której wskazanym elementom zostają skorygowane współrzędne pionowe. Dodatkowo, jeśli w zlokalizowanym elemencie został ukryty znak i nie jest on znakiem potęgowania, zostaje on odkryty przez ustawienie pola `druk` w pozycji `true` a znakowi przywrócona zostaje właściwa szerokość. Przywrócenie szerokości wiąże się z przesunięciem pozostałych znaków tak, by nie zachodziły na siebie.

```
pom3:=pl1; - zapamiętaj w zmiennej pom3 adresu elementu pl1
pl3:=pom3; - wstępne zapamiętaj w zmiennej pl3 adres poprzednika...
             byłej struktury zamykającej – pl4

po1:=adres4^.ly-pom3^.ly; - wylicz różnicę między polami ly...
          bieżącego elementu adres4 i nowej struktury zamykającej

if po1<>0 then - czy wartość w zmiennej po1 jest różna od zera?
repeat - jeśli tak, uruchom pętlę
      with pom3^ do
      begin
           y1:=y1+po1;
           y2:=y2+po1;   } skoryguj współrzędne pionowe...
           ly:=ly+po1;   } o wyliczoną różnicę
           py:=py+po1
      end;
      if pom3^.kur_d=pl4 then pom3^.kur_d:=nil; - jeśli...
                              zostało zachowane związanie kursorowe...
                              z byłą strukturą zamykającą, usuń to związanie

      pl3:=pom3; - zapamiętaj adres ze wskaźnika pom3
      if not pl3^.druk then - czy obecny element ma ukryty znak?
         if pl3^.znak<>'^' then - jeśli tak, to czy nie jest to znak...
                                  potęgowania?
         begin - jeśli nie, to...
              pl3^.druk:=true; - odkryj znak
              Form3.Szerokosc(pl3); - przywróć mu szerokość
              Form3.Przesun(pl3,false) - przesuń pozostałe...
                                         elementy w prawo o tę szerokość
         end;
      pom3:=pom3^.prawa; - odczytaj adres następnika w wiązaniu głównym
until pom3=pl4; opuść pętlę, gdy wskaźnik pom3 osiągnie adres wskaźnika pl4
```

Teraz już można związać kursorowo elementy funkcji potęgującej lub podstawy logarytmu z nową strukturą zamykającą.

```
pom3:=adres4^.prawa; - pozyskaj adres pierwszego elementu funkcji potęgującej...
                       lub podstawy logarytmu
```

```
repeat
     if pom3^.kur_d=pl4 then pom3^.kur_d:=pl1; - związ z nową...
                                    strukturą zamykającą tylko te elementy,...
                                    które w polach kur_d mają zachowany...
                                    adres poprzedniej struktury zamykającej
     if pom3^.kur_g=pl4 then pom3^.kur_g:=pl1; (jak wyżej,...
                                    tylko dla elementów podstawy logarytmu)
     pom3:=pom3^.prawa - pozyskaj adres następnika
until pom3=pl1; - opuść pętlę, gdy wskaźnik pom3 będzie miał adres...
                  nowej struktury zamykającej
```

W tej części bloku instrukcji pozostaje jeszcze związanie kursorowe byłej struktury zamykającej ze swoim poprzednikiem lub gdy zawiera ona znak spacji, usunięcie jej.

```
if pl4^.znak=' ' then - czy była struktura zamykająca zawiera znak spacji?
begin - jeśli tak, to...
     pom3:=pl4^.prawa; - do wskaźnika pom3 wpisz adres następnika...
                         w wiązaniu głównym
     pl3^.prawa:=pom3; - związ poprzednika wiązaniem głównym…
                         z następnikiem usuwanego elementu
     if pom3<>nil then pom3^.lewa:=pl3; jeśli istnieje następnik...
      usuwanego elementu, związ go wiązaniem głównym z jego poprzednikiem
     pom3:=pl4^.kur_p; - do wskaźnika pom3 wpisz adres następnika...
                         w wiązaniu kursorowym
     pl3^.kur_p:=pom3; - związ poprzednika wiązaniem kursorowym...
                         z następnikiem usuwanego elementu
     pl3^.kur_g:=pl2;
     if pom3<>nil then pom3^.kur_l:=pl3; - jeśli istnieje następnik...
      usuwanego elementu w wiązaniu kursorowym, związ go z jego poprzednikiem
     dispose(pl4) - usuń element z pamięci wskazywany przez wskaźnik pl4
end else jeśli była struktura zamykająca nie zawiera znaku spacji, to...
    begin
         pl4^.kur_l:=pl3;  } związ ją kursorowo ze swoim poprzednikiem
         pl3^.kur_p:=pl4;  }
         if pl4^.kur_g=pl2 then pl4^.kur_g:=nil;
    end;
```

Jeśli element dostępny we wskaźniku `pl1`, zaraz po opuszczeniu pętli, nie należy do otoczenia bieżącego znaku potęgowania, dalsza analiza jest niemożliwa, dlatego w tej sytuacji otoczenie funkcji potęgującej pozostaje niezmienione. Trzeba tylko przypisać adres ostatniego elementu funkcji potęgującej lub podstawy logarytmu wskaźnikowi `pl2` a wskaźnikowi `pl1` – adres struktury zamykającej.

```
if (pl3=adres4) and (pl1<>pl4) then
begin
     . . .
end else
    begin
         pl2:=pl4^.lewa; - wskaźnikowi pl2 wpisz adres poprzednika...
                           struktury zamykającej
         pom3:=adres4^.prawa; - wskaźnikowi pom3 przypisz adres...
                                pierwszego elementu funkcji potęgującej
         ukryj_nawiasy(pom3,pl2); - gdy w pierwszy i ostatni element...
                                    wpisana została para nawiasów, ukryj te nawiasy
```

```
            pl1:=pl4 - przypisz wskaźnikowi pl1 adres struktury zamykającej
    end
```

Brak struktury zamykającej jest jednoznaczny z brakiem elementu funkcji potęgującej czy podstawy logarytmu, dlatego jeśli dla bieżącego elementu nie istnieje następnik w wiązaniu kursorowym, zmienne logiczne `brak_p` i `brak_l` ustawione zostają w pozycji `true`.

```
if pl4<>nil then - czy istnieje następnik w wiązaniu kursorowym...
                   bieżącego elementu – adres4?
begin
     . . .
end else - jeśli nie, to...
    begin
         brak_l:=true; - ustaw zmienną logiczną brak_l w pozycji true...
                         (brak struktury zamykającej)
         brak_p:=true - ustaw zmienną logiczną brak_p w pozycji true...
                        (brak elementu funkcji potęgującej/podstawy logarytmu)
    end;
```

Wykrycie wartości `true` w powyższych zmiennych logicznych, zmusza procedurę do utworzenia w pamięci tych elementów, wiążąc je zgodnie ze schematem zamieszczonym na rysunku 49 lub 50. Pola `znak` obydwu elementów otrzymują po jednym znaku spacji, redukując ich szerokość do zera.

Jeśli istnieje choć jeden element funkcji potęgującej/podstawy logarytmu, czyli gdy wskaźnik `pl2` nie ma adresu pustego, wszystkie te elementy muszą zostać związane kursorowo ze strukturą zamykającą. Związanie to polega na przypisaniu polom `kur_g` – w przypadku podstawy logarytmu lub `kur_d` – w przypadku funkcji potęgującej, adresów struktury zamykającej, ale tylko tym polom, które mają jeszcze wpisane adresy puste, czyli `nil`. Jest to ważne z uwagi na możliwość istnienia wiązań np. z kreską ułamkową, gdyby funkcja potęgująca składa się z ułamka. W tym przypadku nie wolno rozbijać już istniejącego wiązania. Instrukcje wiążące podzielone zostały na dwie części, jedna zajmuje się wiązaniem funkcji potęgującej a druga, podstawą logarytmu.

```
if brak_p then
begin
     . . .
end else - jeśli element funkcji potęgującej/podstawy logarytmu istnieje, to...
    begin
         pl3:=adres4^.prawa; - wskaźnikowi pl3 przypisz adres...
                               następnika bieżącego elementu w wiązaniu głównym
         if tryb4 then - czy parametr tryb4 jest ustawiony w pozycji true?
         begin - jeśli tak, to zajmij się funkcją potęgującą
              pl1^.kur_g:=pl2; - związ strukturę zamykającą z ostatnim...
                                 elementem funkcji potęgującej
              pl1:=pl2; - przekaż adres ostatniego elementu funkcji...
                          potęgującej wskaźnikowi pl1...
                          (zmienna sterująca pętlą)
              pl4:=adres4^.kur_l; - do wskaźnika pl4 wpisz adres...
                                    poprzednika bieżącego elementu
              pl4^.kur_g:=pl3; - związ kursorowo element pl4...
                                 z pierwszym elementem funkcji potęgującej
              pl4:=adres4^.kur_p; - do wskaźnika pl4 wczytaj adres ...
```

```
                                        struktury zamykającej
            repeat
                  if not wiazanie(pl1,adres4,false)
                                 czy można...
                                 związać element pl1 ze strukturą zamykającą?
                     then pl1^.kur_d:=pl4;  jeśli można – związ
                  pl1:=pl1^.lewa  –  pozyskaj adres poprzednika...
                                        w wiązaniu głównym
            until pl1=adres4;  –  opuść pętlę, gdy wskaźnik pl1 uzyska...
                                  adres bieżącego elementu

            adres4^.kur_g:=pl3;  –  związ bieżący element z pierwszym ...
                                    elementem funkcji potęgującej
        end else  –  skoro parametr tryb4 ustawiony jest w pozycji false, to...
            begin  –  zajmij się podstawą logarytmu
                 pl1^.kur_d:=pl2;
                 pl1:=adres4;
                 pl4:=pl1^.kur_p;
                 repeat
                       pl1:=pl1^.prawa;
                       if not wiazanie(pl1,adres4,true)
                          then pl1^.kur_g:=pl4;
                 until pl1=pl2;
                 adres4^.kur_d:=pl3;
            end;
        pl3^.kur_l:=nil;  –  zamknij pierwszy element z lewej strony
        pl2^.kur_p:=nil;  –  zamknij ostatni element z prawej strony
end;
```

Gdy otoczenie funkcji potęgującej lub logarytmu o dowolnej podstawie jest już kompletne i poprawnie ze sobą związane, można już je przeskalować oraz ułożyć względem znaku wpisanego w element struktury otwierającej. Do przeskalowania, czyli proporcjonalnego zmniejszenia rozmiarów znaków w stosunku do rozmiarów znaku znajdującego się bezpośrednio przed ukrytym znakiem potęgowania, służy procedura `Skalowanie`. Procedurze tej podane zostały w parametrze: adres bieżącego składnika `lisc` oraz wartość `true` nakazująca procedurze zmniejszanie. Brak trzeciego parametru jest informacją dla procedury, że skalowaniu będą poddane elementy, począwszy od następnika elementu wskazywanego przez parametr `adres4` a skończywszy na poprzedniku struktury zamykającej. Po opuszczeniu procedury, należy wyliczyć różnicę potrzebną do przesunięcia w pionie elementów funkcji potęgującej lub podstawy logarytmu. Wyliczenie to odbywa się oddzielnie dla potęgowania i logarytmu. W przypadku potęgowania, dodatkowo wyliczona zostaje ewentualna różnica między wartością pola `lx` pierwszego elementu funkcji potęgującej a wcześniej wyliczoną wartością, zapamiętaną w zmiennej `po2`, od której powinna rozpoczynać się funkcja potęgująca. Zarówno korygowanie pól współrzędnych poziomych `lx` i `px` o wyliczoną różnicę jak też wyszukiwanie najniższego punktu, czyli największej wartości współrzędnej rzeczywistej pionowej z pól `y2` należących do elementów funkcji potęgującej, odbywa się w pętli `repeat..until`. Po opuszczeniu pętli i uzyskaniu maksymalnej współrzędnej pionowej, wyliczona zostaje wartość, która wykorzystana zostanie do pozycjonowania funkcji potęgującej względem ostatniego elementu funkcji potęgowanej. Do wyliczenia wykorzystany został iloraz wysokości poziomu, do którego należy funkcja potęgowana i wartości zawartej w zmiennej pozycjonowania `QP`, należącej do bloku regulacji.

```
Form3.Skalowanie(adres4,true); - zmniejsz rozmiary elementów...
                                 funkcji potęgującej/podstawy logarytmu
pl3:=adres4^.prawa; - pozyskaj adres pierwszego elementu
pl4:=adres4^.kur_p; - pozyskaj adres struktury zamykającej
if tryb4 then - czy procedura ma zajmować się potęgowaniem?
begin - jeśli tak, to...
     if wsb=adres4 then - czy elementem bieżącym edytora jest element...
                          ze znakiem potęgowania?
     begin - jeśli tak, to...
          wsb:=pl3; - elementem bieżącym edytora jest następnik...
                      elementu bieżącego
          stat_kur:=false - ustaw kursor przed znakiem
     end;
     pl1:=pl3; - zapamiętaj we wskaźniku pl1 adres następnika elementu bieżącego
     po1:=pl1^.y1; - zapamiętaj wartość górnej współrzędnej pionowej
     po2:=pl1^.lx-po2; - wylicz różnicę między rzeczywistą a oczekiwaną...
                         współrzędną poziomą rozpoczęcia funkcji potęgującej
     repeat
          if pl1^.y2>po1 then po1:=pl1^.y2; - jeśli wartość dolnej...
                                              współrzędnej pionowej jest większa od tej...
                                              zapamiętanej w zmiennej po1, wpisz ją do tej zmiennej
          pl1^.lx:=pl1^.lx-po2; - skoryguj początkową współrzędną...
                                  poziomą o wartość w zmiennej po2
          pl1^.px:=pl1^.px-po2; - skoryguj końcową współrzędną...
                                  poziomą o wartość w zmiennej po2
          pl1:=pl1^.prawa - pozyskaj adres następnika
     until pl1=pl4; - opuść pętlę, gdy adres we wskaźniku pl1 będzie taki...
                      jak w pl4
     pl1:=adres4^.kur_l; - pozyskaj adres struktury otwierającej
     po2:=pl1^.py-pl1^.ly; - wylicz jej wysokość poziomu
     po2:=pl1^.y1+trunc(po2/Form6.QP); - wylicz wstępną pozycję funkcji...
                                         potęgującej na znaku odniesienia
     po1:=po1-po2 - wylicz ostateczną korekcję pozycji funkcji potęgującej
end else - jeśli nie, procedura zajmuje się logarytmem
    begin
         if wsb=adres4 then
         begin                    } (jak
              wsb:=pl4;           } wyżej)
              stat_kur:=false     }
         end;
         pl1:=pl4^.lewa; - pozyskaj adres poprzednika struktury zamykającej
         po1:=pl1^.y2; - zapamiętaj wartość dolnej współrzędnej pionowej
         repeat
              if pl1^.y1<po1 then po1:=pl1^.y1; - jeśli wartość...
                                                  górnej współrzędnej pionowej jest mniejsza od tej...
                                                  zapamiętanej w zmiennej po1, wpisz ją do tej zmiennej
              pl1:=pl1^.lewa - pozyskaj adres poprzednika
         until pl1=adres4; - opuść pętlę, gdy wskaźnik pl1 osiągnie adres...
                             elementu bieżącego
         po2:=pl1^.py-pl1^.ly; - wylicz wysokość poziomu struktury ...
                                 otwierającej logarytmu
         po2:=pl1^.y2-trunc(po2/Form6.QP); - wylicz wstępną pozycję...
                                             podstawy logarytmu na znaku odniesienia
```

```
            po1:=po1-po2 - wylicz ostateczną korekcję pozycji podstawy logarytmu
      end;
```

Podstawą do wyliczenia wielkości przesunięcia w pionie podstawy logarytmu jest dolna współrzędna struktury otwierającej, w przeciwieństwie do funkcji potęgującej, gdzie podstawą jej wyliczenia jest górna współrzędna pionowa. Jeśli wyliczona wartość przesunięcia jest różna od zera, uruchamiana jest pętla, w której pola współrzędnych pionowych zarówno w elementach funkcji potęgującej, jak i podstawy logarytmu, są korygowane o wartość wyliczoną w zmiennej `po1`.

```
if po1<>0 then - czy wyliczona korekcja położenia elementów w pionie...
                 jest różna od zera?
begin - jeśli tak, to...
     pl1:=pl3; - pozyskaj adres pierwszego elementu...
                 funkcji potęgującej/podstawy logarytmu
     repeat
           with pl1^ do
           begin
                y1:=y1-po1; - współrzędna rzeczywista górna
                y2:=y2-po1; - współrzędna rzeczywista dolna
                ly:=ly-po1; - współrzędna poziomu górna
                py:=py-po1 - współrzędna poziomu dolna
           end;
           pl1:=pl1^.prawa; - pozyskaj adres następnika
     until pl1=pl4; - opuść pętlę, gdy wskaźnik pl1 osiągnie adres struktury...
                      zamykającej
end;
```

Nim procedura zakończy pracę, pozostaje jeszcze ewentualne skorygowanie współrzędnych poziomych struktury zamykającej i elementów znajdujących się za nią. Polega to na wyliczeniu sumy współrzędnej `px` elementu wskazywanego przez parametr `adres4`, długości funkcji potęgującej lub podstawy logarytmu oraz początkowej współrzędnej poziomej struktury zamykającej, czyli wartości z pola `lx`. Suma różna od zera wskazuje na potrzebę korekcji.

```
po1:=adres4^.px+dlugosc(pl3,pl4); - wylicz pozycję początkową struktury...
                                    zamykającej równej sumie wartości w polu...
                                    px elementu struktury otwierającej i długości...
                                    funkcji potęgującej/podstawy logarytmu
po1:=po1-pl4^.lx; - wylicz w zmiennej po1 różnicę między wyliczoną pozycją...
 poziomą struktury zamykającej a obecną współrzędną, wpisaną w pole lx tej struktury
if po1<>0 then - czy zmienna po1 jest różna od zera?
begin - jeśli tak, to...
     po2:=pl4^.px-pl4^.lx; - wylicz szerokość znaku struktury zamykającej
     pl4^.lx:=pl4^.lx+po1; - skoryguj początkową współrzędną poziomą...
                             o wyliczoną różnicę
     pl4^.px:=pl4^.lx+po2; - skoryguj współrzędną końcową względem...
                             skorygowanej współrzędnej początkowej...
                             i szerokości znaku
     Form3.Przesun(pl4,true) - przesuń w poziomie pozostałe znaki...
                               znajdujące się za strukturą zamykającą...
                               względem tej struktury
end;
```

XIV. 2.4. Zarządzanie nawiasami – procedura `nawiasy`

Procedura ta zajmuje się wyłącznie nawiasami. Jej zadania to:

- dopasowanie rozmiarów nawiasów do rozmiarów funkcji zamkniętej w parze nawiasów lub w przypadku pojedynczego nawiasu, do znaków sąsiadujących z nawiasem;
- wybór typu nawiasu, który uzależniony jest od stopnia jego zagnieżdżenia w kolejnych nawiasach otwierających.

Zadania te wymagają skompletowania wszystkich nawiasów zawartych w polach `znak` elementów edytora, zapamiętując adresy tych elementów, rodzaje nawiasów oraz kolejność ich występowania. By usprawnić kompletowanie nawiasów, w procedurze zadeklarowana została struktura danych o nazwie `zbior`.

```
type zbior=^tabl;
     tabl=record
                nawias:boolean;
                adres:lisc;
                typ:integer;
                poprz:zbior;
                nast:zbior
end;
```

Znaczenie pól struktury jest następujące:

- `nawias` – pole typu logicznego, którego wartość `true` lub `false`, pozwoli odróżnić, czy rozpoznany nawias jest otwierający czy zamykający;
- `adres` – pole przechowujące adres elementu `lisc`, w którego pole `znak` wpisany został nawias;
- `typ` – pole pozwalające określić typ nawiasu. Możliwe są następujące wartości:
  - 0 – nawias okrągły,
  - 1 – nawias kwadratowy,
  - 2 lub więcej – nawias sześcienny,
- `poprz`, `nast`. – pola do wiązania z poprzednikiem i następnikiem listy o nazwie `zbior`.

Zadeklarowana stała `typy_nawiasow` zawiera wartości w kodzie ASCII, nawiasów otwierających i zamykających, przez co łatwiej jest te nawiasy identyfikować. W pierwszej pętli `while..do` odbywa się wyszukiwanie spośród wszystkich elementów edytora tych składników, które w polach `znak` mają wpisane znaki nawiasów a ich pola `druk` są ustawione w pozycji `true`, co oznacza, że nawiasy te nie są ukryte. Znalezienie takiego elementu powoduje utworzenie w pamięci elementu `zbior`, natomiast jego pola otrzymują wartości identyfikujące znaleziony nawias. Ponieważ na etapie kompletowania nawiasów, typy nawiasów są jeszcze nieokreślone, pole `typ` utworzonego elementu `zbior` zostaje zainicjowane wartością zero. Przed wczytaniem do zmiennej sterującej pętlą adresu następnego składnika `lisc`, wywoływana jest procedura `wys_nawiasu`, która dostosowuje wysokość nawiasu znalezionego w pętli do wysokości znaku znajdującego się przy tym nawiasie.

```
const typy_nawiasow=[40,41,91,93,123,125];
var z,z1,z2,zp:zbior;
    adr1:lisc;
    k,n1,n2:integer;
    popraw:boolean;

    procedure wys_para          } fragmenty nagłówków...
                                } wewnętrznych procedur
    procedure wys_nawiasu       }

begin
     adr1:=wsp; - do zmiennej adr1 wpisz adres pierwszego elementu lisc
     z:=nil;
     zp:=nil;
     while adr1<>nil do - wykonaj krok pętli, póki wskaźnik adr1...
                          nie ma adresu pustego
     begin
          if adr1^.druk then - czy pole druk analizowanego elementu lisc,...
                               ma ustawione pole druk w pozycji true?
          begin - jeśli tak, to...
               k:=ord(adr1^.znak); - odczytaj znak wpisany w pole znak...
                     i zapamiętaj go w zmiennej k jako wartość w kodzie ASCII
               if k in typy_nawiasow then - czy odczytana wartość...
                                            zawiera się w tablicy typy_nawiasow?
               begin - jeśli tak, to...
                    z1:=z; - w zmiennej z1 zapamiętaj adres dotychczasowego...
                             elementu zbior
                    new(z); - utwórz w pamięci nowy element zbior
                    z^.poprz:=z1; - w pole poprz nowego elementu...
                                    wpisz adres poprzedniego elementu
                    z^.nast:=nil; - pole nast zainicjuj adresem pustym
                    if z1<>nil then z1^.nast:=z; - jeśli poprzedni...
                            element zbior istnieje, wpisz w jego pole nast...
                            adres nowego elementu
                    if k in otw then z^.nawias:=true - jeśli wartość...
                                odczytanego nawiasu zawiera się w tablicy otw...
                                zawierającej wartości nawiasów otwierających...
                                ustaw pole nawias w pozycji true
                    else z^.nawias:=false; - w przeciwnym przypadku...
                                             ustaw pole nawias w pozycji false
                    z^.adres:=adr1; - w pole adres wpisz adres elementu...
                                      lisc, z wpisanym nawiasem
                    z^.typ:=0; - w pole typ wpisz liczbę zero
                    wys_nawiasu(z); - wywołaj procedurę wys_nawiasu
                    if z1=nil then zp:=z - jeśli wskaźnik z1 posiada...
                            adres pusty, oznacza to, że nowy element zbior...
                            jest pierwszym elementem listy, dlatego zachowaj...
                            jego adres w zmiennej zp
               end;
          end;
          adr1:=adr1^.prawa - w zmiennej adr1 pozyskaj adres następnego...
                              elementu lisc w wiązaniu głównym
     end;
```

Po skompletowaniu wszystkich nawiasów w elementach `zbior` uruchamiana jest kolejna pętla `while..do`, w której ustalany jest typ każdego nawiasu. Działanie pętli polega na zliczaniu w zmiennej `n1` utworzonych składników `zbior` w ten sposób, że gdy pole `nawias` analizowanego składnika zostało ustawione w pozycji `true`, a więc zapamiętany nawias jest otwierający, wartość w tej zmiennej jest zwiększana o jeden, w przeciwnym przypadku, a więc dla nawiasu zamykającego, wartość ta jest zmniejszana o jeden. Po rozpoznaniu nawiasu otwierającego, wartość ze zmiennej `n1` zostaje przepisana do pola `typ` analizowanego składnika `zbior` tuż po inkrementacji wartości w tej zmiennej, natomiast po rozpoznaniu nawiasu zamykającego, przed inkrementacją. Zaprezentowany sposób uzupełniania pól `typ` ma na celu zachowanie jednakowych numerów w tych polach dla nawiasów stanowiących parę.

Tab. 2. Wpływ numeracji typu nawiasu na kształt nawiasu oraz wygląd przykładowej funkcji dla niepoprawnej i poprawnej numeracji typu nawiasu

Wstępna numeracja typu nawiasu w pętli zewnętrznej						
Numery typu nawiasu wpisane w pola `typ` kolejnych elementów `zbior`	1	2	3	3	2	1
Kształty nawiasów odpowiadające numerom ich typów	[	{	{	}	}	]
Przykładowa funkcja	$[x + \{5 \cdot \{x - 1\}\}]$					
Ostateczna numeracja typu nawiasu w pętli wewnętrznej						
Numery typu nawiasu wpisane w pola `typ` kolejnych elementów `zbior`	2	1	0	0	1	2
Kształty nawiasów odpowiadające numerom ich typów	{	[	(	)	]	}
Przykładowa funkcja	$\{x + [5 \cdot (x - 1)]\}$					

Z każdym krokiem pętli modyfikowana jest zmienna `n2`, która przyjmuje największą wartość, jaką osiągnie zmienna `n1`.

Gdy zmienna `n1` będzie miała wartość zero, zostaje uruchomiona wewnętrzna pętla `repeat..until`, w której każdy składnik `zbior` otrzyma właściwą numerację typu nawiasu. Numer ten jest wynikiem różnicy maksymalnej wartości zapamiętanej w zmiennej `n2` i numeru typu wpisanego w pole `typ` poprawianego składnika `zbior` – różnica ta od razu zastępuje dotychczasową wartość w tym polu. Pętla ta wykonuje się wstecz, a więc w odwrotnym kierunku niż ten, w jakim wykonuje się pętla zewnętrzna. Jej opuszczenie nastąpi po uzyskaniu przez zmienną sterującą `z` adresu takiego składnika `zbior`, przy którym podczas poprzedniego wywołania pętli wewnętrznej zmienna `n1` uzyskała wartość zero. Oczywiście podczas pierwszego wywołania pętli, jej opuszczenie nastąpi po poprawieniu pola `typ` pierwszemu składnikowi `zbior`. Warunek ten sprawia, że podczas każdego kolejnego wywołania pętli, poprawianie typów rozpoczyna się od takiego nawiasu zamykającego, który z wcześniej wykrytym nawiasem otwierającym stanowią parę niezagnieżdżoną w innej parze

nawiasów. Ponadto te nawiasy, które mają już poprawną numerację typów, nie będą poprawiane po raz kolejny.

Po upuszczeniu pętli wewnętrznej, zmiennej `n2` przypisana zostaje wartość zero, co umożliwia jej dalszą pracę w pętli, zaś wskaźnikowi `z2` przypisany zostaje adres składnika `zbior`, który będzie warunkiem opuszczenia pętli wewnętrznej podczas kolejnego jej wywołania.

```
z1:=zp; - wpisz do zmiennej z1 adres pierwszego elementu listy
z2:=nil;
z:=nil;
n1:=0;
n2:=0;
while z1<>nil do - wykonaj krok pętli, póki zmienna z1 nie ma adresu pustego
begin
     if z1^.nawias then - czy pole nawias elementu analizowanego w pętli...
                          jest ustawione w pozycji true?
     begin - jeśli tak, to nawias jest otwierający, czyli...
          inc(n1); - zwiększ wartość w zmiennej n1 o jeden
          z1^.typ:=n1; - wpisz wartość ze zmiennej n1 w pole typ
     end else - jeśli nie, to nawias jest zamykający, czyli...
         begin
              z1^.typ:=n1; - wpisz w pole typ dotychczasową wartość...
                             ze zmiennej n1
              dec(n1); - zmniejsz wartość w zmiennej n1 o jeden
         end;
     if z1^.typ>n2 then n2:=z1^.typ; - jeśli wartość w polu typ jest...
     większa od wartości w zmiennej n2, wpisz do tej zmiennej wartość z tego pola
     if n1=0 then - czy wartość w zmiennej n1 jest równa zero?
     begin - jeśli tak, to...
          z:=z1; - wpisz do zmiennej z adres elementu analizowanego w pętli
          repeat
                z^.typ:=n2-z^.typ; - wpisz do pola typ elementu...
                     wskazywanemu przez wskaźnik z różnicę wartości...
                     w zmiennej n2 i dotychczasowej wartości w polu typ
                z:=z^.poprz - we wskaźniku z pozyskaj adres poprzednika
          until z=z2; - opuść pętlę, gdy wskaźnik z uzyska adres...
                        zawarty we wskaźniku z2
          n2:=0; - wyzeruj zmienną n2
          z2:=z1 - zachowaj adres elementu analizowanego w pętli w zmiennej z2
     end;
     z:=z1; - zapamiętaj we wskaźniku z adres bieżącego elementu, by po...
              opuszczeniu pętli, wskaźnik ten wskazywał na ostatni element listy
     z1:=z1^.nast - w zmiennej z1 pozyskaj adres następnika
end;
```

Jeśli po opuszczeniu pętli `while..do` zmienna `n1` nie posiada wartości zero, oznacza to, że zostały jeszcze elementy, którym pola `typ` nie zostały poprawione. Uruchamiana jest zatem dodatkowa pętla `while..do`, w której elementowi wskazywanemu przez wskaźnik `z` oraz kolejnym poprzednikom, poprawiana jest numeracja typu nawiasu.

```
if n1<>0 then - czy zmienna n1 jest różna od zera?
   while z<>nil do - jeśli tak, to wykonaj krok pętli, póki zmienna z...
                     nie ma adresu pustego
   begin
        if z=z2 then break; - jeśli zmienna z uzyskała adres elementu...
                              zapamiętanego przez wskaźnik z2, opuść pętlę
        z^.typ:=n2-z^.typ; - wpisz do pola typ elementu...
              wskazywanemu przez wskaźnik z różnicę wartości...
              w zmiennej n2 i dotychczasowej wartości w polu typ
        z:=z^.poprz - w zmiennej z pozyskaj adres poprzednika
   end;
```

Skoro numeracja typu nawiasu jest już poprawna we wszystkich składnikach `zbior`, na jej podstawie oraz informacji z pola `nawias` o tym, czy nawias jest otwierający lub zamykający, aktualizowane są pola `znak` tych elementów `lisc`, których adresy zostały zapamiętane w polach `adres`. W tym celu zostaje uruchomiona pętla `while..do`, w której rozpoznawane są trzy grupy typów nawiasów. Rozpoznanie jednej z nich powoduje wpisanie w pole `znak` wskazanego elementu `lisc`, odpowiedniego znaku nawiasu, zgodnego z numerem typu nawiasu oraz ustawienia pola `nawias`. Ponadto, gdy pole `nawias` elementu analizowanego w pętli, jest ustawione w pozycji `false`, uruchamiana jest kolejna, wewnętrzna pętla `while..do`, w której szukany jest drugi element `zbior`, który ma taki sam numer typu nawiasu jak element bieżący w pętli, ale z ustawionym polem `nawias` w pozycji `true`. Znalezienie dwóch takich elementów oznacza znalezienie pary nawiasów, co pozwala na wywołanie procedury `wys_para`, która dostosuje wysokość tych nawiasów do funkcji zawartej między nimi.

```
z1:=zp; - do zmiennej z1 wpisz adres pierwszego elementu zbior
while z1<>nil do - wykonaj krok pętli, póki zmienna z1 nie ma adresu pustego
begin
     if z1^.typ=0 then - czy pole typ elementu wskazywanego przez wskaźnik...
                         z1 ma wartość zero?
        if z1^.nawias then z1^.adres^.znak:='(' - jeśli tak, to gdy...
                  pole nawias jest ustawione w pozycji true, wstaw w pole znak...
                  elementu lisc, którego adres znajduje się w polu adres znak…
                  nawiasu okrągłego, otwierającego
        else - jeśli pole nawias jest ustawione w pozycji false, to...
        begin
             z1^.adres^.znak:=')'; - wstaw w pole znak elementu lisc,...
                                     znak nawiasu okrągłego, zamykającego
             z2:=z1^.poprz; - do zmiennej z2 wpisz adres poprzednika...
                              elementu wskazywanego przez wskaźnik z1
             while z2<>nil do - wykonaj krok pętli, póki zmienna z2...
                                nie ma adresu pustego
             begin
                  if z2^.nawias then - czy pole nawias jest ustawione...
                                       w pozycji true?
                     if z2^.typ=z1^.typ then - jeśli tak, to czy pola typ..
                                  elementów wskazywanych przez wskaźniki...
                                  z1 i z2 mają takie same wartości?
                     begin - jeśli tak, to...
                          wys_para(z2,z1); - wywołaj procedurę...
                                             wys_para
```

```
                              break - opuść pętlę wewnętrzną
                         end;
                    z2:=z2^.poprz - w zmiennej z2 pozyskaj adres poprzednika
               end;
          end;
     if z1^.typ=1 then - czy pole typ elementu wskazywanego przez wskaźnik...
                         z1 ma wartość jeden?
          if z1^.nawias then z1^.adres^.znak:='[' - jeśli tak, to gdy...
                             pole nawias tego elementu jest ustawione w pozycji..
                             true, wstaw w pole znak elementu lisc, znak...
                             nawiasu kwadratowego, otwierającego
          else - jeśli pole nawias jest ustawione w pozycji false, to...
          begin
               z1^.adres^.znak:=']'; - wstaw w pole znak elementu lisc,...
                                       znak nawiasu kwadratowego, zamykającego
               . . . (jak wyżej)
          end;
     if z1^.typ>=2 then - czy pole typ elementu wskazywanego przez wskaźnik...
                          z1 ma wartość dwa lub większą?
          if z1^.nawias then z1^.adres^.znak:='{' - jeśli tak, to gdy...
                             pole nawias tego elementu jest ustawione w pozycji..
                             true, wstaw w pole znak elementu lisc, znak...
                             nawiasu sześciennego, otwierającego
          else - jeśli pole nawias jest ustawione w pozycji false, to...
          begin
               z1^.adres^.znak:='}'; - wstaw w pole znak elementu lisc,...
                                       znak nawiasu sześciennego, zamykającego
               . . . (jak wyżej)
          end;
     z1:=z1^.nast - w zmiennej z1 pozyskaj adres następnika
end;
```

Po poprawieniu wszystkich nawiasów, pozostało jeszcze uwolnienie pamięci zajmowanej przez wszystkie utworzone składniki `zbior`.

```
z1:=zp; - do wskaźnika z1 wczytaj adres pierwszego składnika zbior
while z1<>nil do - wykonaj krok pętli, póki wskaźnik z1 nie ma adresu pustego
begin
     z1:=z1^.nast; - we wskaźniku z1 pozyskaj adres następnika
     dispose(zp); - uwolnij pamięć zajmowaną przez element...
                    wskazywany przez wskaźnik zp
     zp:=z1 - przypisz wskaźnikowi zp adres zawarty we wskaźniku z1
end;
```

XIV. 2.4.1. Wysokość pary nawiasów – procedura `wys_para`

Zadaniem procedury jest dopasowanie rozmiarów pary nawiasów do rozmiarów wszystkich znaków zawartych między tymi nawiasami. Możliwość tą zapewnia istnienie adresów elementów edytora zawartych w polach `adres` dwóch składników `zbior`, które przekazane zostały procedurze w postaci parametrów. W polach `znak` wskazanych elementów edytora znajdują się odpowiednio: nawias otwierający i zamykający, które są dla siebie parą. Mając dostęp do struktury `lisc`, można zbadać pozycję znaków zawartych między tymi nawiasami i na ich podstawie dopasować wysokość nawiasów. Zadanie to realizowane

jest w pętli `while..do`, w której szukane są: najmniejsza i największa współrzędna pionowa znaków zawartych między nawiasami. Pętla wykonuje się od następnika w wiązaniu głównym elementu z wpisanym znakiem nawiasu otwierającego a kończy na poprzedniku elementu z wpisanym znakiem nawiasu zamykającego. By pętla mogła zostać uruchomiona, muszą być spełnione dwa warunki:

- następnym znakiem stojącym za nawiasem otwierającym nie może być potęgowanie,
- obydwa nawiasy muszą znajdować się na tym samym poziomie.

W pierwszym warunku ochroną objęty został ostatni znak funkcji potęgowanej, na podstawie którego określana jest pozycja funkcji potęgującej. Rezygnacja z tego warunku mogłaby spowodować niekontrolowany wzrost wysokości nawiasu otwierającego i tym samym przesunięcie funkcji potęgującej poza formę.

W drugim warunku nawiasy muszą znajdować się na tym samym poziomie, to znaczy, że wartości współrzędnych pionowych zawarte w polach `ly` i `py` elementów obu nawiasów mają być takie same. Pomocną okazała się tu wewnętrzna funkcja `czy_poziom`, która zwraca wartość logiczną `true`, gdy nawiasy są na tym samym poziomie lub `false`, w przeciwnym przypadku.

$$\frac{(x}{x)}$$

Rys. 52. Przykład niekompletnej funkcji, która co prawda składa się z nawiasu otwierającego i zamykającego, ale z uwagi, iż nie są one umieszczone na tych samych poziomach, nie stanowią pary

Znalezione w pętli skrajne wartości współrzędnych pionowych są jednocześnie współrzędnymi pary nawiasów. Jeśli dotychczasowe współrzędne nawiasów różnią się od tych znalezionych w pętli, współrzędne nawiasów są poprawiane. Wraz ze zmianą wysokości nawiasów zmienia się także ich szerokość, stąd konieczność przesunięcia znaków w prawo. Polega to na modyfikacji pól współrzędnych poziomych, począwszy od następnika w wiązaniu kursorowym składnika listy z wpisanym nawiasem otwierającym a skończywszy na takim składniku, który nie ma swego następnika w wiązaniu kursorowym.

```
procedure wys_para(const wyp1,wyp2:zbior);
var p1,p2:integer;
    aw1,aw2,aw3:lisc;
    czy1:boolean;

    function czy_poziom(const cz1,cz2:lisc):boolean;
    var acz1:lisc;
    begin
         Result:=false; - zwracana, domyślna wartość false przez funkcję
         acz1:=cz1; - do zmiennej acz1 wpisz adres elementu lisc...
                      z wpisanym nawiasem otwierającym
         while acz1<>nil do - wykonaj krok pętli, póki wskaźnik acz1...
                              nie ma adresu pustego
         begin
              if acz1=cz2 then - czy wskaźnik acz1 uzyskał adres...
                                 elementu lisc z wpisanym...
                                 nawiasem zamykającym?

              begin - jeśli tak, to...
                   Result:=true; - zwróć wartość true
                   exit - opuść funkcję
```

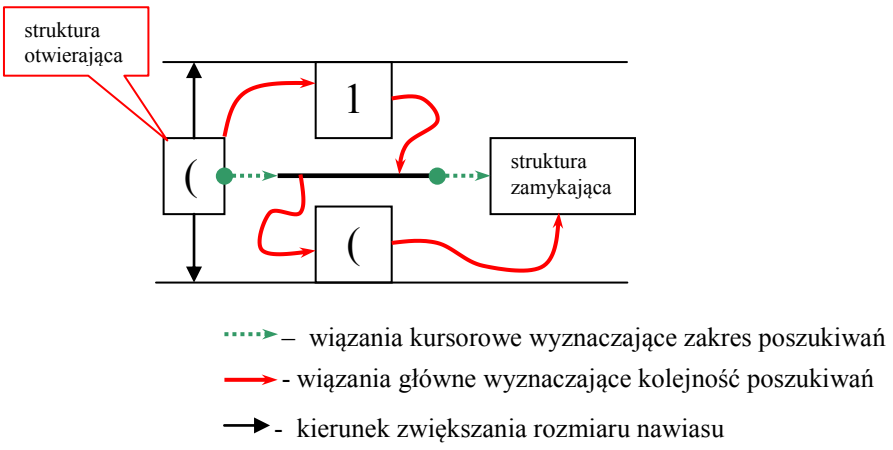
```
                end;
                acz1:=acz1^.kur_p - w zmiennej acz1 pozyskaj adres...
                                    następnika w wiązaniu kursorowym
            end;
        end;

begin
     aw1:=wyp1^.adres; - z pola adres elementu zbior wpisz do zmiennej...
                aw1 adres elementu lisc z wpisanym nawiasem otwierającym
     aw2:=wyp2^.adres; - z pola adres elementu zbior wpisz do zmiennej...
                aw2 adres elementu lisc z wpisanym nawiasem zamykającym
     czy1:=false; - zainicjuj zmienną logiczną czy1 wartością false
     aw3:=aw1^.kur_p; - do zmiennej ad3 wpisz adres następnika w wiązaniu...
                kursorowym elementu wskazywanego przez wskaźnik aw1
     if aw3<>nil then - czy wskaźnik aw3 nie ma adresu pustego?
        if aw3^.znak='^' then czy1:=true; - jeśli nie, to jeśli w jego...
                                pole znak jest wpisany znak potęgowania,...
                                ustaw zmienną czy1 w pozycji true?
     if not czy1 then - czy zmienna czy1 jest ustawiona w pozycji false?
        if czy_poziom(aw1,aw2) then jeśli tak, to czy oba nawiasy...
                zawarte w składnikach aw1 i aw2, są na tym samym poziomie?
        begin - jeśli tak, to...
             p1:=maxint; - wpisz do zmiennej p1 wartość startową...
                        do szukania minimalnej współrzędnej pionowej
             p2:=-maxint; - wpisz do zmiennej p2 wartość startową...
                        do szukania maksymalnej współrzędnej pionowej

            aw1:=aw1^.prawa; - we wskaźniku aw1 pozyskaj adres...
                                następnego elementu w wiązaniu głównym
            while aw1<>aw2 do - wykonaj krok pętli, póki wskaźnik aw1...
                nie uzyska adresu elementu zawierającego w swym polu znak...
                nawias zamykający
            begin
                 if aw1=nil then break; - jeśli wskaźnik aw1...
                                posiada adres pusty, opuść pętlę
                 if aw1^.y1<p1 then p1:=aw1^.y1; - jeżeli...
                            wartość współrzędnej zawartej w polu y1...
                            elementu analizowanego w pętli, jest mniejsza...
                            od wartości w zmiennej p1, wpisz do tej...
                            zmiennej wartość z pola y1
                 if aw1^.y2>p2 then p2:=aw1^.y2; - jeżeli...
                            wartość współrzędnej zawartej w polu y2...
                            elementu analizowanego w pętli, jest większa...
                            od wartości w zmiennej p2, wpisz do tej...
                            zmiennej wartość z pola y2
                 czy1:=true; - ustaw zmienną czy1 w pozycji true
                 aw1:=aw1^.prawa - pozyskaj w zmiennej aw1 adres...
                                następnika w wiązaniu głównym
            end;
            if czy1 then - czy zmienna logiczna czy1 ustawiona została...
                            w pozycji true?
            begin - jeżeli tak, to...
                 czy1:=false; - ustaw zmienną czy1 w pozycji false
```

```
                    aw1:=wyp1^.adres; - z pola adres elementu...
                   zbior wpisz do zmiennej aw1 adres elementu lisc...
                   z wpisanym nawiasem otwierającym
                    if aw1^.y1<>p1 then - czy wartości w zmiennej p1...
                                i w polu y1 elementu lisc, wskazywanego...
                                przez wskaźnik aw1, różnią się?
                    begin - jeśli tak, to...
                         czy1:=true; - ustaw zmienną czy1...
                                       w pozycji true
                         aw1^.y1:=p1 - wpisz w pole y1 wartość...
                                       ze zmiennej p1
                    end;
                    if aw1^.y2<>p2 then - czy wartości w zmiennej p2...
                                i w polu y2 elementu lisc, wskazywanego...
                                przez wskaźnik aw1, różnią się?
                    begin - jeśli tak, to...
                         czy1:=true; - ustaw zmienną czy1...
                                       w pozycji true
                         aw1^.y2:=p2 - wpisz w pole y2 wartość...
                                       ze zmiennej p2
                    end;
                    if czy1 then - czy zmienna czy1 została ustawiona...
                                   w pozycji true?
                    begin - jeśli tak, to...
                         czy1:=false; - ustaw zmienną czy1...
                                        w pozycji false

Form2.Szerokosc(aw1); - wylicz szerokość...
                                znaku nawiasu otwierającego i zmodyfikuj pole px...
                                elementu wskazywanego przez wskaźnik aw1
                         Form2.Przesun(aw1,true) -przesuń...
                                w poziomie pozostałe znaki w prawo...
                                o wyliczoną szerokość nawiasu
                    end;
                    if aw2^.y1<>p1 then
                    begin
                         . . .
                    end;
                    if aw2^.y2<>p2 then
                    begin
                         . . .
                    end;
                    if czy1 then
                    begin
                         . . .
                    end;
               end;
          end;
```

ustaw pola współrzędnych elementu z wpisanym nawiasem zamykającym, w taki sam sposób, w jaki ustawione zostały pola elementu z nawiasem otwierającym

XIV. 2.4.2. Wysokość pojedynczego nawiasu – procedura `wys_nawiasu`

Jej zadanie polega na dostosowaniu rozmiaru pojedynczego nawiasu do rozmiarów sąsiadujących ze sobą znaków z prawej strony nawiasu otwierającego lub z lewej strony nawiasu zamykającego. By było to możliwe, składnik `lisc` z wpisanym znakiem nawiasu otwierającego musi posiadać swego następnika w wiązaniu kursorowym, natomiast składnik `lisc` z wpisanym znakiem nawiasu zamykającego musi posiadać swego poprzednika, również w wiązaniu kursorowym. Jeśli odpowiedni element sąsiadujący istnieje i nie ma w swym polu `znak` znaku dzielenia, do pól współrzędnych pionowych elementu zawierającego nawias wpisywane są wartości współrzędnych od elementu sąsiadującego. W ten sposób nawias przyjmuje taką samą wysokość jak sąsiadujący z nim znak. Nieco inna jest sytuacja, gdy odpowiedni element sąsiadujący zawiera w swym polu `znak` znak dzielenia. Kreska ułamkowa nie posiada wysokości, dlatego by określić wysokość nawiasu, trzeba zbadać wysokość otoczenia ułamka. W tym celu, do wskaźnika `adw3` wczytany zostaje adres kolejnego następnika dla nawiasu otwierającego lub poprzednika dla zamykającego, otrzymując w ten sposób adres odpowiednio: struktury zamykającej bądź otwierającej ułamek. Adres zawarty w tym wskaźniku wytyczy zakres wyszukania w pętli `repeat..until` minimalnej i maksymalnej współrzędnej pionowej wśród elementów otoczenia ułamka. Na podstawie uzyskanych wartości w zmiennych `l1` i `l2`, współrzędne pionowe rzeczywiste elementu z wpisanym nawiasem zostają skorygowane tak, że nawias ten obejmie cały ułamek.



Rys. 53. Sposób wyszukiwania minimalnej i maksymalnej współrzędnej pionowej, w celu dostosowania wysokości nawiasu otwierającego do ułamka

Jeśli wysokość nawiasu uległa zmianie, zmianie musi ulec także jego szerokość. Zmiana szerokości nawiasu zmusza do przesunięcia wszystkich następnych znaków o różnicę między polem `px` elementu z wpisanym znakiem nawiasu a polem `lx` następnego elementu w wiązaniu kursorowym.

```
procedure wys_nawiasu(const wy1:zbior);
var adw1,adw2,adw3:lisc;
    l1,l2:integer;
begin
     adw2:=wy1^.adres; – do zmiennej adw2 wpisz adres elementu lisc,...
                         pochodzący z pola adres elementu zbior
     if wy1^.nawias then – czy pole nawias elementu zbior jest ustawione...
```

```
                                  w pozycji true (nawias otwierający)?
begin - jeśli tak, to...
     adw1:=adw2^.kur_p; - do zmiennej adw1 wpisz adres następnika...
                          w wiązaniu kursorowym elementu lisc...
                          z wpisanym nawiasem otwierającym

     if adw1<>nil then - czy następnik istnieje?
     begin - jeśli tak, to...
          l1:=adw1^.y1;  } do zmiennych l1 i l2 wpisz wartości ...
          l2:=adw1^.y2;  } współrzędnych rzeczywistych następnika

          if adw1^.znak='/' then - czy następnik ma wpisany...
                                   znaku dzielenia?

          begin - jeśli tak, to...
               adw3:=adw1^.kur_p; - do zmiennej adw3 wpisz adres...
                                    następnika w wiązaniu kursorowym, a więc...
                                    struktury zamykającej ułamek

               if adw3<>nil then - czy następnik istnieje?
               begin - jeśli tak, to...
                    adw1:=adw2; - wpisz do zmiennej adw1 adres...
                                  elementu lisc z wpisanym w polu znak...
                                  nawiasem otwierającym


                    repeat
                         adw1:=adw1^.prawa; - w zmiennej...
                                              adw1 pozyskaj adres następnika...
                                              w wiązaniu głównym

                         if adw1=nil then break; - jeśli...
                                                   wskaźnik adw1 uzyskał adres pusty,...
                                                   opuść pętlę

                         if adw1^.y1<l1 then l1:=adw1^.y1;
                        jeżeli wartość w polu y1 bieżącego elementu...
                        analizowanego w pętli jest mniejsza od wartości...
                        w zmiennej l1, wpisz do tej zmiennej wartość...
                        z tego pola

                         if adw1^.y2>l2 then l2:=adw1^.y2
                        jeżeli wartość w polu y2 bieżącego elementu...
                        analizowanego w pętli jest większa od wartości...
                        w zmiennej l2, wpisz do tej zmiennej wartość...
                        z tego pola

                    until adw1=adw3; - opuść pętlę, gdy wskaźnik...
                       adw1 uzyska adres zawarty we wskaźniku adw3
               end;
          end;
          adw2^.y1:=l1;  } w pola y1 i y2 elementu z wpisanym nawiasem...
          adw2^.y2:=l2;  } wpisz wartości ze zmiennych l1 i l2

          Form2.Szerokosc(adw2); - skoryguj szerokość nawiasu
          adw3:=adw2^.kur_p; - do zmiennej adw3 wpisz adres następnika
                               elementu z wpisanym nawiasem w wiązaniu kursorowym

          l1:=adw2^.px- adw3^.lx; - w zmiennej l1 wylicz różnicę...
                                    między wartościami w polach współrzędnych...
                                    poziomych px elementu z wpisanym nawiasem...
                                    a lx następnego elementu

          if l1<>0 then Form2.Przesun(adw2,true); - jeśli w...
                                                    zmiennej l1 jest wartość różna od zera,...
```

```
                                                przesuń w poziomie wszystkie znaki,...
                                                względem poprawionego nawiasu
            end;
        end else - jeśli nie, pole nawias elementu zbior jest ustawione w pozycji...
                   false (nawias zamykający)
            begin
                                (dostosuj wysokość nawiasu zamykającego do znaków...
                      . . .      go poprzedzających, w porównywalny sposób, w jaki...
                                 dostosowana została wysokość nawiasu otwierającego)
            end;
end;
```

XIV. 2.5. Ukrywanie i odkrywanie nawiasów – procedura `ukryj_nawiasy`

Jak podpowiada nazwa procedury, jej zadaniem jest ukrywanie pary nawiasów, ale także ich odkrywanie. O roli jaką procedura ma spełnić decyduje ustawienie pola `Checked` elementu menu formy o nazwie `Ukryj1`, którego wybór, dokonany przez użytkownika, ustawia to pole w pozycji `true` bądź `false`, powodując odkrycie pary nawiasów lub ponowne ich ukrycie. Procedura otrzymuje w parametrach adresy dwóch elementów edytora, poprzedzając je słowem kluczowym języka – `const`, które zabezpiecza te parametry przed przypisaniem im innych wartości niż te, które zostały procedurze przekazane. Na wstępie parametry te są sprawdzane pod kątem tego, czy nie mają adresów pustych, a jeśli nie, to czy otrzymane w ten sposób elementy edytora mają tak wpisane znaki nawiasów, że pierwszy jest otwierający a drugi zamykający. Spełnienie warunków pozwala na uruchomienie pętli, w której zliczane są nawiasy zawarte w elementach edytora, począwszy od elementu otrzymanego w pierwszym parametrze a skończywszy na poprzedniku elementu zawartego w drugim parametrze. Zliczenie nawiasu otwierającego w pierwszym kroku pętli może wydać się bezzasadne, skoro nawias ten został rozpoznany w instrukcji warunkowej, lecz stanowi to pewne zabezpieczenie przed zapętleniem się pętli, gdyby między otrzymanymi w parametrach elementami edytora nie było innego elementu, to znaczy, że obydwa te elementy byłyby bezpośrednio z sobą związane. Sytuacja taka może się zdarzyć, gdy użytkownik bezpośrednio za nawiasem otwierającym wstawi nawias zamykający, kończący np. mianownik ułamka. Dzięki temu zabezpieczeniu, pętla nie zapętli się programowo, natomiast obydwa nawiasy zostaną ukryte.

Po opuszczeniu pętli, licznik nawiasów powinien posiadać wartość jeden, co pozwoli na odkrycie lub ukrycie nawiasów otrzymanych za pośrednictwem parametrów.

Jeśli nawiasy mają być ukryte, pola `druk` w obydwu elementach edytora, zostają ustawione w pozycji `false`, zaś szerokość ukrytych nawiasów zostaje zredukowana do zera. Wszystkie następne elementy, znajdujące się za ukrytym nawiasem otwierającym, zostają przesunięte w jego miejsce. By kursor nie pozostał przy ukrytym znaku nawiasu, zmiennej `wsb` odpowiedzialnej za pozycję kursora zostaje przypisany adres poprzednika lub następnika w wiązaniu głównym elementu zawierającego w polu `znak` ukrywany nawias, zależnie od tego, czy kursor znajduje się przy nawiasie otwierającym czy zamykającym.

Jeśli nawiasy mają być odkryte, pola `druk` w elementach ukrytych nawiasów, są ustawiane w pozycji `true`, zaś ich szerokości zostają przywrócone. By pozostałe znaki nie przysłaniały odkrytych nawiasów, znaki te zostają odpowiednio przesunięte.

```
procedure ukryj_nawiasy(const pocz,kon:lisc);
var b1,b2,b3:lisc;
    n:integer;
begin
     b1:=pocz; - wpisz do zmiennej b1 adres zawarty w pierwszym parametrze
     b2:=kon; - wpisz do zmiennej b2 adres zawarty w drugim parametrze
     if (b1=nil) or (b2=nil) then exit; - jeśli któryś w adresów...
                                          jest pusty, opuść procedurę
     if ord(b1^.znak) in otw then - czy w elemencie wskazywanym przez...
                                    wskaźnik b1 wpisany jest nawias otwierający?
        if ord(b2^.znak) in zam then - jeśli tak, to czy w elemencie...
           wskazywanym przez wskaźnik b2 wpisany jest nawias zamykający?
        begin - jeśli tak, to...
             b3:=b1; - zmiennej b3 wpisz adres elementu z wpisanym...
                       nawiasem otwierającym
             n:=0; - wyzeruj licznik nawiasów
             repeat
                   if ord(b3^.znak) in otw then inc(n);
                     jeśli w elemencie wskazywanym przez wskaźnik b3...
                     jest wpisany nawias otwierający,...
                     zwiększ wartość w zmiennej n o jeden,...
                   if ord(b3^.znak) in zam then dec(n);
                     ...jeśli wpisany jest nawias zamykający,...
                       zmniejsz wartość w zmiennej n o jeden
                   if n<1 then break; - gdy licznik n ma wartość...
                                        mniejszą od jedności, opuść pętlę
                   b3:=b3^.prawa - w zmiennej b3 pozyskaj adres następnika...
                                   w wiązaniu głównym
             until (b3=b2) or (b3=nil); - opuść pętlę, gdy zmienna b3...
                     uzyska adres elementu z wpisanym nawiasem zamykającym,...
                     zawartym w zmiennej b2 lub gdy zmienna b3 uzyska adres pusty
             if n=1 then - czy licznik nawiasów posiada wartość jeden?
                if not Ukryj1.Checked then - jeśli tak, to czy pole...
                  Checked elementu menu jest ustawione w pozycji false?
                begin - jeśli tak, to ukryj nawiasy
                     b1^.druk:=false; - ustaw pole druk w pozycji...
                       false w elemencie zawierającym nawias otwierający
                     n:=b1^.lx-b1^.px; - zapamiętaj w zmiennej n różnicę...
                         między polem lx a px ukrywanego nawiasu (szerokość)
                     b1^.px:=b1^.lx; - zredukuj szerokość ukrywanego...
                                       nawiasu do zera, wpisując w pole px...
                                       taką samą wartość, jaką ma pole lx
                     if n<>0 then Form2.Przesun(b1,true); - jeśli...
                                  nawias otwierający posiadał szerokość, dosuń...
                                  pozostałe znaki w miejsce ukrywanego nawiasu
                     b2^.druk:=false; - ustaw pole druk w pozycji...
                       false w elemencie zawierającym nawias zamykający
                     b2^.px:=b2^.lx; - zredukuj szerokość ukrywanego...
                                       nawiasu do zera, wpisując w pole px...
                                       taką samą wartość, jaką ma pole lx
                     if b1=wsb then - czy wskaźnik wsb posiada adres...
                           elementu z ukrywanym nawiasem otwierającym?
```

```
                        begin - jeśli tak, to...
                              wsb:=b1^.lewa; - wpisz do wskaźnika wsb...
                                    adres poprzednika w wiązaniu głównym
                              stat_kur:=true - ustaw zmienną stat_kur...
                                  w pozycji true, ustawiając kursor za znakiem
                        end;
                        if b2=wsb then - czy wskaźnik wsb posiada adres...
                              elementu z ukrywanym nawiasem zamykającym?
                        begin - jeśli tak, to...
                              wsb:=b2^.prawa; - wpisz do wskaźnika wsb...
                                    adres następnika w wiązaniu głównym
                              stat_kur:=false - ustaw zmienną stat_kur...
                                w pozycji false, ustawiając kursor przed znakiem
                        end
                  end else - jeśli nie, to odkryj nawiasy
                  begin
                        b1^.druk:=true; - ustaw pole druk w pozycji...
                         true w elemencie zawierającym nawias otwierający
                        Form2.Szerokosc(b1); - przywróć szerokość...
                                    ukrytemu nawiasowi otwierającemu
                        Form2.Przesun(b1,true); - przesuń pozostałe znaki...
                                    w prawo, względem odkrytego nawiasu
                        b2^.druk:=true; - ustaw pole druk w pozycji...
                         true w elemencie zawierającym nawias zamykający
                        Form2.Szerokosc(b2); - przywróć szerokość...
                                    ukrytemu nawiasowi zamykającemu
                        Form2.Przesun(b2,true) - przesuń pozostałe znaki...
                                    w prawo, względem odkrytego nawiasu
                  end;
            end;
      end;
```

XIV. 2.6. Czy znak licznika/mianownika można związać z kreską ułamkową? Funkcja `wiazanie`

Zadaniem funkcji jest ocena możliwości związania za pomocą pól `kur_g` lub `kur_d` elementu dostarczonego funkcji za pośrednictwem pierwszego parametru z elementem dostarczonym w drugim parametrze i zwrócenie informacji logicznej zgodnej z tą oceną. Trzeci parametr logiczny podpowiada funkcji, jakim wiązaniem ma się zająć:

- `false` dotyczy związania polem `kur_d` elementów licznika z kreską ułamkową oraz elementów funkcji potęgującej ze znakiem potęgowania;
- `true` dotyczy związania polem `kur_g` elementów mianownika z kreską ułamkową, elementów podstawy logarytmu ze strukturą otwierającą oraz elementów funkcji podpierwiastkowej ze znakiem pierwiastkowania.

W drugim parametrze znajduje się adres elementu z wpisanym znakiem akcji, natomiast w pierwszym, element z jego otoczenia. Domyślnie funkcja zwraca wartość `false` zezwalającą na to związanie, lecz gdy okaże się, że element jest już związany ze wskazanym elementem w drugim parametrze, funkcja zwraca wartość `true`. Jeśli jednak sprawdzany element jest związany z innym elementem, zadaniem funkcji jest sprawdzenie, czy ten inny element znajduje się ponad elementem wskazanym w drugim parametrze, gdy badanie dotyczy wiąza-

nia polem kur_d lub poniżej tego elementu, gdy badanie dotyczy wiązania polem kur_g. Jeśli tak, funkcja zwraca wartość true, która jest informacją dla odpowiednich procedur wywołujących opisywaną funkcję, że wiązanie to musi pozostać nienaruszone. Opisane zabezpieczenia mają uchronić istniejące związania, w które mogą być wyposażone składniki należące np. do licznika jednego ułamka i do mianownika drugiego (ułamek piętrowy). Możliwych kombinacji łączenia składników polami kur_g i kur_d może być więcej, stąd tak skrupulatne warunki badania.

```
function wiazanie(const wpo,wpk:lisc;kierunek:boolean):boolean;
    var w2:lisc;
        x2:integer;
    begin
         Result:=false; - domyślne zwrócenie wartości false przez funkcję
         if (wpo=nil) or (wpk=nil) then exit; - jeśli chociaż jeden ...
                                z parametrów posiada adres pusty, opuść funkcję

         if kierunek then - czy parametr kierunek posiada wartość true?

         begin - jeśli tak, to zbadaj możliwość związania polem kur_g...
                  składników podanych w parametrach

              x2:=wpk^.y1; - odczytaj i zapamiętaj w zmiennej x2 wartość ...
                górnej współrzędnej rzeczywistej składnika z wpisanym znakiem akcji

              w2:=wpo^.kur_g; - do zmiennej w2 wpisz adres składnika...
                                związanego polem kur_g ze składnikiem...
                                podanym w pierwszym parametrze

              if w2<>nil then - czy składnik związany polem kur_g istnieje?
                 if w2=wpk then Result:=true - jeśli tak, to jeśli…
               składnik ten jest tym podanym w drugim parametrze, zwróć wartość true

                 else if w2^.y1>=x2 then Result:=true
                                jeśli nie jest..
                               to składnik podany w drugim parametrze, to gdy...
                               jego górna współrzędna rzeczywista jest większa...
                               od takiej samej współrzędnej składnika...
                               ze znakiem akcji, zwróć wartość true

         end else - jeśli nie, parametr kierunek posiada wartość false, więc...
         begin - zbadaj możliwość związania polem kur_d...
                  składników podanych w parametrach

              x2:=wpk^.y2; - odczytaj i zapamiętaj w zmiennej x2 wartość ...
                dolnej współrzędnej rzeczywistej składnika z wpisanym znakiem akcji

              w2:=wpo^.kur_d; - do zmiennej w2 wpisz adres składnika...
                                związanego polem kur_d ze składnikiem...
                                podanym w pierwszym parametrze

              if w2<>nil then - czy składnik związany polem kur_d istnieje?
                 if w2=wpk then Result:=true - (jak wyżej)
                 else if w2^.y2<=x2 then Result:=true
                                jeśli nie jest..
                               to składnik podany w drugim parametrze, to gdy...
                               jego dolna współrzędna rzeczywista jest mniejsza...
                               od takiej samej współrzędnej składnika...
                               ze znakiem akcji, zwróć wartość true

         end;
    end;
```

XIV. 2.7. Jaka jest długość licznika/mianownika? Funkcja `dlugosc`

Jest to prosta funkcja, której zadaniem jest znalezienie w pętli minimalnej i maksymalnej współrzędnej poziomej spośród elementów zawartych między pierwszym i drugim parametrem. Po opuszczeniu pętli, funkcja zwraca różnicę między największą a najmniejszą współrzędną, a więc długość wskazanej funkcji.

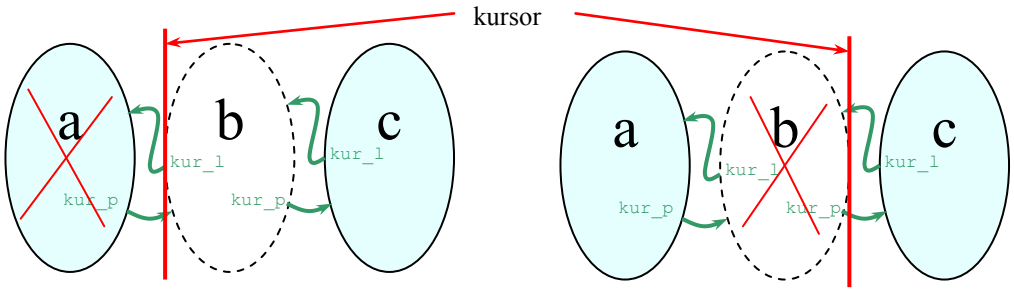
```
function dlugosc(const pocz4,kon4:lisc):integer;
var di1:lisc;
    dw1,dw2:integer;
begin
     Result:=0; - zwróć domyślną wartość równą zero
     if pocz4=nil then exit; - jeśli pierwszy parametr ma wpisany...
                                adres pusty, opuść funkcję
     di1:=pocz4; - zmiennej di1 wpisz adres zawarty w pierwszym parametrze
     dw1:=pocz4^.lx; - zmiennej dw1 wpisz początkową wartość współrzędnej,...
                       zawartej w polu lx pierwszego elementu
     dw2:=pocz4^.px; - zmiennej dw2 wpisz początkową wartość współrzędnej...
                       zawartej w polu px pierwszego elementu
     while di1<>kon4 do - wykonaj krok pętli, póki zmienna di1 nie uzyska...
                          adres drugiego parametru
     begin
          if di1^.lx<dw1 then dw1:=di1^.lx; - jeśli wartość w polu lx...
                       elementu analizowanego w pętli jest mniejsza od wartości...
                       w zmiennej dw1, wpisz do tej zmiennej wartość z tego pola
          if di1^.px>dw2 then dw2:=di1^.px; - jeśli wartość w polu px...
                       elementu analizowanego w pętli jest większa od wartości...
                       w zmiennej dw2, wpisz do tej zmiennej wartość z tego pola
          di1:=di1^.prawa; - w zmiennej di1 pozyskaj adres następnika...
                             w wiązaniu głównym
     end;
     Result:=dw2-dw1; - zwróć różnicę wartości zawartych w zmiennych: dw2 i dw1
end;
```

XIV. 3. Usunięcie znaku – procedura `Usun_znak`

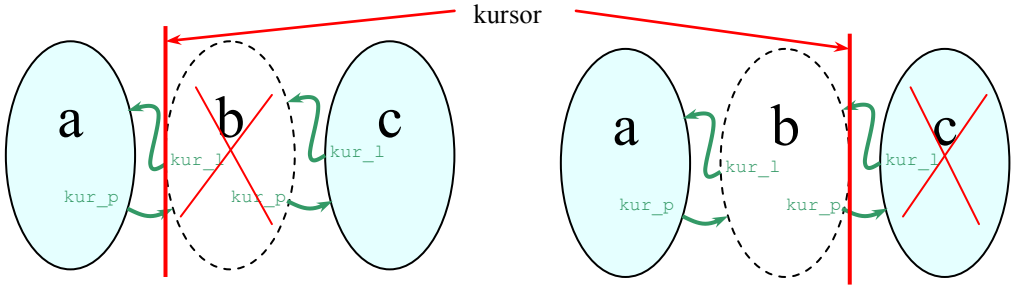
Procedura zajmuje się usunięciem wskazanego przez użytkownika znaku. Polega to na uwolnieniu pamięci zajmowanej przez składnik listy edytora, który w polu `znak` ma wpisany znak przeznaczony do usunięcia oraz, by zachować ciągłość listy, związania ze sobą pozostałych składników. Procedura posiada tylko jeden parametr logiczny o nazwie `tryb`, który podpowiada jej, który klawisz został użyty do usunięcia znaku:

- `true` wskazuje na klawisz BACKSPACE, co oznacza usunięcie znaku znajdującego się przed kursorem;
- `false` wskazuje na klawisz DELETE, co oznacza usunięcie znaku znajdującego się za kursorem.

Na początku ustalony zostaje składnik, który ma zostać usunięty z listy. Polega to na porównaniu ustawienia parametru `tryb` oraz zmiennej `stat_kur`. Wybór składnika najlepiej wyjaśnią rysunki 54 i 55:



Rys. 54. Wskazanie składnika przeznaczonego do usunięcia, po użyciu klawisza BACKSPACE i ustawienia kursora z lewej oraz z prawej strony znaku bieżącego, którego element został zaznaczony linią przerywaną



Rys. 55. Wskazanie składnika przeznaczonego do usunięcia, po użyciu klawisza DELETE i ustawienia kursora z lewej oraz z prawej strony znaku bieżącego, którego składnik został zaznaczony linią przerywaną

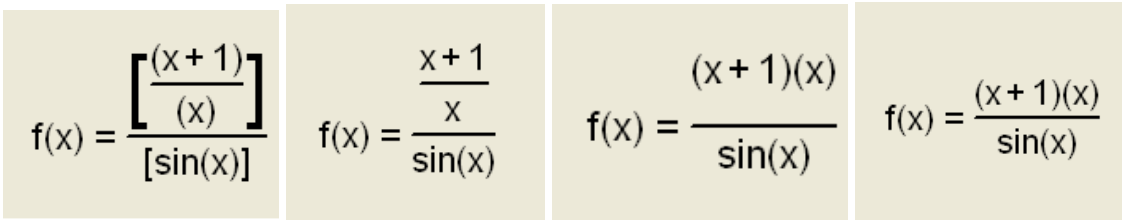
Gdy składnik został już ustalony, jego adres zostaje zachowany w zmiennej lokalnej `u1`. Teraz można już sprawdzić, czy znak wpisany w pole `znak` usuwanego elementu należy do znaków akcji, jeśli tak, to na podstawie wartości tego znaku wywoływana jest jedna z czterech wewnętrznych procedur, których zadaniem jest poprawne ułożenie i związanie ze sobą tych elementów, które bezpośrednio związane są z usuwanym składnikiem. Po powrocie z procedury, składnik wskazany przez wskaźnik `u1` zostaje usunięty z listy.

```
if tryb then - czy parametr tryb został ustawiony w pozycji true?
begin - jeśli tak, użyto klawisza BACKSPACE, czyli...
     if stat_kur then u1:=wsb else u1:=wsb^.kur_l; - jeśli zmienna...
                                 logiczna stat_kur ustawiona jest w pozycji true,...
                                 wpisz do zmiennej u1 adres elementu bieżącego,...
                                 w przeciwnym przypadku wpisz do tej zmiennej...
                                 adres jego poprzednika w wiązaniu kursorowym
     if u1<>nil then - czy wskaźnik u1 nie ma adresu pustego?
        if czy_log(u1) then - jeśli nie, to czy znak wpisany w jego pole znak...
                                    jest częścią nazwy funkcji 'log'?
        begin - jeśli tak, to...
             us_log(u1); - wywołaj procedurę usuwającą logarytm...
                           o dowolnej podstawie
             Form2.Odswiez_funkcje; - wywołaj procedurę odświeżającą...
                                      strukturę edytora po usunięciu logarytmu
             exit - opuść procedurę
```

```
          end else - jeśli nie jest to logarytm, to...
              if ord(u1^.znak) in dzed then - czy znak wpisany w pole...
                        znak usuwanego składnika listy należy do znaków akcji?
              begin - jeśli tak, to...
                   p1:=ord(u1^.znak); - zapamiętaj w zmiennej p1 wartość...
                                        tego znaku akcji
                   case p1 of - instrukcja wyboru
                        47:us_dziel(u1); - po rozpoznaniu znaku dzielenia...
                                           wywołaj procedurę us_dziel
                        92:us_pie(u1,true); - po rozpoznaniu znaku...
                             pierwiastkowania, wywołaj procedurę us_pie
                        else p1:=0; gdy nie rozpoznano znaku dzielenia...
                      lub pierwiastkowania, przypisz zmiennej p1 wartość zero
                   end;
                   if p1>0 then - czy wartość w zmiennej p1 jest większa od zera?
                   begin - jeśli tak, to...
                        dispose(u1); - usuń składnik listy edytora...
                                       wskazany przez wskaźnik u1
                        if not Form2.Ukryj1.Checked then - czy nawiasy...
                                                           zostały ukryte?
                        begin - jeśli tak, to...
                             Form2.Ukryj1.Checked:=true; - ustaw pole...
                                       Checked, należące do menu Ukryj1,...
                                       w pozycji true (odkryj nawiasy)
                             Form2.Odswiez_funkcje; - odśwież strukturę...
                                                      edytora
                             Form2.Ukryj1.Checked:=false - ustaw pole...
                                   Checked, w pozycji false (ukryj nawiasy)
                        end;
                        Form2.Odswiez_funkcje; - wywołaj procedurę...
                        odświeżającą strukturę edytora po usunięciu znaku akcji
                   end;
                   exit - opuść procedurę
              end
end else - jeśli nie, użyto klawisza DELETE, więc...
    begin
         if stat_kur then u1:=wsb^.kur_p else u1:=wsb; - jeśli zmienna...
                         logiczna stat_kur ustawiona jest w pozycji true, wpisz..
                         do zmiennej u1 adres następnika elementu bieżącego..
                         w wiązaniu kursorowym, w przeciwnym przypadku...
                         wpisz do tej zmiennej adres elementu bieżącego
         if u1<>nil then
            if czy_log(u1) then
            begin
                 us_log(u1);
                 Form2.Odswiez_funkcje;
                 exit
            end else
                if ord(u1^.znak) in dzed then
                begin
                     p1:=ord(u1^.znak);
                     case p1 of
                          47:us_dziel(u1); - po rozpoznaniu znaku...
                                dzielenia wywołaj procedurę us_dziel
                          94:us_pot(u1); - po rozpoznaniu znaku...
```

```
                                        potęgowania wywołaj procedurę us_pot
                                    92:us_pie(u1,false); - po rozpoznaniu znaku...
                                          pierwiastkowania wywołaj procedurę us_pie
                                    else p1:=0;
                               end;
                               if p1>0 then
                               begin
                                    dispose(u1);
                                    if not Form2.Ukryj1.Checked then
                                    begin
                                         Form2.Ukryj1.Checked:=true;
                                         Form2.Odswiez_funkcje;
                                         Form2.Ukryj1.Checked:=false
                                    end;
                                    Form2.Odswiez_funkcje
                               end;
                               exit
                          end
               end;
```

Dodatkowego wyjaśnienia wymaga potrzeba dwukrotnego wywołania procedury `Odswiez_funkcje` oraz przestawiania pola `Checked` elementu menu o nazwie `Ukryj1` służącego do ukrywania lub odkrywania nawiasów. Może się zdarzyć, że licznik i mianownik usuwanego ułamka zamknięty został w parze ukrytych nawiasów. Jeśli ten ułamek był jednocześnie licznikiem lub mianownikiem ułamka nadrzędnego (ułamek piętrowy), do ponownego pozycjonowania ułamka nadrzędnego wykorzystana została także para nawiasów, w której ułamek ten był zamknięty. Ponieważ nawiasy te nie należały do usuwanego ułamka, ich rozmiary pozostały niezmienione, czego efektem jest błędna pozycja elementów po usuniętym ułamku względem kreski ułamkowej ułamka nadrzędnego. By przywrócić właściwą wysokość ukrytym nawiasom po usuniętym znaku dzielenia, a tym samym zapewnić odpowiednią pozycję wszystkim elementom należącym do ułamka nadrzędnego, ukryte nawiasy muszą zostać odkryte przed odświeżeniem struktury edytora.



Rys. 56. Wyjaśnienie potrzeby odkrycia ukrytych nawiasów oraz dodatkowego odświeżenia struktury edytora po usunięciu znaku dzielenia należącego do ułamka podrzędnego

A – widok ułamka piętrowego z odkrytymi nawiasami
B – widok ułamka piętrowego z ukrytymi nawiasami
C – efekt usunięcia znaku dzielenia ułamka podrzędnego bez dodatkowego odświeżenia struktury edytora
D – efekt usunięcia znaku dzielenia ułamka podrzędnego w wyniku odświeżeniu struktury edytora po odkryciu nawiasów i po ponownym ich ukryciu

Jeśli w polu `znak` usuwanego składnika znajduje się znak nienależący do znaków akcji, postępowanie procedury zależne jest od wyniku analizy składników związanych z jego lewej i prawej strony wiązaniem kursorowym. Analiza rozpoczyna się od poprzednika usuwanego

składnika – jeśli poprzednik ten istnieje oraz gdy w jego polu `znak` znajduje się znak akcji lub jedna z liter składających się na nazwę funkcji ‘log’, sprawdzany jest następnik usuwanego składnika. Jeśli następnik ten nie istnieje lub gdy istnieje, ale w jego polu `znak` znajduje się znak dzielenia lub potęgowania, lub też jest to znak ukryty, oznacza to, że ustalony do usunięcia składnik nie może zostać usunięty, gdyż jego brak doprowadziłby do rozbicia otoczenia znaku akcji. Podobnie jest, gdy istnieje poprzednik w wiązaniu głównym, w którego polu `znak` znajduje się znak pierwiastkowania. W takich sytuacjach, znak wpisany w pole `znak` owego składnika zostaje zastąpiony spacją. Nie spełnienie wszystkich wymienionych warunków, pozostawia zmienną logiczną `czy1` w pozycji `false`, zezwalając na usunięcie wskazanego składnika z listy.

```
if u1<>nil then - czy składnik listy przeznaczony do usunięcia istnieje?
begin - jeśli tak, to...
     czy1:=false; - zainicjuj zmienną czy1 wartością false
     u3:=u1^.kur_l; - do zmiennej u3 wczytaj adres poprzednika...
                      usuwanego składnika w wiązaniu kursorowym
     u5:=u1^.kur_p; - do zmiennej u5 wczytaj adres następnika...
                      usuwanego składnika w wiązaniu kursorowym
     if u3<>nil then - czy poprzednik usuwanego składnika istnieje?
     begin - jeśli tak, to...
          if czy_log(u3) then czy1:=true; - jeśli w pole znak...
                              tego poprzednika wpisany jest znak składający się...
                              na nazwę ‘log’, ustaw zmienną czy1 w pozycji true
          if ord(u3^.znak) in dzed then czy1:=true; - jeśli...
                              w pole znak poprzednika wpisany jest znak akcji,...
                              ustaw zmienną czy1 w pozycji true
          if czy1 then - czy zmienna czy1 została ustawiona w pozycji true?
          begin - jeśli tak, to...
               czy1:=false; - ustaw zmienną czy1 w pozycji false
               if u5<>nil then - czy następnik usuwanego składnika istnieje?
               begin - jeśli tak, to...
                    u2:=u1^.lewa; - do zmiennej u2 wpisz adres...
                     poprzednika usuwanego składnika w wiązaniu głównym
                    if u2<>nil then - czy poprzednik istnieje?
                       if u2^.znak='\' then czy1:=true;
                  jeśli tak, to gdy w jego polu znak znajduje się znak pierwiastkowania,...
                  składnik przeznaczony do usunięcia jest elementem stopnia pierwiastka,...
                  dlatego nie wolno go usunąć – ustaw zmienną czy1 w pozycji true
                    if u5^.znak='/' then czy1:=true; - jeśli...
                          w pole znak tego następnika wpisany jest znak...
                          dzielenia, ustaw zmienną czy1 w pozycji true
                    if u5^.znak='^' then czy1:=true; - jeśli...
                     w pole znak następnika wpisany jest znak potęgowania,...
                     ustaw zmienną czy1 w pozycji true
                    if not u5^.druk then czy1:=true; - jeśli...
                                pole druk następnika jest ustawione...
                                w pozycji false (znak ukryty), to...
                                ustaw zmienną czy1 w pozycji true
                    if czy1 then - czy zmienna czy1 została ustawiona...
                                   w pozycji true?
                    begin - jeśli tak, to...
```

```
                    u1^.znak:=' '; - wpisz w pole znak...
                                     usuwanego składnika spację
                    u1^.px:=u1^.lx; - zredukuj szerokość znaku do zera
                    u1^.y1:=u1^.ly; } zrównaj współrzędne rzeczywiste...
                    u1^.y2:=u1^.py; } ze współrzędnymi poziomu
                    stat_kur:=true; - ustaw kursor za znakiem
                    Form2.Przesun(u1,true) - dosuń pozostałe znaki...
                                             w miejsce usuniętego znaku
               end else
          end else - jeśli następnik nie istnieje, to...
               begin
                    czy1:=true; - ustaw zmienną czy1 w pozycji true
                    u1^.znak:=' '; - wpisz w pole znak...
                                     usuwanego składnika spację
                    u1^.px:=u1^.lx; - zredukuj szerokość znaku do zera
                    u1^.y1:=u1^.ly; } zrównaj współrzędne rzeczywiste...
                    u1^.y2:=u1^.py; } ze współrzędnymi poziomu
                    wsb:=u3; - przypisz głównemu wskaźnikowi edytora...
                        adres poprzednika, czyniąc go elementem bieżącym
                    Form2.Przesun(u1,true) - dosuń pozostałe znaki...
                                             w miejsce usuniętego znaku
               end
     end else
end else
```

Brak poprzednika sprowadza się jedynie do przeprowadzenia analizy następnika usuwanego elementu. Jeśli więc następnik ten również nie istnieje lub gdy jest obecny, ale w jego polu `znak` znajduje się znak dzielenia lub potęgowania – bądź jego pole `druk` ustawione zostało w pozycji `false`, tak jak w poprzednim przykładzie – pole `znak` usuwanego składnika zostaje zastąpione spacją. Nie spełnienie tych warunków zezwala na usunięcie wskazanego składnika z listy, co zostaje zaznaczone przez pozostawienie w zmiennej `czy1` wartości `false`.

```
if u3<>nil then - czy poprzednik usuwanego składnika istnieje?
begin
     . . .
end else - jeśli nie istnieje, to...
    begin
         czy1:=false; - ustaw zmienną czy1 w pozycji false
         if u5<>nil then - czy następnik usuwanego składnika istnieje?
         begin - jeśli tak, to...
                (instrukcje identyczne do tych,...
                 zaprezentowanych w poprzednim przykładzie)
         end else - jeśli nie, to...
             begin
                  u1^.znak:=' '; - wpisz w pole znak...
                                   usuwanego składnika spację
                  u1^.px:=u1^.lx; - zredukuj szerokość znaku do zera
                  u1^.y1:=u1^.ly; } zrównaj współrzędne rzeczywiste...
                  u1^.y2:=u1^.py; } ze współrzędnymi poziomu
                  czy1:=true - ustaw zmienną czy1 w pozycji true
             end
    end;
```

W wyniku wykonania się poprzednio opisanych instrukcji, można mieć pewność, że gdy zmienna logiczna `czy1` jest ustawiona w pozycji `false`, istnieje na pewno poprzednik lub następnik usuwanego składnika w wiązaniu kursorowym lub istnieją obydwa te elementy. Oznacza to, że nie może zdarzyć się sytuacja, w której nie istnieją zarówno poprzednik, jak i następnik, gdyż ich brak zostałby wykryty w poprzednich instrukcjach. Informacja ta pozwala ograniczyć instrukcje warunkowe, sprawdzające istnienie tych elementów podczas ustalania nowego elementu bieżącego. Jeśli więc zmienna `czy1` nadal posiada wartość `false`, poza wybraniem elementu bieżącego przy którym będzie mógł pojawić się kursor, muszą zostać związane ze sobą te elementy edytora, które nadal związane są z usuwanym składnikiem. Wybór elementu bieżącego, czyli tego wskazywanego przez wskaźnik edytora `wsb`, wymagany jest tylko wtedy, gdy dotychczasowy element bieżący przeznaczony jest do usunięcia z listy. Brane są pod uwagę dwie możliwości:

- gdy do usunięcia znaku użyty został klawisz BACKSPACE a kursor usytuowany jest za znakiem, elementem bieżącym staje się poprzednik usuwanego składnika; a jeśli poprzednik ten nie istnieje, wybrany zostaje jego następnik, lecz w tym przypadku kursor musi znaleźć się przed znakiem;
- gdy do usunięcia znaku użyty został klawisz DELETE a kursor znajduje się przed znakiem, elementem bieżącym staje się następnik usuwanego składnika, a przy jego braku, wybrany zostaje jego poprzednik.

W pozostałych przypadkach, element bieżący dostępny przez wskaźnik `wsb` pozostaje niezmieniony. Jeśli do usuwanego składnika dowiązane zostały inne elementy listy za pomocą pól `kur_g` lub `kur_d`, jego adres zawarty w polach tych elementów zostaje zastąpiony adresem pustym. Pozostaje jeszcze związanie ze sobą pozostałych elementów listy zarówno wiązaniem kursorowym, jak i głównym oraz przesunięcie znaków w poziomie w miejsce usuwanego znaku. Bardzo ważnym punktem wiązania jest przypisanie wskaźnikowi `wsp` adresu pierwszego składnika listy, gdy nie istnieje poprzednik usuwanego składnika w wiązaniu głównym. W tym przypadku, pierwszym elementem listy będzie następnik usuwanego składnika i to jego adres musi znaleźć się w tym wskaźniku. Pozostawienie w nim adresu nieistniejącego elementu doprowadziłoby do załamania się programu, dlatego nie wolno pominąć tego punku.

Po wykonaniu wszystkich opisanych czynności, pozostaje usunięcie z pamięci elementu wskazywanego przez wskaźnik `u1`.

```
if u1<>nil then - czy składnik listy przeznaczony do usunięcia istnieje?
begin - jeśli tak, to...
     czy1:=false; - zainicjuj zmienną czy1 wartością false
     u3:=u1^.kur_l; - do zmiennej u3 wczytaj adres poprzednika...
                      usuwanego składnika w wiązaniu kursorowym
     u5:=u1^.kur_p; - do zmiennej u5 wczytaj adres następnika...
                      usuwanego składnika w wiązaniu kursorowym
     if u3<>nil then - czy poprzednik usuwanego składnika istnieje?
     begin
          . . .
     end else
         begin
              . . .
         end;
     if not czy1 then - czy zmienna logiczna czy1 jest ustawiona w pozycji false?
     begin - jeśli tak, to...
          if tryb then - czy parametr tryb jest ustawiony w pozycji true?
             if stat_kur then - jeśli tak, użyto klawisza BACKSPACE, więc...
                                czy zmienna stat_kur jest ustawiona...
```

```
                                  w pozycji true (kursor za znakiem)?
      if u3<>nil then wsb:=u3 - jeśli tak, to gdy wskaźnik u3...
                                  nie ma adresu pustego, wpisz jego adres...
                                  wskaźnikowi wsb (element bieżący)
      else - jeśli wskaźnik u3 ma adres pusty, to...
      begin
           wsb:=u5; - wpisz wskaźnikowi wsb adres zawarty...
                      we wskaźniku u5 (element bieżący)
           stat_kur:=false - ustaw zmienną stat_kur w pozycji..
                             false (kursor przed znakiem)
      end
   else
else - jeśli parametr tryb jest ustawiony w pozycji false, to...
if not stat_kur then - czy zmienna stat_kur jest ustawiona...
                       w pozycji false (kursor przed znakiem)?
   if u5<>nil then wsb:=u5 - jeśli tak, to gdy wskaźnik u5 nie ma...
    adresu pustego, to wpisz jego adres wskaźnikowi wsb (element bieżący)
   else wsb:=u3; - w przeciwnym przypadku, wpisz wskaźnikowi...
            wsb adres zawarty we wskaźniku u3 (element bieżący)
u2:=u1^.kur_g; - wczytaj do wskaźnika u2 adres zawarty w polu...
                 kur_g usuwanego składnika
if u2<>nil then - czy wskaźnik u2 nie jest pusty?
   if u2^.kur_d=u1 then u2^.kur_d:=nil; - jeśli nie, to...
            gdy element dostępny przez wskaźnik u2 ma wpisany...
            w polu kur_d adres usuwanego składnika, wpisz w to...
            pole identyfikator adresu pustego
u2:=u1^.kur_d; - wczytaj do wskaźnika u2 adres zawarty w polu...
                 kur_d usuwanego składnika
if u2<>nil then - czy wskaźnik u2 nie jest pusty?
   if u2^.kur_g=u1 then u2^.kur_g:=nil; - jeśli nie, to...
            gdy element dostępny przez wskaźnik u2 ma wpisany...
            w polu kur_g adres usuwanego składnika, wpisz w to...
            pole identyfikator adresu pustego
u2:=u1^.lewa; - wpisz do wskaźnika u2 adres poprzednika...
                usuwanego składnika w wiązaniu głównym
u4:=u1^.prawa; - wpisz do wskaźnika u2 adres następnika...
                 usuwanego składnika w wiązaniu głównym
if u2<>nil then u2^.prawa:=u4 else wsp:=u4; - jeśli...
                  poprzednik w wiązaniu głównym istnieje, zwiąż...
                  następnika z prawej strony, w przeciwnym...
                  przypadku, usuwany składnik jest pierwszym...
                  składnikiem listy, dlatego wpisz do wskaźnika...
                  wsp adres następnika zawarty we wskaźniku u4...
                  (teraz on będzie pierwszym składnikiem listy)
if u4<>nil then u4^.lewa:=u2; jeśli następnik w wiązaniu głównym...
                        istnieje, zwiąż z lewej strony poprzednika
if u3<>nil then u3^.kur_p:=u5; - jeśli poprzednik w wiązaniu...
           kursorowym istnieje, zwiąż z prawej strony jego następnika
if u5<>nil then - czy następnik w wiązaniu kursorowym istnieje?
begin - jeśli tak, to...
     u5^.kur_l:=u3; - zwiąż z lewej strony poprzednika
     p1:=u5^.px-u5^.lx; - wylicz w zmiennej p1 szerokość...
```

```
                                                    znaku zawartego w następniku
                    u5^.lx:=u1^.lx; - wpisz w pole lx następnika wartość...
                                      współrzędnej poziomej z tego samego pola,...
                                      należącego do usuwanego elementu
                    u5^.px:=u5^.lx+p1; - zmodyfikuj pole px następnika sumą...
                                         nowej współrzędnej lx i zapamiętanej...
                                         w zmiennej p1 szerokości znaku

                    Form2.Przesun(u5,true) - przesuń pozostałe znaki w prawo...
                        względem poprawionych współrzędnych poziomych następnika
                end;
                dispose(u1); - usuń z pamięci składnik listy, wskazany przez wskaźnik u1
            end
        end;
```

XIV. 3.1. Przygotowanie do usunięcia znaku dzielenia – procedura `us_dziel`

Zadaniem procedury jest ułożenie licznika i mianownika na poziomie usuwanej kreski ułamkowej, związanie ze sobą elementów licznika i mianownika oraz wybór elementu bieżącego. Na początku, w zmiennych liczbowych `st1` i `st2` wyliczane są różnice współrzędnych pionowych poziomów między odpowiednio: elementem znaku dzielenia i ostatnim elementem licznika oraz elementem znaku dzielenia i pierwszym elementem mianownika. W tym miejscu warto wspomnieć, że istnienie poprzednika i następnika elementu znaku dzielenia jest zagwarantowane zarówno przez procedurę `dzielenie`, podczas tworzenia otoczenia ułamka, jak i procedurę `usun_znak`, która nie pozwoli na usunięcie jedynego elementu licznika czy mianownika. Zapewnienie to pozwala na rezygnację z warunków sprawdzających istnienie tych elementów, co jest jednoznaczne z możliwością bezpośredniego odwołania się do ich pól. Na podstawie wyliczonych różnic, będzie wiadomo, o ile należy skorygować współrzędne pionowe wszystkich elementów licznika i mianownika, by obydwie części ułamka znalazły się w jednej linii, jeden przy drugim. W pierwszej kolejności badany jest licznik ułamka. Brane są pod uwagę trzy możliwości:

- ostatni element licznika zawiera w polu `znak` spację i jest to jedyny element licznika. Gdy ułamek przestanie istnieć, element ten będzie zbędny, dlatego jest on usuwany z listy. Pierwszy element mianownika oraz struktura otwierająca są ze sobą związane, a w pole `lewa` składnika otrzymanego w parametrze wpisany zostaje adres najbliższego składnika z lewej strony znaku dzielenia, czyli struktury otwierającej;
- ostatni element licznika zawiera w polu `znak` spację, ale nie jest on jedynym elementem licznika. Może być to struktura zamykająca, należąca do otoczenia znaku akcji znajdującego się w liczniku, dlatego nie wolno jej usunąć. Element ten jest wiązany wzajemnie z pierwszym elementem mianownika;
- ostatni element licznika zawiera w polu `znak` każdy inny znak niż spacja. Tak jak w poprzednim przypadku, element ten wiązany jest wzajemnie z pierwszym elementem mianownika.

Jeśli licznik istnieje, o czym świadczy niepusty wskaźnik `us1` otrzymujący adres poprzednika w wiązaniu głównym elementu przeznaczonego do usunięcia, wszystkie pola współrzędnych pionowych jego elementów zostają poprawione o wyliczoną różnicę zapamiętaną w zmiennej `st1`. Ewentualne ich związania z elementem znaku dzielenia poprzez pola

kur_d zostają usunięte, gdyż nie wolno pozostawiać w polach wiązań adresów nieistniejących elementów. Jeśli licznik zaopatrzony został w skrajną parę ukrytych nawiasów, nawiasy te zostają odkryte a pozostałe znaki przesunięte o szerokość tych nawiasów. Poprawianie elementów licznika odbywa się w pętli while..do.

Gdy struktura otwierająca zawiera w swym polu znak spację i element ten nie posiada swego poprzednika w wiązaniu kursorowym, struktura ta musi zostać usunięta. W tej sytuacji najbliższa, lewa strona kreski ułamkowej nie istnieje, dlatego w pole lewa składnika przeznaczonego do usunięcia, wpisany zostaje identyfikator adresu pustego. Będzie to wyraźna wskazówka podczas wyboru elementu bieżącego. Jeśli struktura otwierająca nie posiada także swego poprzednika w wiązaniu głównym, oznacza to, że jest ona pierwszym składnikiem listy, dlatego głównemu wskaźnikowi edytora wsp, pamiętającemu pierwszy element listy, przypisany zostaje adres następnika w wiązaniu głównym usuwanej struktury otwierającej.

```
procedure us_dziel(const odn1:lisc);
var us1,us2,us3,us4:lisc;
    ua1,st1,st2:integer;
begin
     us1:=odn1^.lewa; - zmiennej us1 wpisz adres poprzednika w wiązaniu...
                        głównym elementu znaku dzielenia, czyli parametru odn1
     us2:=odn1^.prawa; - zmiennej us2 wpisz adres następnika w wiązaniu...
                         głównym elementu znaku dzielenia
     st1:=odn1^.ly-us1^.ly; - w zmiennej st1 wylicz różnicę między...
                              współrzędnymi poziomu ly elementu znaku dzielenia...
                              i ostatniego elementu licznika
     st2:=odn1^.ly-us2^.ly; - w zmiennej st2 wylicz różnicę między...
                              współrzędnymi poziomu ly elementu znaku dzielenia...
                              i pierwszego elementu mianownika
     us3:=odn1^.kur_l; - zmiennej us3 wpisz adres struktury otwierającej
     if us1^.znak=' ' then - czy ostatni element licznika...
                            ma w polu znak spację?
        if us1^.lewa=us3 then - jeśli tak, to czy jego poprzednikiem...
                                w wiązaniu głównym jest struktura otwierająca?
        begin - jeśli tak, to jest to jedyny element licznika, czyli...
             odn1^.lewa:=us3; - w polu lewa elementu znaku dzielenia...
                                wpisz adres struktury otwierającej
             us4:=us1^.kur_g; - zmiennej us4 wpisz adres zawarty...
                                w polu kur_g
             if us4<>nil then - czy zmienna us4 nie ma adresu pustego?
                if us4^.kur_d=us1 then us4^.kur_d:=nil; - jeśli...
                          nie jest pusty, to gdy element wskazywany przez ten...
                          wskaźnik ma w swoim polu kur_d adres zawarty...
                          we wskaźniku us1, wpisz w pole kur_d tego elementu...
                          identyfikator adresu pustego
             us3^.prawa:=us2;  } zwiąż wzajemnie wiązaniem kursorowym...
             us3^.kur_p:=us2;  } i głównym strukturę otwierającą (us3)...
             us2^.lewa:=us3;   } oraz pierwszy element mianownika (us2)
             us2^.kur_l:=us3;  }
             dispose(us1); - usuń z pamięci element licznika...
                             wskazywany przez wskaźnik us1
             us1:=nil - przypisz wskaźnikowi us1 identyfikator adresu pustego
        end else - jeśli nie jest to jedyny element licznika, to...
            begin
```

```
            us1^.prawa:=us2;
            us1^.kur_p:=us2;
            us2^.lewa:=us1;
            us2^.kur_l:=us1
        end
else
Begin
```

zwiąż wzajemnie wiązaniem głównym...
i kursorowym ostatni element licznika...
i pierwszy mianownika

```
     us1^.prawa:=us2;
     us1^.kur_p:=us2;
     us2^.lewa:=us1;
     us2^.kur_l:=us1
end;
if us1<>nil then - czy licznik istnieje?
   while us1<>us3 do - wykonaj krok pętli, póki zmienna us1...
                       nie ma adresu struktury otwierającej (us3)
   begin
        if us1^.kur_d=odn1 then us1^.kur_d:=nil; - jeśli...
                element wskazywany przez wskaźnik us1 posiada w polu...
                kur_d adres elementu znaku dzielenia, wpisz w to pole...
                identyfikator adresu pustego
        if not us1^.druk then - czy pole druk zostało ustawione...
                                w pozycji false?
        begin - jeśli tak, to element jest ukryty, czyli...
             us1^.druk:=true; - ustaw pole druk w pozycji true
             Form2.Szerokosc(us1); - przywróć szerokość znakowi
             Form2.Przesun(us1,true) - przesuń w prawo pozostałe...
                                       znaki  o przywróconą szerokość
        end;
        with us1^ do
        begin
             ly:=ly+st1;
             py:=py+st1;
             y1:=y1+st1;
             y2:=y2+st1
        end;
        us1:=us1^.lewa - w zmiennej us1 pozyskaj adres poprzednika...
                         w wiązaniu głównym
   end;
if us3^.znak=' ' then - czy struktura otwierająca ma w polu znak spację?
   if us3^.kur_l=nil then - jeśli tak, to czy nie istnieje poprzednik...
                            w wiązaniu kursorowym?
   begin - jeśli brak poprzednika w wiązaniu kursorowym, to...
        if odn1^.lewa=us3 then odn1^.lewa:=nil; - jeśli w polu...
                         lewa elementu wskazywanego przez parametr...
                         odn1 wpisany jest adres struktury otwierającej,...
                         wstaw w to pole identyfikator adresu pustego
        us4:=us3^.kur_g;
        if us4<>nil then
           if us4^.kur_d=us3 then us4^.kur_d:=nil;
        us4:=us3^.kur_d;
        if us4<>nil then
           if us4^.kur_g=us3 then us4^.kur_g:=nil;
        usuń związania struktury otwierającej z innymi elementami...
        poprzez pola kur_g i kur_d
        us4:=us3^.lewa; - zmiennej us4 wpisz adres poprzednika...
```

zmodyfikuj współrzędne pionowe o wyliczoną..
w zmiennej st1 różnicę

```
                                    struktury otwierającej w wiązaniu głównym
            us1:=us3^.prawa; - zmiennej us4 wpisz adres następnika...
                                    struktury otwierającej w wiązaniu głównym

            us1^.kur_l:=nil; - następnik dostępny przez wskaźnik us1...
                                nie ma już poprzednika w wiązaniu kursorowym,...
                                dlatego jego pole kur_l zainicjuj adresem pustym


            us1^.lewa:=us4; - zwiąż możliwy poprzednik struktury...
                                otwierającej z następnikiem wiązaniem głównym

            if us4<>nil then us4^.prawa:=us1 else wsp:=us1;
            jeśli poprzednik (us4) struktury otwierającej istnieje, zwiąż następnika...
            (us1) z tym poprzednikiem wiązaniem głównym, jeśli poprzednik...
            nie istnieje, następnik jest pierwszym składnikiem listy, dlatego...
            wpisz jego adres wskaźnikowi edytora wsp
            ua1:=us1^.px-us1^.lx; - w zmiennej ua1 wylicz szerokość...
                                                znaku następnika

            us1^.lx:=us3^.lx; - następnikowi (us1) wpisz wartość...
             początkowej współrzędnej poziomej struktury otwierającej (us3)

            us1^.px:=us1^.lx+ua1; - wpisz do końcowej współrzędnej...
                                poziomej następnika sumę wartości poprawionej,...
                                początkowej współrzędnej i szerokości znaku

            Form2.Przesun(us1,true); - dosuń pozostałe znaki...
                                                do znaku następnika

            dispose(us3) - usuń z pamięci element lisc – strukturę otwierającą
        end else
    else
    begin
         us4:=us3^.prawa; - wpisz do zmiennej us4 adres następnika...
                                struktury otwierającej w wiązaniu głównym

         us4^.kur_l:=us3;  } zwiąż wzajemnie wiązaniem kursorowym
         us3^.kur_p:=us4;  } strukturę otwierającą i jej następnika

         Form2.Przesun(us3,true) - dosuń pozostałe znaki do znaku...
                                            w strukturze otwierającej
    end;
```

Skoro lewa strona kreski ułamkowej została już zwymiarowana i związana z jej prawą stroną, pora na wymiarowanie mianownika i struktury zamykającej ułamek. W przypadku pierwszego elementu mianownika, badane są tylko dwie możliwości:

– w polu `znak` pierwszego elementu mianownika znajduje się spacja. Jeśli jest to jedyny element mianownika, o czym świadczy wpisany w polu `prawa` adres struktury zamykającej, element ten musi zostać usunięty z listy, gdyż obecność spacji między ostatnim znakiem licznika a następnym znakiem, byłaby niepożądana. Jeśli istnieje element znajdujący się pod znakiem mianownika, czyli gdy w polu `kur_d` elementu mianownika znajduje się adres innego elementu listy, adres wpisany w pole `kur_g` tego elementu musi być zastąpiony adresem pustym. Lewa strona kreski ułamkowej zostaje związana ze strukturą zamykającą, zapewniając tym związaniem ciągłość listy. Najbliższą, prawą stroną kreski ułamkowej jest w tej sytuacji struktura zamykająca i to jej adres wpisany zostaje w pole `prawa` parametru `odn1`. Gdyby okazało się, że usuwany składnik nie posiada swego poprzednika w wiązaniu głównym, do wskaźnikowa edytora `wsp` wpisany zostaje adres struktury zamykającej, która po usunięciu jedynego elementu mianownika stanie się pierwszym składnikiem listy;

– obecność każdego innego znaku niż spacja, zawartego w polu `znak` pierwszego elementu mianownika, sprowadza się do wyliczenia różnicy w zmiennej `ua1` między końcową współrzędną poziomą `px` elementu należącego do lewej strony kreski ułamkowej (standardowo jest to ostatni element licznika) a początkową współrzędną poziomą `lx` pierwszego elementu mianownika. Jeśli lewa strona kreski ułamkowej nie istnieje, zastępuje ją współrzędna startowa `AX`. Uruchamiana jest pętla, w której wszystkim elementom mianownika modyfikowane są współrzędne pionowe i poziome w ten sposób, że do dotychczasowej wartości współrzędnej dodawana jest różnica wyliczona w zmiennych liczbowych `st2` i `ua1`. Skrajne nawiasy mianownika, które były ukryte, teraz zostają odkryte. Po opuszczeniu pętli, ostatni element mianownika i struktura zamykająca, są ze sobą związane wiązaniem kursorowym.

Jeśli struktura zamykająca ma w swoim polu `znak` spację i nie ma ona swego następnika w wiązaniu kursorowym, element ten też musi zostać usunięty z listy, ale pod warunkiem, że istnieje lewa strona kreski ułamkowej. Ostatni warunek gwarantuje obecność przynajmniej jednego składnika listy, gdyby wszystkie pozostałe części ułamka zostały usunięte. Obecność następnika w wiązaniu kursorowym może stanowić niebezpieczeństwo, że następnik ten ma wpisany znak potęgowania, stąd zabezpieczenie warunkowe przed rozbiciem otoczenia tego znaku akcji. Jeśli warunki pozwalają na usunięcie struktury zamykającej, wszystkie związania innych elementów z tą strukturą polami `kur_g` i `kur_d` muszą zostać usunięte. Po związaniu ze sobą poprzedników i następników struktury zamykającej w wiązaniu kursorowym i głównym, struktura ta zostaje usunięta z listy. Nie spełnienie choć jednego z powyższych warunków powoduje przesunięcie znaku znajdującego się w polu `znak` tej struktury do znaku zawartego w jej poprzedniku.

```
us3:=odn1^.kur_p; – do zmiennej us3 wczytaj adres struktury zamykającej
if us2^.znak=' ' then – czy pierwszy element mianownika ma wpisany znak spacji?
   if us2^.prawa=us3 then – jeśli tak, to czy w polu prawa zawarty jest...
                            adres struktury zamykającej?

   begin – jeśli tak, jest to jedyny element mianownika, czyli...
        odn1^.prawa:=us3; – w pole prawa parametru odn1 wpisz adres...
                            struktury zamykającej

        us4:=us2^.kur_d; – do zmiennej us4 wpisz adres elementu...
                           związanego z mianownikiem polem kur_d

        if us4<>nil then – czy wskaźnik us4 nie jest pusty?
           if us4^.kur_g=us2 then us4^.kur_g:=nil; – jeśli nie, to...
                            w pole kur_g elementu, na który on wskazuje,...
                            wpisz identyfikator adresu pustego

        us4:=us2^.lewa; – do zmiennej us4 wpisz adres poprzednika...
                          elementu mianownika w wiązaniu głównym

        us3^.lewa:=us4; – w pole lewa struktury zamykającej wpisz adres...
                          elementu zawartego we wskaźniku us4

        if us4<>nil then us4^.prawa:=us3 else wsp:=us3;
   jeżeli poprzednik istnieje, wpisz w jego pole prawa adres struktury zamykającej,...
   gdy poprzednik nie istnieje, wpisz wskaźnikowi edytora wsp adres struktury...
   zamykającej – pierwszy składnik listy

        us4:=us2^.kur_l; – wpisz zmiennej us4 adres elementu należącego...
                           do lewej strony kreski ułamkowej

        us3^.kur_l:=us4; – wpisz w pole kur_l struktury zamykającej...
                           adres zawarty we wskaźniku us4
```

```
        if us4<>nil then us4^.kur_p:=us3; - jeśli wskaźnik us4...
                                            nie ma adresu pustego, wpisz w pole kur_p...
                                            elementu na który on wskazuje, adres...
                                            struktury zamykającej
        dispose(us2) - usuń z pamięci element mianownika wskazywany przez...
                       wskaźnik us2
   end else
else - pierwszy element mianownika ma wpisany inny znak niż spacja, więc...
begin
     us1:=us2; - zachowaj adres pierwszego elementu mianownika w zmiennej us1
     us4:=us1^.lewa; - zmiennej us4 wpisz adres poprzednika elementu...
                       mianownika w wiązaniu głównym
     if us4<>nil then ua1:=us4^.px-us1^.lx - jeżeli poprzednik istnieje,...
                                             w zmiennej ua1 wylicz różnicę między współrzędną...
                                             poziomą px poprzednika a współrzędną lx pierwszego...
                                             elementu mianownika
     else ua1:=AX-us1^.lx; - jeśli poprzednik nie istnieje, wylicz różnicę...
                             między współrzędną startową zawartą w zmiennej AX...
                             a współrzędną pozioma lx pierwszego elementu mianownika
     repeat
          if us1^.kur_g=odn1 then us1^.kur_g:=nil; - jeżeli element...
                                  wskazywany przez wskaźnik us1 posiada w polu kur_g...
                                  adres elementu znaku dzielenia, wpisz w to pole...
                                  identyfikator adresu pustego
          if not us1^.druk then - czy pole druk ustawione jest w pozycji...
                                  false?
          begin - jeśli tak, znak jest ukryty, czyli...
               us1^.druk:=true; - odkryj znak ustawiając pole druk...
                                  w pozycji true
               Form2.Szerokosc(us1); - przywróć szerokość znakowi
               Form2.Przesun(us1,true) - przesuń pozostałe znaki...
                                         w prawo o tę szerokość
          end;
          with us1^ do
          begin
               lx:=lx+ua1;  } zmodyfikuj współrzędne poziome jako suma
               px:=px+ua1;  } dotychczasowych wartości i wyliczonej różnicy
               ly:=ly+st2;
               py:=py+st2;  } zmodyfikuj współrzędne pionowe jako suma
               y1:=y1+st2;  } dotychczasowych wartości i wyliczonej różnicy
               y2:=y2+st2
          end;
          us4:=us1; - zapamiętaj w zmiennej us4 adres ostatniego...
                      elementu mianownika
          us1:=us1^.prawa  w zmiennej us1 pozyskaj adres następnika
     until us1=us3; - opuść pętlę, gdy zmienna us1 osiągnie adres us3
     us4^.kur_p:=us3;  } związ wzajemnie wiązaniem kursorowym ostatni...
     us3^.kur_l:=us4   } element mianownika i strukturę zamykającą
end;
us1:=us3^.lewa; - wpisz do zmiennej us1 adres poprzednika struktury...
                  zamykającej w wiązaniu głównym
if us1<>nil then - czy adres zawarty w zmiennej us1 nie jest pusty?
   if us3^.znak=' ' then - jeśli nie, to czy struktura zamykająca ma...
                           w polu znak spację?
```

```
        if us3^.kur_p=nil then - jeśli tak, to czy w polu kur_p jest adres pusty?
        begin - jeśli tak, to...
              us4:=us3^.kur_g;
              if us4<>nil then
                 if us4^.kur_d=us3 then us4^.kur_d:=nil;
              us4:=us3^.kur_d;
              if us4<>nil then
                 if us4^.kur_g=us3 then us4^.kur_g:=nil;
                 usuń związania struktury zamykającej z innymi elementami...
                 poprzez pola kur_g i kur_d

              us4:=us3^.prawa; - do zmiennej us4 wpisz adres następnika...
                                 struktury zamykającej w wiązaniu głównym

              us1^.prawa:=us4; - zwiąż następnika z poprzednikiem...
                                 struktury zamykającej w wiązaniu głównym

              us1^.kur_p:=nil; - poprzednik nie ma swego następnika w wiązaniu...
               kursorowym, dlatego wpisz w jego pole kur_p identyfikator adresu pustego

              if us4<>nil then us4^.lewa:=us1; - jeżeli następnik...
                         w wiązaniu głównym struktury zamykającej istnieje,...
                         zwiąż poprzednika z tym następnikiem

              dispose(us3) - usuń z pamięci składnik listy wskazywany przez...
                             wskaźnik us3

        end else Form2.Przesun(us1,true) - jeśli istnieje następnik struktury...
                         zamykającej w wiązaniu kursorowym, przesuń pozostałe znaki...
                         do znaku wpisanego w tę strukturę

   else Form2.Przesun(us1,true); - jeśli znak wpisany w pole znak...
              struktury zamykającej jest inny niż spacja, przesuń pozostałe znaki...
              do znaku wpisanego w tę strukturę
```

Na koniec pozostało ustalenie elementu bieżącego, czyli tego, przy którym ma pojawić się kursor. Początek tego ustalania miał już miejsce podczas badania licznika oraz mianownika, w wyniku czego zapamiętane zostały dwa najbliższe elementy lewej i prawej strony kreski ułamkowej. Jeśli więc lewa strona istnieje, o czym świadczy niepusty adres zawarty w polu `lewa` składnika dostępnego przez parametr `odn1`, elementem bieżącym staje się ten, którego adres widnieje w tym polu. Ponadto zmienna `stat_kur` ustawiona zostaje w pozycji `true`, co spowoduje, że kursor będzie ustawiony za znakiem. Jeżeli lewa strona nie istnieje, elementem bieżącym staje się ten, którego adres wpisany został w pole `prawa` owego parametru. Zmienna `stat_kur` ustawiona zostaje w pozycji `false`, co sprawi, że kursor znajdzie się przed znakiem ustawiająca kursor przed znakiem.

```
if odn1^.lewa<>nil then - czy pole lewa elementu wskazywanego przez...
                          parametr odn1 nie posiada adresu pustego?

begin - jeśli nie, to...
     wsb:=odn1^.lewa; - wpisz wskaźnikowi edytora wsb adres zawarty...
                        w polu lewa

     stat_kur:=true - ustaw zmienną stat_kur w pozycji true
end else - jeśli pole lewa ma wpisany adres pusty, to...
    begin
         wsb:=odn1^.prawa; - wpisz wskaźnikowi edytora wsb adres...
                             zawarty w polu prawa

         stat_kur:=false - ustaw zmienną stat_kur w pozycji false
    end;
```

XIV. 3.2. Przygotowanie do usunięcia znaku pierwiastkowania – procedura `us_pie`

Jej zadaniem jest usunięcie z listy elementu stopnia pierwiastka, ułożenie funkcji podpierwiastkowej na poziomie znaku pierwiastkowania oraz wybór elementu bieżącego. Jeżeli element znaku pierwiastkowania posiada swego poprzednika w wiązaniu głównym, pierwszy element funkcji podpierwiastkowej zostaje wzajemnie związany z tym poprzednikiem, w przeciwnym przypadku usuwany element znaku pierwiastkowania jest na pewno pierwszym składnikiem listy edytora. W takiej sytuacji należy zadbać o to, by nie pozostawiać we wskaźniku edytora `wsp` adresu składnika, który za chwilę przestanie istnieć. Dlatego wskaźnikowi temu wpisany zostaje adres pierwszego elementu funkcji podpierwiastkowej i to ten element stanie się pierwszym składnikiem listy, gdy składnik przeznaczony do usunięcia faktycznie przestanie istnieć.

```
procedure us_pie(const odn2:lisc;tryb2:boolean);
var usp1,usp2,usp3,usp4:lisc;
    su1,su2:integer;
begin
     usp1:=odn2^.prawa; - do zmiennej usp1 wpisz adres elementu...
                          stopnia pierwiastka
     usp2:=usp1^.prawa; - do zmiennej usp2 wpisz adres pierwszego elementu...
                          funkcji podpierwiastkowej
     usp3:=odn2^.lewa; - do zmiennej usp3 wpisz adres poprzednika elementu...
                         znaku pierwiastkowania w wiązaniu głównym
     usp4:=usp1^.kur_g; - do zmiennej usp4 wpisz adres zawarty w polu...
                          kur_g elementu stopnia pierwiastka...
                          (znak umieszczony nad pierwiastkiem)
     if usp4<>nil then - czy wskaźnik usp4 nie ma adresu pustego?
        if usp4^.kur_d=usp1 then usp4^.kur_d:=nil; - jeśli nie, to...
           wpisz w pole kur_d elementu wskazywanego przez ten wskaźnik...
           identyfikator adresu pustego, pod warunkiem, że w tym polu...
           widnieje adres elementu stopnia pierwiastka
     usp4:=odn2^.kur_d; - do zmiennej usp4 wpisz adres zawarty w polu...
                          kur_d elementu znaku pierwiastkowania...
                          (znak umieszczony pod pierwiastkiem)
     if usp4<>nil then - czy wskaźnik usp4 nie ma adresu pustego?
        if usp4^.kur_g=odn2 then usp4^.kur_g:=nil; - jeśli nie, to...
           wpisz w pole kur_g elementu wskazywanego przez ten wskaźnik...
           identyfikator adresu pustego, pod warunkiem, że w tym polu...
           widnieje adres elementu znaku pierwiastkowania
     dispose(usp1); - usuń z pamięci element stopnia pierwiastka
     usp2^.lewa:=usp3; - wpisz w pole lewa, pierwszego elementu funkcji...
          podpierwiastkowej, adres poprzednika elementu znaku pierwiastkowania
     if usp3<>nil then usp3^.prawa:=usp2 else wsp:=usp2; - jeżeli...
          wskaźnik usp3 nie zawiera adresu pustego, wpisz w pole prawa poprzednika...
          adres pierwszego elementu funkcji podpierwiastkowej, gdy wskaźnik ten...
          zawiera adres pusty, wpisz wskaźnikowi edytora wsp adres pierwszego...
          elementu funkcji podpierwiastkowej
     usp1:=odn2^.kur_l; - do zmiennej usp1 wpisz adres poprzednika elementu...
                          znaku pierwiastkowania w wiązaniu kursorowym
     if usp1<>nil then - czy poprzednik istnieje?
     begin - jeśli tak, to...
```

```
            usp1^.kur_p:=usp2;  } związ wzajemnie wiązaniem kursorowym tego...
            usp2^.kur_l:=usp1   } poprzednika i element funkcji podpierwiastkowej
      end;
```

W zmiennej liczbowej `su1` wyliczana jest różnica między współrzędnymi pionowymi `ly` elementów: znaku pierwiastkowania i funkcji podpierwiastkowej, natomiast w zmiennej `su2` – różnica między współrzędnymi poziomymi `lx` tych elementów. Na podstawie uzyskanych różnic wyliczane są nowe wartości, które wpisywane są w pola współrzędnych elementów funkcji podpierwiastkowej. W wyniku poprawionych wartości współrzędnych, funkcja podpierwiastkowa przyjmuje początkową pozycję znaku pierwiastkowania. Ponieważ standardowo, pierwszy znak funkcji podpierwiastkowej zawiera w polu `znak` ukryty nawias otwierający, natomiast ostatni znak – ukryty nawias zamykający, nawiasy te zostają odkryte a wszystkie pozostałe znaki przesuwane są w poziomie o wielkość odtworzonej szerokości tych nawiasów.

```
su1:=odn2^.ly-usp2^.ly; - w zmiennej su1 wylicz różnicę między współrzędną...
                          poziomu ly elementu znaku pierwiastkowania (odn2)...
                          a pierwszym elementem funkcji podpierwiastkowej (usp2)
su2:=odn2^.lx-usp2^.lx; - w zmiennej su2 wylicz różnicę między współrzędną...
                          poziomą lx elementu znaku pierwiastkowania (odn2)...
                          a pierwszym elementem funkcji podpierwiastkowej (usp2)
if not usp2^.druk then - czy pole druk pierwszego elementu funkcji...
                         podpierwiastkowej jest ustawione w pozycji false?
begin - jeśli tak, to znak jest ukryty, czyli...
     usp2^.druk:=true; - ustaw pole druk w pozycji true
     Form2.Szerokosc(usp2); - przywróć szerokość ukrytemu nawiasowi
     Form2.Przesun(usp2,true) - przesuń pozostałe znaki w prawo...
                                względem odkrytego nawiasu
end;
usp3:=odn2^.kur_p; - do zmiennej usp3 wpisz adres struktury zamykającej
usp1:=usp2; - zachowaj w zmiennej usp1 adres pierwszego elementu...
              funkcji podpierwiastkowej
repeat
      with usp1^ do
      begin
           lx:=lx+su2;  } zmodyfikuj współrzędne poziome jako suma
           px:=px+su2;  } dotychczasowych wartości i wyliczonej różnicy
           ly:=ly+su1;
           py:=py+su1;  } zmodyfikuj współrzędne pionowe jako suma
           y1:=y1+su1;  } dotychczasowych wartości i wyliczonej różnicy
           y2:=y2+su1
      end;
      if usp1^.kur_g=odn2 then usp1^.kur_g:=nil; - jeżeli w polu...
        kur_g elementu funkcji podpierwiastkowej, wskazywanego przez wskaźnik...
        usp1, znajduje się adres elementu znaku pierwiastkowania, wpisz w to pole...
        identyfikator adresu pustego
      usp4:=usp1; - zachowaj w zmiennej usp4 adres ostatniego elementu...
                    funkcji podpierwiastkowej
      usp1:=usp1^.prawa - w zmiennej usp1 pozyskaj adres następnika...
                          w wiązaniu głównym
until usp1=usp3; - opuść pętlę, gdy wskaźnik usp1 uzyska adres...
                   struktury zamykającej
```

```
if not usp4^.druk then - czy pole druk ostatniego elementu funkcji...
                         podpierwiastkowej zostało ustawione w pozycji false?
begin - jeśli tak, to znak jest ukryty, czyli...
     usp4^.druk:=true; - ustaw pole druk w pozycji true
     Form2.Szerokosc(usp4); - przywróć szerokość ukrytemu nawiasowi
end;
```

Funkcja podpierwiastkowa jest już przygotowana do usunięcia znaku pierwiastkowania, pozostało jeszcze zbadanie struktury zamykającej. Jeśli w polu `znak` tej struktury znajduje się spacja a struktura ta nie ma swego następnika w wiązaniu kursorowym, jest ona usunięta z listy. Nieco inaczej wygląda sytuacja, gdy struktura ta ma swego następnika w wiązaniu kursorowym. Obecność spacji może sugerować istnienie znaku akcji w polu `znak` tego następnika, dlatego po usunięciu tej struktury zamykającej, rolę struktury otwierającej dla znaku akcji przejmuje ostatni element funkcji podpierwiastkowej. Obydwa elementy muszą być zatem poprawnie związane, by zachować nienaruszone otoczenie tego znaku akcji.

Jeśli struktura zamykająca ma wpisany inny znak niż spacja, jest ona związana kursorowo z ostatnim elementem funkcji podpierwiastkowej.

```
if usp3^.znak=' ' then - czy w polu znak struktury zamykającej...
                         wpisana jest spacja?
begin - jeśli tak, to...
     usp1:=usp3^.kur_g;
     if usp1<>nil then
        if usp1^.kur_d=usp3 then usp1^.kur_d:=nil;
     usp1:=usp3^.kur_d;
     if usp1<>nil then
        if usp1^.kur_g=usp3 then usp1^.kur_g:=nil;
      usuń adresy struktury zamykającej, zawarte w polach...
      kur_g i kur_d istniejących składników listy, wpisując w te pola...
      identyfikatory adresu pustego
     usp1:=usp3^.kur_p; - do zmiennej usp1 wczytaj adres następnika...
                          struktury zamykającej w wiązaniu kursorowym
     if usp1<>nil then - czy następnik istnieje?
     begin - jeśli tak, to...
          usp4^.kur_p:=usp1; } związ wzajemnie wiązaniem kursorowym ostatni...
          usp1^.kur_l:=usp4; } element funkcji podpierwiastkowej z następnikiem
          usp1:=usp3^.prawa; - do zmiennej usp1 wpisz adres następnika...
                               struktury zamykającej w wiązaniu głównym
          usp1^.lewa:=usp4; } związ wzajemnie wiązaniem głównym następnika..
          usp4^.prawa:=usp1; } z ostatnim elementem funkcji podpierwiastkowej
          dispose(usp3) - usuń z pamięci strukturę zamykającą
     end else - jeśli następnik nie istnieje, to...
         begin
              usp1:=usp3^.prawa; - do zmiennej usp1 wpisz adres następnika...
                                   struktury zamykającej w wiązaniu głównym
              if usp1<>nil then usp1^.lewa:=usp4; - jeśli następnik...
                      istnieje, wpisz w jego pole lewa adres ostatniego elementu...
                      funkcji podpierwiastkowej
              usp4^.prawa:=usp1; w pole prawa tego elementu...
                                 wpisz adres następnika
              dispose(usp3) - usuń z pamięci strukturę zamykającą
         end
end else - jeśli w polu znak struktury zamykającej znajduje się inny znak niż spacja, to...
```

```
begin
     usp4^.kur_p:=usp3;  } związ wzajemnie wiązaniem kursorowym...
     usp3^.kur_l:=usp4;  } strukturę zamykającą i ostatni element...
                           funkcji podpierwiastkowej

     Form2.Przesun(usp4,true) - dosuń znak zawarty w strukturze...
                    zamykającej oraz następne znaki do ostatniego znaku...
                    funkcji podpierwiastkowej
end;
```

Ostatnią czynnością procedury jest ustalenie składnika bieżącego. W przypadku tej procedury, wybór uzależniony jest od wartości drugiego parametru o nazwie `tryb2`. Jego wartość podpowiada procedurze, jaki klawisz klawiatury został użyty do usunięcia znaku pierwiastkowania:

- `true` oznacza, że został naciśnięty klawisz BACKSPACE. Kursor musiał więc być usytuowany za znakiem pierwiastka. W tej sytuacji, najlepszym wyborem składnika jako bieżącego jest niewątpliwie ostatni element funkcji podpierwiastkowej przy kursorze ustawionym za znakiem;
- `false` oznacza, że został naciśnięty klawisz DELETE. Kursor musiał znajdować się przed znakiem pierwiastka, dlatego jako składnik bieżący wybrany zostaje pierwszy element funkcji podpierwiastkowej przy kursorze ustawionym przed znakiem.

```
if tryb2 then - czy parametr tryb2 jest ustawiony w pozycji true?
begin - jeśli tak, użyto klawisza BACKSPACE, więc...
     wsb:=usp4; - zmiennej edytora wsb wpisz adres ostatniego elementu...
                  funkcji podpierwiastkowej

     stat_kur:=true - zmienną stat_kur ustaw w pozycji true...
                      (kursor za znakiem)

end else - jeśli nie, użyto klawisza DELETE, więc...
     begin
          wsb:=usp2; - zmiennej edytora wsb wpisz adres pierwszego...
                       elementu funkcji podpierwiastkowej

          stat_kur:=false - zmienną stat_kur ustaw w pozycji false...
                            (kursor przed znakiem)
     end;
```

XIV. 3.3. Przygotowanie do usunięcia znaku potęgowania – procedura `us_pot`

Zadaniem procedury jest przywrócenie znakom funkcji potęgującej ich pierwotnego rozmiaru oraz przesunięcie tej funkcji jako całości do takiego poziomu, na jakim znajduje się ukryty znak potęgowania. Ponadto pomiędzy strukturą otwierającą a pierwszym elementem funkcji potęgującej musi istnieć obustronne związanie kursorowe, podobnie jak między ostatnim elementem tej funkcji a strukturą zamykającą, co umożliwi płynne poruszanie się kursorem między sąsiadującymi znakami. Podczas analizy funkcji potęgującej najważniejszy jest jej pierwszy element, a konkretnie znak wpisany w jego pole `znak`, dlatego procedura została podzielona na dwie części: pierwsza część dotyczy spacji, natomiast część druga – każdego, innego znaku. Jeśli więc pierwszy element funkcji potęgującej ma wpisany znak spacji i jest to jedyny znak tej funkcji, o czym świadczy obecność adresu struktury zamykającej w polu `prawa` tego elementu, musi on zostać usunięty z listy, ale przedtem, struktura otwierająca i zamykająca są ze sobą związane zarówno wiązaniem głównym, jak i kursorowym. Jeśli nie

jest to jedyny element funkcji potęgującej, wszystkie jej znaki zostają powiększone do pierwotnego rozmiaru. Powiększanie znaków odbywa się w procedurze `Skalowanie`, która w parametrach otrzymuje adres elementu znaku potęgowania oraz wartość `false` nakazującą procedurze powiększanie. Po powrocie z procedury, w zmiennej `ap1` wyliczana zostaje różnica między współrzędnymi poziomu `py` elementu znaku potęgowania i pierwszego elementu funkcji potęgującej. Jeśli wartość bezwzględna różnicy jest większa od jedności (niewielki błąd musi być dopuszczony z uwagi na błędy zaokrągleń po przeskalowaniu znaków), pola współrzędnych pionowych elementów funkcji potęgującej są poprawiane o tę różnicę. Obecność znaku spacji daje pewność, że pierwszy element funkcji potęgującej jest jednocześnie strukturą otwierającą znaku akcji będącego częścią tej funkcji, dlatego rolę struktury otwierającej dla tego znaku akcji przejmie dotychczasowa struktura otwierająca usuwanego elementu znaku potęgowania, natomiast element z wpisanym znakiem spacji zostaje usunięty z listy. Gdy ostatni element funkcji potęgującej ma w polu `znak` spację, postępowanie jest analogiczne do poprzedniego, z tą różnicą, że rolę struktury zamykającej znaku akcji zawartej w elementach funkcji potęgującej przejmuje struktura zamykająca należąca do usuwanego elementu znaku potęgowania, zaś element z wpisanym znakiem spacji zostaje usunięty z listy.

```
procedure us_pot(const podn5:lisc);
var up1,up2,up3:lisc;
    ap1:integer;
begin
     up1:=podn5^.prawa; - do zmiennej up1 wpisz adres pierwszego elementu...
                          funkcji potęgującej
     up2:=podn5^.kur_p; - do zmiennej up2 wpisz adres struktury zamykającej
     up3:=up2^.lewa; - do zmiennej up3 wpisz adres ostatniego elementu...
                       funkcji potęgującej
     if up1^.znak=' ' then - czy pierwszy element funkcji potęgującej...
                            ma wpisaną w polu znak spację?
        if up1=up3 then - jeśli tak, to czy adresy: pierwszego i ostatniego...
                          elementu funkcji potęgującej są takie same?
        begin - jeśli tak, to jest to jedyny element funkcji potęgującej, czyli...
             up3:=up1^.kur_g; - do zmiennej up3 wpisz adres zawarty...
                                w polu kur_g elementu funkcji potęgującej
             if up3<>nil then - czy zmienna up3 nie ma adresu pustego?
                if up3^.kur_d=up1 then up3^.kur_d:=nil; - jeśli ...
                                       nie, to gdy w polu kur_d elementu wskazywanego przez...
                                       ten wskaźnik znajduje się adres elementu funkcji...
                                       potęgującej, wpisz w to pole identyfikator adresu pustego
             up3:=podn5^.kur_l; - do zmiennej up3 wpisz adres struktury...
                                  otwierającej
             up3^.prawa:=up2;  } związ wzajemnie wiązaniem głównym...
             up2^.lewa:=up3;   } struktury: otwierającą i zamykającą
             up3^.kur_p:=up2;  } związ wzajemnie wiązaniem kursorowym...
             up2^.kur_l:=up3;  } struktury: otwierającą i zamykającą
             up3^.kur_g:=nil;  } usuń związania tych struktur polami kur_g..
             up2^.kur_g:=nil;  } z elementem funkcji potęgującej
             dispose(up1) - usuń z pamięci element funkcji potęgującej,...
                            wskazywanej przez wskaźnik up1
        end else - jeśli nie, to nie jest to jedyny element funkcji potęgującej
            begin
                 Form2.Skalowanie(podn5,false); - przywróć...
```

```
rozmiary znakom funkcji potęgującej sprzed zmniejszenia
ap1:=podn5^.py-up1^.py; - w zmiennej ap1 wylicz...
                          różnicę między współrzędnymi poziomu py...
                          elementu znaku potęgowania i pierwszego...
                          elementu funkcji potęgującej
if abs(ap1)>1 then - czy wartość bezwzględna...
                     wyliczonej różnicy jest większa od jedności?
begin - jeśli tak, to...
     up3:=up1; - zachowaj w zmiennej up3 adres...
                 pierwszego elementu funkcji potęgującej
     repeat
           with up3^ do
           begin
                ly:=ly+ap1; } zmodyfikuj pola...
                py:=py+ap1; } współrzędnych...
                y1:=y1+ap1; } pionowych elementów..
                y2:=y2+ap1  } funkcji potęgującej
           end;
           up3:=up3^.prawa -we wskaźniku up3...
           pozyskaj adres następnika w wiązaniu głównym
     until up3=up2; - opuść pętlę, gdy wskaźnik up3...
                      uzyska adres struktury zamykającej,...
                      zawartej we wskaźniku up2
end;
up3:=up1^.kur_g; - do zmiennej up3 wpisz adres zawarty...
    w polu kur_g pierwszego elementu funkcji potęgującej
if up3<>nil then - czy zmienna up3...
                   nie ma adresu pustego?
   if up3^.kur_d=up1 then up3^.kur_d:=nil;
   jeśli nie, to gdy w polu kur_d elementu wskazywanego przez...
   ten wskaźnik znajduje się adres pierwszego elementu funkcji...
   potęgującej, wpisz w to pole identyfikator adresu pustego
up3:=up1^.prawa; - do zmiennej up3 wpisz adres...
  następnika w wiązaniu głównym pierwszego elementu...
  funkcji potęgującej
up2:=podn5^.lewa; - do zmiennej up2 wpisz adres...
                    struktury otwierającej
up2^.prawa:=up3; } związ wzajemnie wiązaniem głównym...
up3^.lewa:=up2;  } elementy wskazane przez wskaźniki
up3:=up1^.kur_p; - do zmiennej up3 wpisz adres...
                   następnika w wiązaniu kursorowym...
                   pierwszego elementu funkcji potęgującej
up3^.kur_l:=up2; } wskazane przez wskaźniki elementy
up2^.kur_p:=up3; } związ wiązaniem kursorowym
dispose(up1); - usuń z pamięci pierwszy element...
                funkcji potęgującej
```

```
                    up2:=podn5^.kur_p; - do zmiennej up2 wpisz adres...
                                         struktury zamykającej
                    up1:=up2^.lewa; - do zmiennej up1 wpisz adres...
                                      ostatniego elementu funkcji potęgującej
                    if up1^.znak=' ' then - czy element ten ma w polu...
                                            znak spację?
                    begin - jeśli tak, to...
                         up3:=up1^.kur_g; - do zmiennej up3 wpisz adres...
                                            zawarty w polu kur_g ostatniego elementu...
                                            funkcji potęgującej
                         if up3<>nil then - czy zmienna up3 nie ma...
                                            adresu pustego?
                            if up3^.kur_d=up1 then
                               up3^.kur_d:=nil; - jeśli nie, to w pole...
                                 kur_d wpisz identyfikator adresu pustego,...
                                 zgodnie z warunkami opisanymi powyżej
                         up3:=up1^.lewa;   } związ wzajemnie wiązaniem...
                         up3^.prawa:=up2;  } głównym i kursorowym...
                         up2^.lewa:=up3;   } poprzednika i następnika...
                         up3:=up1^.kur_l;  } ostatniego elementu funkcji...
                         up3^.kur_p:=up2;  } potęgującej
                         up2^.kur_l:=up3;  }
                         dispose(up1) - usuń z pamięci ostatni element...
                                        funkcji potęgującej
                    end;
               end;
          else
```

Jeśli pierwszy element funkcji potęgującej posiada w polu `znak` każdy inny znak prócz spacji, wszystkim znakom funkcji potęgującej zostaje przywrócony ich pierwotny rozmiar sprzed zmniejszenia. W tym celu wywołana zostaje procedura `Skalowanie`, która otrzymuje w parametrach: adres elementu znaku potęgowania oraz wartość `false` nakazująca procedurze powiększanie. Po powrocie z procedury, w zmiennej `ap1` wyliczana jest różnica między wartościami w polach współrzędnych poziomu `py` elementu znaku potęgowania i pierwszego elementu funkcji potęgującej. Wartość bezwzględna tej różnicy większa od jeden uruchamia pętlę, w której pola współrzędnych wszystkich elementów funkcji potęgującej otrzymują nowe wartości, które będą sumą dotychczasowej wartości współrzędnej poprawianego elementu i wyliczonej różnicy. Po tak zmodyfikowanych współrzędnych pionowych, funkcja potęgująca zostanie przesunięta do poziomu ukrytego znaku potęgowania.

Często zdarza się, że funkcja potęgująca zamknięta jest w parze nawiasów. Jeśli nawiasy te są ukryte, muszą one zostać odkryte, ponieważ pozostawienie ich w stanie ukrytym spowoduje trwałe ich ukrycie oraz uniemożliwi poruszanie się kursorem w miejscu, w którym znajduje się ukryty nawias. Odkrycie ukrytego nawiasu polega na ustawieniu pola `druk` w pozycji `true`, należącego do składnika listy, który w polu `znak` ma wpisany nawias. Ukryty nawias nie posiada szerokości, zatem po odkryciu nawiasu, koniecznie trzeba wyliczyć mu szerokość i zmodyfikować współrzędną poziomą `px` należącą do tego składnika. W tym celu wywołana zostaje procedura `Szerokosc`, której w parametrze podany zostaje adres składnika listy z wpisanym nawiasem.

Jeśli ostatni element funkcji potęgującej ma w polu `znak` wpisaną spację, jest on zapewne strukturą zamykającą znaku akcji, będącego częścią funkcji potęgującej. W takiej sytuacji element ten zostaje usunięty z listy a rolę struktury zamykającej dla tego znaku akcji

przejmie dotychczasowa struktura zamykająca, należąca do usuwanego znaku potęgowania. Przed usunięciem składnika z listy, ustalone zostały adresy jego poprzedników tak w wiązaniu głównym, jak i kursorowym. Na podstawie tych adresów, wskazane nimi elementy zostały związane ze strukturą zamykającą należącą do usuwanego elementu znaku potęgowania.

Struktura zamykająca może zawierać w polu `znak` spację. Jeśli nie jest ona strukturą zamykającą innego znaku akcji, może zostać z listy usunięta. W tym celu sprawdzane jest istnienie jej następników w wiązaniu głównym oraz kursorowym. Ich obecność pozwala na wzajemne związanie z poprzednikiem usuwanej struktury zamykającej zarówno wiązaniem głównym, jak i kursorowym. Ponieważ usuwany element nie należy do otoczenia znaku akcji, jego poprzednik w wiązaniu głównym i kursorowym jest tym samym elementem.

```
up1:=podn5^.prawa; - do zmiennej up1 wpisz adres pierwszego elementu...
                     funkcji potęgującej
up2:=podn5^.kur_p; - do zmiennej up2 wpisz adres struktury zamykającej
up3:=up2^.lewa; - do zmiennej up3 wpisz adres ostatniego elementu...
                  funkcji potęgującej
if up1^.znak=' ' then - czy pierwszy element funkcji potęgującej...
                        ma wpisaną w polu znak spację?
   . . .
else - pierwszy element funkcji potęgującej nie ma wpisanej spacji, więc...
begin
     Form2.Skalowanie(podn5,false);
     ap1:=podn5^.py-up1^.py;
     if abs(ap1)>1 then
     begin
          up3:=up1;
          repeat
                with up3^ do
                begin
                     ly:=ly+ap1;
                     py:=py+ap1;
                     y1:=y1+ap1;
                     y2:=y2+ap1
                end;
                up3:=up3^.prawa
          until up3=up2;
     end;
     if not up1^.druk then - czy pole druk pierwszego elementu funkcji...
                             potęgującej jest ustawione w pozycji false?
     begin - jeśli tak, znak jest ukryty, więc...
          up1^.druk:=true; - ustaw pole druk w pozycji true
          Form2.Szerokosc(up1); - przywróć szerokość znakowi
          Form2.Przesun(up1,true) - przesuń pozostałe znaki w prawo...
                                    o przywróconą szerokość
     end;
     up3:=podn5^.lewa; - do zmiennej up3 wpisz adres struktury otwierającej
     up3^.prawa:=up1;  } związ wzajemnie wiązaniem głównym pierwszy...
     up1^.lewa:=up3;   } element funkcji potęgującej i strukturę otwierającą
     up3^.kur_p:=up1;  } związ wzajemnie wiązaniem kursorowym pierwszy...
     up1^.kur_l:=up3;  } element funkcji potęgującej i strukturę otwierającą
     if up1^.kur_d=up3 then up1^.kur_d:=nil; - usuń z pola kur_d,...
             pierwszego elementu funkcji potęgującej, adres struktury otwierającej
     up3^.kur_g:=nil; - usuń z pola kur_g adres elementu funkcji potęgującej
     up3:=up2^.lewa; - do zmiennej up3 wpisz adres ostatniego elementu...
                       funkcji potęgującej
```

```
if not up3^.druk then - czy pole druk tego elementu jest ustawione...
                        w pozycji true?
begin - jeśli tak, to znak jest ukryty, czyli...
     up3^.druk:=true; - ustaw pole druk w pozycji true
     Form2.Szerokosc(up3) - przywróć szerokość znakowi
end;
if up3^.znak=' ' then - czy ostatni element funkcji potęgującej...
                        ma w polu znak spację?
begin - jeśli tak, to...
     up2^.kur_g:=nil; - usuń z pola kur_g struktury zamykającej...
                        adres elementu funkcji potęgującej
     up1:=up3^.lewa; - do zmiennej up1 wczytaj adres poprzednika...
           w wiązaniu głównym ostatniego elementu funkcji potęgującej
     if up1<>nil then up1^.prawa:=up2; - jeśli poprzednik...
          istnieje, wpisz w jego pole prawa adres struktury zamykającej
     up2^.lewa:=up1; - w pole lewa struktury zamykającej wpisz adres...
                       tego poprzednika
     up1:=up3^.kur_l; - do zmiennej up1 wczytaj adres poprzednika...
           w wiązaniu kursorowym ostatniego elementu funkcji potęgującej
     if up1<>nil then up1^.kur_p:=up2; - jeśli poprzednik...
          istnieje, wpisz w jego pole kur_p adres struktury zamykającej
     up2^.kur_l:=up1; - w pole kur_l struktury zamykającej wpisz...
                        adres tego poprzednika
     dispose(up3); - usuń z pamięci ostatni element funkcji potęgującej
     up3:=up1 - teraz ostatnim elementem funkcji potęgującej jest...
                jego poprzednik – wpisz jego adres do zmiennej up3
end else - jeśli nie, ostatni element funkcji potęgującej nie ma...
           w polu znak spacji, więc...
    begin
         if up3^.kur_d=up2 then up3^.kur_d:=nil; - usuń...
                        adres struktury zamykającej z pola kur_d...
                        ostatniego elementu funkcji potęgującej
         up2^.kur_g:=nil; - usuń adres ostatniego elementu...
            funkcji potęgującej z pola kur_g struktury zamykającej
         up2^.kur_l:=up3; } związ wzajemnie wiązaniem kursorowym..
         up3^.kur_p:=up2 } ostatni element funkcji potęgującej...
                           i strukturę zamykającą
    end;
up1:=up2^.kur_l; - do zmiennej up1 wpisz adres poprzednika...
                   w wiązaniu kursorowym struktury zamykającej
if up2^.znak=' ' then - czy w polu znak struktury zamykającej...
                        znajduje się spacja?
   if ord(up1^.znak) in dzed then - jeśli tak, to czy w polu...
                znak elementu wskazywanego przez wskaźnik up1...
                znajduje się znak akcji?
   else - jeśli nie, to...
   begin
        up1:=up2^.prawa; - do zmiennej up1 wpisz adres następnika...
                           w wiązaniu głównym struktury zamykającej
        if up1<>nil then up1^.lewa:=up3; - jeżeli następnik...
                           istnieje, wpisz w jego pole lewa adres...
```

```
                                        ostatniego elementu funkcji potęgującej
                up3^.prawa:=up1; - w pole prawa ostatniego elementu...
                                   funkcji potęgującej wpisz adres następnika...
                                   zawartego we wskaźniku up1
                up1:=up2^.kur_p; - do zmiennej up1 wpisz adres następnika...
                                   w wiązaniu kursorowym struktury zamykającej
                if up1<>nil then up1^.kur_l:=up3; - jeżeli następnik...
                                   ten istnieje, wpisz w jego pole kur_l adres...
                                   ostatniego elementu funkcji potęgującej
                up3^.kur_p:=up1; - w pole kur_p ostatniego elementu...
                                   funkcji potęgującej wpisz adres następnika...
                                   zawartego we wskaźniku up1
                dispose(up2) - usuń z pamięci element wskazywany przez...
                               wskaźnik up2 (struktura zamykająca)
           end
      else Form2.Przesun(up3,true) jeśli w polu znak struktury...
           zamykającej nie ma spacji, usuń odstęp między tym znakiem...
           a ostatnim znakiem funkcji potęgującej
end;
```

XIV. 3.4. Usunięcie funkcji 'log' – procedura `us_log`

Zadaniem tej procedury jest usunięcie z listy trzech kolejnych składników, które razem tworzą nazwę funkcji 'log'. Jeśli funkcja ta posiada podstawę, jej znakom przywrócony zostaje pierwotny rozmiar sprzed zmniejszenia. Ponadto procedura musi wyłonić spośród pozostałych składników element bieżący, przy którym znajdzie się kursor. Na początku ustalany jest adres pierwszego elementu z wpisanym w polu `znak` literą 'l' oraz adres ostatniego elementu zawierającego w polu `znak` literę 'g'. Odbywa się to poprzez identyfikację znaku zawartego w elemencie, którego adres został procedurze dostarczony za pośrednictwem parametru o nazwie `odn7`. Rozpoznając w instrukcji wyboru `case` jedną z trzech liter nazwy funkcji, można wskazać i zapamiętać w lokalnych wskaźnikach `lo1` i `lo2` adresy: pierwszego i ostatniego elementu rdzenia tej funkcji. Nie trzeba sprawdzać, czy elementy wskazywane przez te wskaźniki zawierają w polach `znak` wszystkie litery nazwy funkcji i w pożądanej kolejności, gdyż zostało to już wcześniej sprawdzone przez funkcję `czy_log`.

```
ua7:=ord(odn7^.znak); - w zmiennej ua7 zapamiętaj wartość znaku zawartego...
                        w polu znak elementu wskazywanego przez parametr odn7
case ua7 of
     108: - rozpoznano wartość litery 'l'
     begin
          lo1:=odn7; - zapamiętaj w zmiennej lo1 adres zawarty w parametrze
          lo2:=odn7^.kur_p; - do zmiennej lo2 wpisz adres następnika...
                              elementu wskazywanego przez parametr odn7
          lo2:=lo2^.kur_p - w zmiennej lo2 pozyskaj adres następnika
     end;
     111: - rozpoznano wartość litery 'o'
     begin
          lo1:=odn7^.kur_l; - do zmiennej lo1 wpisz adres poprzednika...
                              elementu wskazywanego przez parametr odn7
          lo2:=odn7^.kur_p - do zmiennej lo2 wpisz adres następnika...
                             elementu wskazywanego przez parametr odn7
```

```
        end;
        103: - rozpoznano wartość litery 'g'
        begin
             lo1:=odn7^.kur_l; - do zmiennej lo1 wpisz adres poprzednika...
                                 elementu wskazywanego przez parametr odn7
             lo1:=lo1^.kur_l; - w zmiennej lo1 pozyskaj adres poprzednika
             lo2:=odn7 - do zmiennej lo2 wpisz adres zawarty w parametrze
        end;
        else - nie rozpoznano żadnej z wymienionych liter
        begin
             lo1:=nil; } zmiennym lo1 i lo2 wpisz...
             lo2:=nil  } identyfikatory adresów pustych
        end;
end;
```

Niezależnie od tego, czy w polu `znak` elementu podstawy logarytmu, znajduje się spacja czy inny znak, wywołana zostaje procedura `Skalowanie`, która przywraca rozmiar wszystkim znakom, które należą do podstawy logarytmu. Po powrocie z procedury, pierwszy element tej podstawy jest wiązany z ewentualnym poprzednikiem rdzenia funkcji zarówno wiązaniem głównym, jak i kursorowym. Jeśli poprzednik w wiązaniu głównym nie istnieje, pierwszym elementem listy będzie wówczas pierwszy element podstawy logarytmu – jego adres zostaje przekazany głównemu wskaźnikowi edytora `wsp`. Nim elementy rdzenia funkcji zostaną usunięte, w zmiennych liczbowych `ua7` i `ub7` zapamiętane zostają wartości współrzędnych z pól `lx` i `ly` pierwszego elementu rdzenia. Na ich podstawie będzie można zmodyfikować współrzędne pozostałym elementom tak, by zajęły miejsce usuwanego rdzenia funkcji. Teraz już można zająć się usuwaniem z listy wspomnianych elementów. Uruchamiana jest zatem pętla, w której prócz usunięcia wskazanego elementu z listy, usuwany jest także adres każdego z elementów rdzenia funkcji, jaki mógł zostać zapamiętany w polach `kur_g` lub `kur_d` elementów usytuowanych nad lub pod logarytmem.

```
if (lo1=nil) or (lo2=nil) then exit; - jeśli wskaźniki lo1 lub lo2 mają...
                                       adresy puste, opuść procedurę

Form2.Skalowanie(lo2,false); - przywróć znakom podstawy logarytmu...
                               pierwotny rozmiar sprzed zmniejszenia

lo3:=lo2^.prawa; - do zmiennej lo3 wpisz adres pierwszego elementu...
                   podstawy logarytmu

lo2:=lo2^.kur_p; - do zmiennej lo2 wpisz adres struktury zamykającej
lo4:=lo1^.lewa; - do zmiennej lo4 wczytaj adres poprzednika w wiązaniu...
                  głównym pierwszego elementu rdzenia funkcji

lo3^.lewa:=lo4; - wpisz w pole lewa pierwszego elementu podstawy logarytmu...
                  adres poprzednika

if lo4<>nil then lo4^.prawa:=lo3 else wsp:=lo3; - jeśli...
                  poprzednik istnieje, wpisz w jego pole prawa adres...
                  pierwszego elementu podstawy logarytmu, jeśli...
                  poprzednik nie istnieje, wpisz wskaźnikowi edytora...
                  wsp adres pierwszego elementu podstawy logarytmu

lo4:=lo1^.kur_l; - do zmiennej lo4 wpisz adres poprzednika w wiązaniu...
                   kursorowym pierwszego elementu rdzenia funkcji

lo3^.kur_l:=lo4; - wpisz w pole kur_l pierwszego elementu...
                   podstawy logarytmu adres poprzednika

if lo4<>nil then lo4^.kur_p:=lo3; - jeśli poprzednik istnieje,...
 wpisz w jego pole kur_p adres pierwszego elementu podstawy logarytmu
```

```
ua7:=lo1^.lx; - zapamiętaj w zmiennej ua7 wartość współrzędnej poziomej...
                z pola lx pierwszego elementu rdzenia funkcji
ub7:=lo1^.ly; - zapamiętaj w zmiennej ub7 wartość współrzędnej pionowej...
                z pola ly pierwszego elementu rdzenia funkcji
lo4:=lo1; - zachowaj w zmiennej lo4 adres pierwszego elementu rdzenia funkcji
repeat
     lo1:=lo4^.kur_g; - do zmiennej lo1 wpisz adres elementu...
                        znajdującego się nad logarytmem
     if lo1<>nil then - czy element nad logarytmem istnieje?
        if lo1^.kur_d=lo4 then lo1^.kur_d:=nil; - jeśli tak, to ...
                        usuń z jego pola kur_d adres elementu rdzenia funkcji
     lo1:=lo4^.kur_d; - do zmiennej lo1 wpisz adres elementu...
                        znajdującego się pod logarytmem
     if lo1<>nil then - czy element pod logarytmem istnieje?
        if lo1^.kur_g=lo4 then lo1^.kur_g:=nil; jeśli tak, to...
                        usuń z jego pola kur_g adres elementu rdzenia funkcji
     lo1:=lo4; - zapamiętaj adres elementu rdzenia funkcji w zmiennej lo1
     lo4:=lo4^.prawa; - w zmiennej lo4 pozyskaj adres następnika
     dispose(lo1) - usuń z pamięci element wskazywany przez wskaźnik lo1
until lo4=lo3; - opuść pętlę, gdy wskaźnik lo4 uzyska adres...
                 pierwszego elementu podstawy logarytmu
```

Wykrycie w instrukcji warunkowej znaku spacji zawartej w polu `znak` pierwszego elementu podstawy logarytmu, daje pewność, że jest to jedyny element tej podstawy, dlatego można usunąć go z listy. Nim to nastąpi, usunięty zostaje adres wykluczonego składnika, który może znajdować się w polach `kur_g` oraz `kur_d` innych składników listy. Struktura zamykająca zostaje związana z istniejącymi poprzednikami elementu podstawy logarytmu w wiązaniu głównym i kursorowym. Brak poprzednika w wiązaniu głównym wskazuje, że pierwszym składnikiem listy będzie struktura zamykająca, dlatego jej adres musi zostać przekazany wskaźnikowi edytora `wsp`. Po uwolnieniu pamięci zajmowanej przez element podstawy logarytmu, wskaźnik `lo3` przyjmuje adres struktury zamykającej, zaznaczając w ten sposób brak podstawy logarytmu. Będzie to wyraźna wskazówka dla instrukcji ustalających element bieżący.

Jeżeli w polu `znak` pierwszego elementu podstawy logarytmu znajduje się inny znak niż spacja, można przyjąć, że podstawa ta jest liczbowa, dlatego należy ją zachować. Na podstawie wcześniej zapamiętanych wartości współrzędnych rdzenia funkcji, wyliczana jest różnica między nimi a wartościami współrzędnych, należących do pierwszego elementu podstawy logarytmu. Na podstawie wyliczonych wartości, w pętli `repeat..until` wyliczane są nowe wartości współrzędnych, które wpisane zostają w pola współrzędnych elementów podstawy. Ponadto usuwany jest adres struktury zamykającej zawarty w polach `kur_g` poprawianych elementów.

```
if lo3^.znak=' ' then - czy pierwszy element podstawy logarytmu zawiera spację?
begin - jeśli tak, to...
     lo1:=lo3^.kur_g; - wpisz do zmiennej lo1 adres zawarty w polu kur_g
     if lo1<>nil then
        if lo1^.kur_d=lo3 then lo1^.kur_d:=nil;
        jeśli element istnieje i zawiera w swym polu kur_d...
        adres elementu podstawy logarytmu, wpisz w to pole...
        identyfikator adresu pustego
```

```
      lo1:=lo3^.kur_d; - wpisz do zmiennej lo1 adres zawarty w polu kur_d
      if lo1<>nil then
         if lo1^.kur_g=lo3 then lo1^.kur_g:=nil;
         jeśli element istnieje i zawiera w swym polu kur_g...
         adres elementu podstawy logarytmu, wpisz w to pole...
         identyfikator adresu pustego

      lo1:=lo3^.lewa; - wpisz do zmiennej lo1 adres poprzednika...
                        w wiązaniu głównym elementu podstawy

      lo2^.lewa:=lo1; - wpisz w pole lewa struktury zamykającej...
                        adres poprzednika

      if lo1<>nil then lo1^.prawa:=lo2 else wsp:=lo2; - jeżeli...
                       poprzednik istnieje, wpisz w jego pole prawa adres...
                       struktury zamykającej, w przeciwnym przypadku, wpisz...
                       wskaźnikowi edytora wsp adres struktury zamykającej

      lo1:=lo3^.kur_l; - wpisz zmiennej lo1 adres poprzednika w wiązaniu...
                         kursorowym elementu podstawy

      lo2^.kur_l:=lo1; - w pole kur_l struktury zamykającej...
                         wpisz adres poprzednika

      if lo1<>nil then lo1^.kur_p:=lo2; - jeśli poprzednik istnieje,...
                       wpisz w jego pole kur_p adres struktury zamykającej

      dispose(lo3); - usuń z pamięci element podstawy logarytmu...
                      wskazywany przez wskaźnik lo3

      lo3:=lo2 - wpisz zmiennej lo3 adres struktury zamykającej
end else - jeśli pierwszy element podstawy logarytmu nie zawiera spacji, to...
    begin
         ua7:=lo3^.lx-ua7; - w zmiennej ua7 wylicz różnicę między...
                             wartością współrzędnej poziomej lx...
                             pierwszego elementu podstawy a wartością...
                             współrzędnej lx rdzenia funkcji

         ub7:=lo3^.ly-ub7; - w zmiennej ub7 wylicz różnicę między...
                             wartością współrzędnej pionowej ly...
                             pierwszego elementu podstawy a wartością...
                             współrzędnej ly rdzenia funkcji

         lo1:=lo3; - zachowaj w zmiennej lo1 adres pierwszego elementu...
                     podstawy logarytmu

         repeat
               with lo1^ do
               begin
                    lx:=lx-ua7;  } wylicz nowe wartości ...
                    px:=px-ua7;  } współrzędnych poziomych...
                    ly:=ly-ub7;  } i pionowych elementów...
                    py:=py-ub7;  } podstawy logarytmu..
                    y1:=y1-ub7;
                    y2:=y2-ub7
               end;
               lo1^.kur_g:=nil; - wpisz do pola kur_g elementu...
                                  podstawy identyfikator adresu pustego

               lo3:=lo1; - zachowaj w zmiennej lo3 adres elementu podstawy,..
                           by po opuszczeniu pętli, w zmiennej tej...
                           znajdował się adres ostatniego elementu podstawy

               lo1:=lo1^.prawa - w zmiennej lo1 pozyskaj adres...
                                 następnika w wiązaniu głównym
```

```
            until lo1=lo2; - opuść pętlę, gdy wskaźnik lo1 uzyska adres...
                                struktury zamykającej
        end;
```

Gdy struktura zamykająca, będąca jednocześnie pierwszym elementem argumentu funkcji, będzie zawierała w polu `znak` spację, może zostać usunięta z listy, ale pod warunkiem, że istnieje przynajmniej jeden element podstawy logarytmu. Zachowanie przynajmniej jednego elementu po usuwanym logarytmie pozwala na łatwiejsze wskazanie elementu bieżącego. Jeśli więc warunki pozwalają na usunięcie struktury zamykającej, elementy dowiązane do tej struktury z prawej strony wiązaniem głównym oraz kursorowym, są wzajemnie wiązane z ostatnim elementem podstawy logarytmu. Ewentualne związanie tej struktury z elementem dowiązanym polem `kur_g` zostaje przeniesione do ostatniego elementu podstawy logarytmu, natomiast w pole `kur_d` wstawiony zostaje identyfikator adresu pustego.

Jeśli warunki nie pozwalają na usunięcie struktury zamykającej, wyróżnione są dwie możliwości:

- jeśli ostatni element podstawy logarytmu i struktura zamykająca są różnymi elementami listy, obydwa te elementy są ze sobą łączone wiązaniem kursorowym. Zmienna logiczna edytora `stat_kur` ustawiona jest w pozycji `true`, co spowoduje ustawienie kursora za ostatnim znakiem podstawy logarytmu;
- jeśli ostatni element podstawy logarytmu i struktura zamykająca, to ten sam element listy, oznacza to, że nie istnieje podstawa logarytmu, dlatego jedyna pozostałość po logarytmie, czyli argument funkcji zostaje przesunięty w miejsce w którym znajdował się rdzeń funkcji. Jest to możliwe dzięki zachowanej w zmiennej `ua7` współrzędnej poziomej początku rdzenia. Zmienna `stat_kur` jest w tej sytuacji ustawiona w pozycji `false`, ustawiając kursor przed pierwszym znakiem argumentu funkcji – struktury zamykającej logarytm.

Ostatnią instrukcją procedury jest przypisanie wskaźnikowi edytora `wsb` adresu zawartego w zmiennej `lo3`. Zmienna ta pamięta adres ostatniego elementu podstawy logarytmu lub, gdy podstawa ta nie istnieje, adres struktury zamykającej.

```
if (lo2^.znak=' ') and (lo3<>lo2) then - czy w pole znak...
                        struktury zamykającej wpisana jest spacja oraz...
                        czy ostatni element podstawy logarytmu i struktura...
                        zamykająca są różnymi elementami listy?
begin - jeśli obydwa warunki zostały spełnione, oznacza to, że istnieje podstawa
     lo1:=lo2^.kur_g; - wpisz do zmiennej lo1 adres zawarty w polu...
                          kur_g struktury zamykającej
     if lo1<>nil then - czy zmienna lo1 nie ma adresu pustego?
        if lo1^.kur_d=lo2 then lo1^.kur_d:=lo3; - jeśli nie, to...
                gdy w polu kur_d elementu, na który ten wskaźnik pokazuje,...
                widnieje adres struktury zamykającej, wpisz w to pole adres...
                ostatniego elementu podstawy logarytmu
     lo3^.kur_g:=lo1; - w pole kur_g ostatniego elementu podstawy...
            logarytmu wpisz adres elementu zawartego we wskaźniku lo1
     lo4:=lo2^.prawa; - do zmiennej lo4 wpisz adres następnika...
                        w wiązaniu głównym struktury zamykającej
     lo3^.prawa:=lo4; - wpisz w pole prawa ostatniego elementu...
                          podstawy adres tego następnika
     if lo4<>nil then lo4^.lewa:=lo3; - jeżeli następnik istnieje,...
                                wpisz w jego pole lewa adres ostatniego...
                                elementu podstawy logarytmu
```

```
      lo4:=lo2^.kur_p; - do zmiennej lo4 wpisz adres następnika...
                         w wiązaniu kursorowym struktury zamykającej

      lo3^.kur_p:=lo4; - wpisz w pole kur_p ostatniego elementu...
                         podstawy logarytmu adres tego następnika

      if lo4<>nil then lo4^.kur_l:=lo3; - jeżeli następnik istnieje,...
        wpisz w jego pole kur_l adres ostatniego elementu podstawy logarytmu

      dispose(lo2); - usuń z pamięci element wskazywany przez...
                      wskaźnik lo2 (struktura zamykająca)

      stat_kur:=true - ustaw zmienną edytora stat_kur w pozycji true
end else - jeśli jeden z warunków lub obydwa nie zostały spełnione, oznacza to...
           że nie istnieje podstawa logarytmu, więc...
    begin
        lo2^.kur_d:=nil; - wpisz w pole kur_d struktury zamykającej...
                           identyfikator adresu pustego

        if lo3<>lo2 then - czy istnieje podstawa logarytmu, to znaczy,...
                           czy wskaźniki lo2 i lo3 mają różne adresy?

        begin - jeśli tak, to...
            lo3^.kur_g:=nil; - wpisz w pole kur_g ostatniego elementu...
                               podstawy logarytmu identyfikator adresu pustego

            lo3^.kur_p:=lo2; } związ wzajemnie wiązaniem kursorowym..
            lo2^.kur_l:=lo3; } ostatni element podstawy logarytmu
                              i strukturę zamykającą

            Form2.Przesun(lo3,true); - dosuń do ostatniego znaku...
                                       podstawy logarytmu pozostałe znaki

            stat_kur:=true - ustaw zmienną edytora stat_kur...
                             w pozycji true

        end else - jeśli nie istnieje podstawa logarytmu, to...
            begin
                ub7:=lo2^.px-lo2^.lx; - wylicz w zmiennej ub7...
                                        szerokość znaku zawartego w polu znak...
                                        struktury zamykającej

                lo2^.lx:=ua7; - w pole lx współrzędnej poziomej...
                                struktury zamykającej wpisz wartość współrzędnej...
                                początku rdzenia funkcji, zapamiętanej w zmiennej ua7

                lo2^.px:=lo2^.lx+ub7; - zmodyfikuj współrzędną...
                                        poziomą px struktury zamykającej dodając...
                                        do wartości z jej pola lx zapamiętaną...
                                        w zmiennej ub7 szerokość znaku

                Form2.Przesun(lo2,true); - przesuń pozostałe znaki...
                                           do nowego miejsca struktury zamykającej

                stat_kur:=false - ustaw zmienną edytora stat_kur...
                                  w pozycji false

            end;
    end;
wsb:=lo3; - wpisz wskaźnikowi edytora wsb adres zawarty we wskaźniku lo3...
            (element bieżący)
```

XIV. 4. Skalowanie znaków funkcji potęgującej lub podstawy logarytmu – procedura `Skalowanie`

Zadanie procedury polega na zmniejszeniu rozmiaru znaków wpisanych w pola `znak` elementów edytora należących do funkcji potęgującej czy podstawy logarytmu lub przywrócenie tym znakom rozmiarów sprzed zmniejszenia. O tym, jakie zadanie ma do zrealizowania procedura, podpowiada jej drugi parametr logiczny o nazwie `tryb`. Standardowo skalowanie rozpoczyna się od następnika w wiązaniu głównym elementu wskazanego w pierwszym parametrze procedury do następnika tego elementu w wiązaniu kursorowym. Może się jednak zdarzyć, że skalowanie będzie musiało się rozpocząć od innego składnika, dlatego do listy parametrów dodany został trzeci parametr wyposażony w wartość domyślną równą adresowi pustemu, za pomocą którego można ten składnik procedurze wskazać.. Wartość domyślna pozwala na pominięcie takiego parametru podczas wywoływania procedury, gdyby potrzeba wskazania pierwszego elementu podlegającego skalowaniu okazała się zbędna.

Procedura rozpoczyna swą pracę od zbadania, czy parametr z wartością domyślną posiada nadal adres pusty czy adres elementu edytora. Jeśli jest to adres pusty, wskaźnikowi lokalnemu `sk3` przypisany zostaje adres następnika w wiązaniu głównym elementu, którego adres widnieje w pierwszym parametrze, w przeciwnym przypadku wskaźnikowi temu przypisany zostaje adres otrzymany w parametrze, w którym wartość domyślna została zastąpiona adresem składnika edytora. Z kolei wskaźnik lokalny `sk2` otrzymuje adres następnika w wiązaniu kursorowym, który wyznaczy granicę skalowania. Gdy oba te wskaźniki mają różne adresy, procedura kontynuuje swoje dalsze zadanie, w przeciwnym przypadku następuje opuszczenie procedury instrukcją `exit`, gdyż sytuacja taka wskazywałaby na brak elementów do skalowania.

Pierwszym ważnym punktem skalowania jest zbudowanie prostokątnego obszaru, w którym znajdują się znaki, które należy pomniejszyć lub powiększyć. Polega to na wyłonieniu w pętli `repeat..until` minimalnej i maksymalnej współrzędnej poziomej i pionowej z pól współrzędnych tych elementów edytora, które w polach `znak` mają wpisane znaki poddane skalowaniu. Uzyskane wartości posłużą do wyliczenia nowych współrzędnych każdemu z tych elementów. Wartości startowe dla czterech zmiennych, przeznaczonych na te współrzędne pochodzą z pól współrzędnych elementu odniesienia, czyli tego, który został przekazany procedurze w pierwszym parametrze. Po opuszczeniu pętli, uzyskane wartości zostają zachowane w dwóch zmiennych typu `TPoint`, co pozwala zaoszczędzić ilość zmiennych liczbowych, gdyż zmienne tego typu składają się z dwóch pól typu `LongInt`.

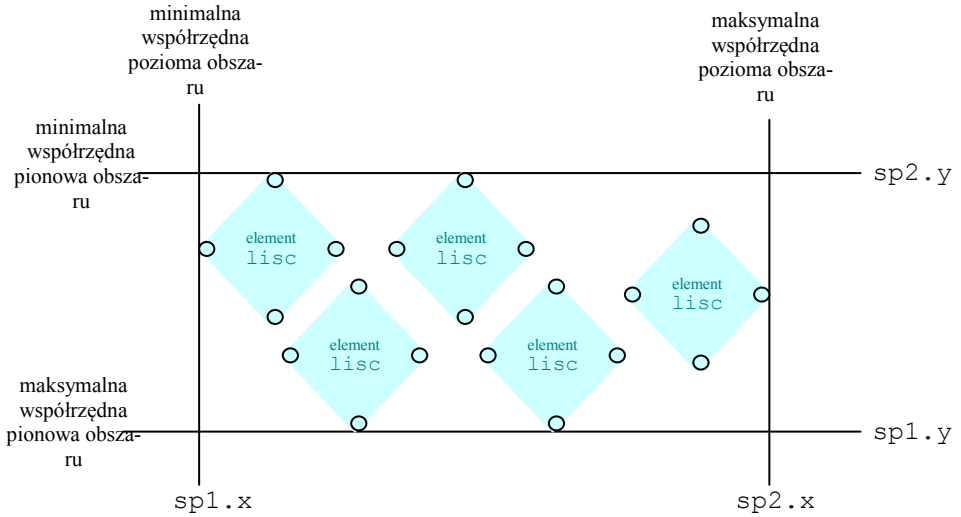
```
procedure TForm2.Skalowanie
          (const czak:lisc;tryb:boolean;const pocz:lisc=nil);
var sk1,sk2,sk3:lisc;
    sw1,sw2,sw3,sw4,swp1,swp2:integer;
    sp1,sp2:TPoint;
    czy_skalowac:boolean;
begin
     if pocz=nil then sk3:=czak^.prawa else sk3:=pocz;
             jeśli parametr pocz jest równy nil, przypisz zmiennej sk3...
             adres następnika w wiązaniu głównym elementu wskazywanego...
             przez parametr czak, w przeciwnym przypadku zmiennej tej...
             przypisz adres parametru pocz

     sk1:=sk3; - przypisz zmiennej sk1 adres zawarty w zmiennej sk3
     sk2:=czak^.kur_p; - zmiennej sk2 wpisz adres następnika elementu...
                         czak w wiązaniu kursorowym
```

```
if (sk1=nil) or (sk2=nil) then exit - jeśli jeden ze...
                wskaźników posiada adres pusty, opuść procedurę
else if (sk1=sk2) then exit; - jeśli oba wskaźniki...
              posiadające adresy następników elementu czak:...
              w wiązaniu głównym i kursorowym, są takie same,...
              opuść procedurę
sw1:=sk1^.lx; - wartość startowa lewej współrzędnej poziomej
sw2:=sk1^.py; - wartość startowa dolnej współrzędnej pionowej poziomu
sw3:=sk1^.px; - wartość startowa prawej współrzędnej poziomej
sw4:=sk1^.ly; - wartość startowa górnej współrzędnej pionowej poziomu
swp1:=sk1^.py- sk1^.ly+1; - w zmiennej swp1 wylicz...
           wysokość pierwszego znaku poddanego skalowaniu ...
           (dodanie jedynki koryguje błędy zaokrągleń)
repeat
      if sk1^.lx<sw1 then sw1:=sk1^.lx; - jeśli wartość zawarta...
            w polu lx badanego elementu jest mniejsza od wartości w zmiennej...
            sw1, wpisz do tej zmiennej wartość z tego pola
      if sk1^.py>sw2 then sw2:=sk1^.py; - jeśli wartość zawarta...
            w polu py badanego elementu jest większa od wartości w zmiennej...
            sw2, wpisz do tej zmiennej wartość z tego pola
      if sk1^.ly<sw4 then sw4:=sk1^.ly; - jeśli wartość zawarta...
            w polu ly badanego elementu jest mniejsza od wartości w zmiennej...
            sw4, wpisz do tej zmiennej wartość z tego pola
      if sk1^.px>sw3 then sw3:=sk1^.px; - jeśli wartość zawarta...
            w polu px badanego elementu jest większa od wartości w zmiennej...
            sw3, wpisz do tej zmiennej wartość z tego pola
      sk1:=sk1^.prawa - w zmiennej sk1 pozyskaj adres następnika...
                              w wiązaniu głównym
until sk1=sk2; - opuść pętlę, gdy zmienna sk1 uzyska adres...
                 zawarty w zmiennej sk2
                 (następnik elementu odniesienia w wiązaniu kursorowym)
sp1.x:=sw1; - minimalna współrzędna pozioma
sp1.y:=sw2; - maksymalna współrzędna pionowa
sp2.x:=sw3; - maksymalna współrzędna pozioma
sp2.y:=sw4; - minimalna współrzędna pionowa
```

Kolejnym krokiem procedury jest zbadanie, czy skalowanie można przeprowadzić. Zadanie to zostało podzielone na dwie części: pierwsze dla zmniejszania a drugie dla zwiększania. Jeśli procedura ma dokonać zmniejszenia znaków funkcji potęgującej lub podstawy logarytmu, o czym informuje parametr `tryb` ustawiony w pozycji `true`, ważne jest sprawdzenie, czy zmniejszenie mogło zostać wykonane podczas możliwego, wcześniejszego wywołania tej procedury. Informacji o tym dostarcza różnica wysokości znaku odniesienia i pierwszego znaku podlegającego zmniejszeniu. Jeśli wynik będzie mniejszy od dwóch, oznacza to, że zmniejszenie nie zostało jeszcze wykonane – liczba dwa pozwala pominąć możliwy błąd zaokrąglenia podczas skalowania. Jeśli więc skalowanie nie zostało wykonane, kolejnym krokiem jest sprawdzenie, czy w wyniku zmniejszenia wskazanego znaku nie zostanie przekroczona jego minimalna wysokość. Sprawdzenie to polega na porównaniu ilorazu dotychczasowej wysokości pierwszego znaku podlegającego zmniejszeniu i stopnia skalowania, czyli wartości znajdującej się w zmiennej globalnej `QR` należącej do modułu regulacji. Jeśli iloraz ten będzie większy lub równy minimalnej wysokości znaku ustawionej w kolejnej zmiennej globalnej bloku regulacji – `RM`, zmniejszanie jest dozwolone.

Dzięki zapamiętanym w zmiennych `sp1` i `sp2` współrzędnych prostokątnego obszaru skalowania, wyliczane są: wysokość i szerokość tego obszaru. Dzieląc każdą z tych wielkości przez stopień skalowania, czyli wartość w zmiennej globalnej modułu regulacji `QR`, zostają uzyskane współczynniki skalowania, osobno dla wyliczenia nowych współrzędnych poziomych i pionowych.

Jeśli parametr `tryb` został ustawiony w pozycji `false`, zadaniem procedury jest zbadanie, czy różnica wysokości znaku odniesienia i pierwszego znaku podlegającego zwiększeniu jest większa lub równa dwa, jeśli tak, przywrócenie rozmiarów sprzed zmniejszenia jest dozwolone. W tym przypadku współczynniki zwiększania zostały uzyskane w wyniku mnożenia stopnia skalowania oraz odpowiednio: wysokości obszaru skalowania dla wyliczenia nowych współrzędnych pionowych oraz szerokości obszaru skalowania dla wyliczenia nowych współrzędnych poziomych.



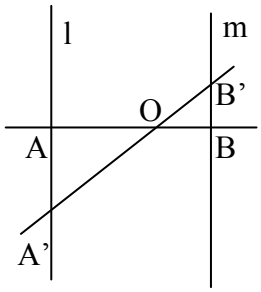
Rys. 57. Obszar skalowania oraz zawarte w nim elementy podlegające skalowaniu

```
sk1:=sk3; - wpisz do zmiennej sk1 adres pierwszego elementu lisc...
                podlegającego skalowaniu
swp2:=(czak^.py-czak^.ly)-(sk1^.py-sk1^.ly); - w zmiennej...
                      swp2 wylicz różnicę wysokości elementu odniesienia...
                      i pierwszego elementu przeznaczonego do skalowania

czy_skalowac:=false; - zmienną czy_skalowac ustaw w pozycji false
if tryb then - czy parametr tryb jest ustawiony w pozycji true?
   if swp2<2 then - jeśli tak, to czy wartość w zmiennej swp2...
                       jest mniejsza od liczby dwa?

      if round(swp1/Form6.QR)>=Form6.RM then - jeśli tak, to...
               czy iloraz wysokości pierwszego znaku przeznaczonego do...
               zmniejszenia i stopnia skalowania jest większy lub równy...
               minimalnej wysokości znaku po zmniejszeniu?

      begin - jeśli tak, to...
            sw1:=sp1.y-sp2.y; - wylicz wysokość obszaru zmniejszania
            sw2:=sp2.x-sp1.x; - wylicz szerokość obszaru zmniejszania
            swp1:=round(sw1/Form6.QR); - wylicz współczynnik...
                                zmniejszania współrzędnych pionowych

            swp2:=round(sw2/Form6.QR); - wylicz współczynnik...
                                zmniejszania współrzędnych poziomych
```

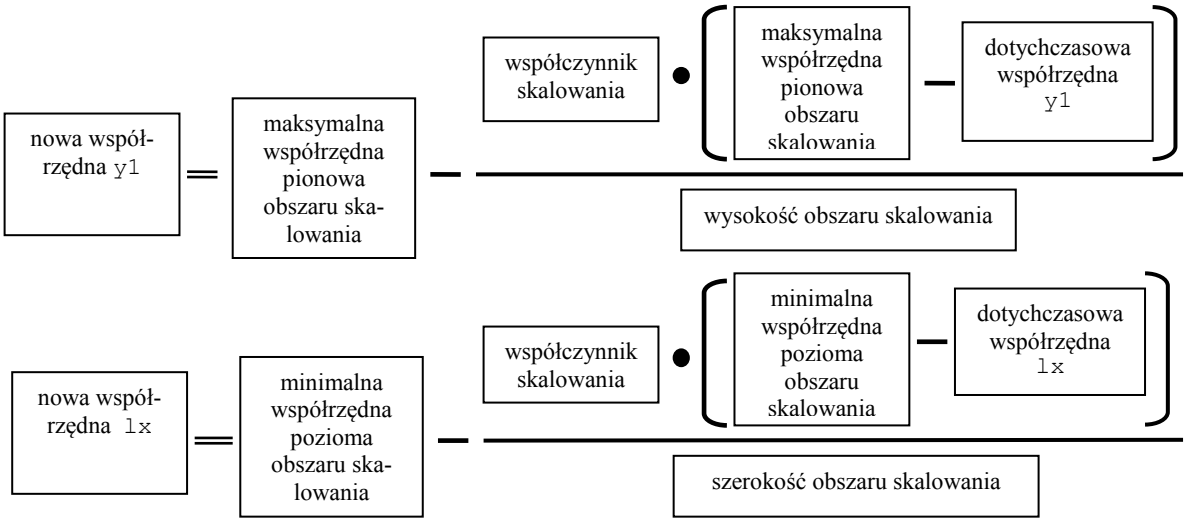
```
                czy_skalowac:=true - ustaw zmienną czy_skalowac...
                      w pozycji true, oznaczającej dozwolone zmniejszanie
          end else
      else
else - gdy parametr tryb jest ustawiony w pozycji false, to...
if swp2>2 then - czy wartość w zmiennej swp2 jest większa od liczby dwa?
begin - jeśli tak, to zwiększenie nie zostało jeszcze wykonane, czyli...
     sw1:=sp1.y-sp2.y; - wylicz wysokość obszaru zwiększania
     sw2:=sp2.x-sp1.x; - wylicz szerokość obszaru zwiększania
     swp1:=trunc(sw1*Form6.QR); - wylicz współczynnik zwiększania...
                                  współrzędnych pionowych
     swp2:=trunc(sw2*Form6.QR); - wylicz współczynnik zwiększania...
                                  współrzędnych poziomych
     czy_skalowac:=true - ustaw zmienną czy_skalowac...
           w pozycji true, oznaczającej dozwolone zwiększanie
end;
```

Jeśli zmienna logiczna `czy_skalowac` została ustawiona w pozycji `true`, uruchamiana jest pętla, w której wskazanym w parametrach elementom edytora, zostają wyliczone nowe współrzędne. Wyliczenie to opiera się na twierdzeniu Talesa o następującej treści:
Jeżeli proste l i m są równoległe, to odcinki wyznaczone przez nie na jednej prostej są proporcjonalne do odcinków wyznaczonych na drugiej prostej.

$$OA:OB=OA':OB'$$



W programie proporcjonalność obszaru skalowania i poszczególnych elementów w nim zawartych zostały wykorzystane do wyliczenia nowych współrzędnych każdemu z elementów. Poniżej zaprezentowane wzory powinny wyjaśnić sposób wyliczenia współrzędnej pionowej `y1` oraz poziomej `lx` wskazanemu w pętli elementowi.



Rys. 58. Schemat blokowy przedstawiający sposób wyliczania nowych współrzędnych elementom podlegającym skalowaniu

```
if czy_skalowac then - czy zmienna czy_skalowac została ustawiona...
                       w pozycji true?
repeat - jeśli tak, to uruchom pętlę
      with sk1^ do - instrukcja wiążąca
      begin
           if sw2<>0 then - czy zmienna sw2 (szerokość...
                            obszaru skalowania) jest różna od zera?
           begin - jeśli tak, to...
              lx:=sp1.x-trunc((swp2*(sp1.x-sk1^.lx))/sw2);
              px:=sp1.x-trunc((swp2*(sp1.x-sk1^.px))/sw2);
              wylicz i zmodyfikuj pola współrzędnych poziomych...
              elementu bieżącego w pętli
           end;
           y1:=sp1.y-trunc((swp1*(sp1.y-sk1^.y1))/sw1);
           y2:=sp1.y-trunc((swp1*(sp1.y-sk1^.y2))/sw1);
           ly:=sp1.y-trunc((swp1*(sp1.y-sk1^.ly))/sw1);
           py:=sp1.y-trunc((swp1*(sp1.y-sk1^.py))/sw1);
           wylicz i zmodyfikuj wszystkie pola współrzędnych pionowych...
           elementu bieżącego w pętli
      end;
      sk1:=sk1^.prawa - w zmiennej sk1 pozyskaj adres następnika...
                        w wiązaniu głównym
until sk1=sk2; - opuść pętlę, gdy zmienna sk1 uzyska adres następnika...
                 elementu odniesienia w wiązaniu kursorowym
```

Ostatnią czynnością procedury jest ewentualne przesunięcie w poziomie zmniejszonych lub zwiększonych znaków tak, by między znakiem odniesienia a pierwszym skalowanym znakiem nie było przerwy ani zachodzenia znaków na siebie. Uruchomienie pętli przesuwającej te znaki jest uzależnione od istnienia różnicy między wartościami w polach `px` elementu odniesienia i `lx` pierwszego skalowanego elementu.

```
sk1:=sk3; - zmiennej sk1 przypisz adres pierwszego, skalowanego elementu
sw1:=czak^.px-sk1^.lx; - wylicz różnicę między polem px elementu...
      odniesienia i polem lx elementu wskazywanego przez wskaźnik sk1
if sw1<>0 then - czy wyliczona wartość w zmiennej sw1 jest różna od zera?
repeat - jeśli tak, to uruchom pętlę
      sk1^.lx:=sk1^.lx+sw1; - skoryguj pole lx o wyliczoną różnicę
      sk1^.px:=sk1^.px+sw1; - skoryguj pole px o wyliczoną różnicę
      sk1:=sk1^.prawa - w zmiennej sk1 pozyskaj adres następnika...
                        w wiązaniu głównym
until sk1=sk2; - opuść pętlę, gdy zmienna sk1 uzyska adres następnika...
                 elementu odniesienia w wiązaniu kursorowym
```

XIV. 5. Poruszanie się po funkcji za pomocą klawiszy sterowania kursorem

Naciśnięcie jednego z klawiszy klawiatury uruchamia zdarzenie, w wyniku którego uruchomiona zostaje procedura `FormKeyDown`. Procedura ta posiada w jednym z parametrów wartość znaku odpowiadającego naciśniętemu klawiszowi. Jeśli wartość tą wywołało naciśnięcie jednego z klawiszy kierunkowych klawiatury, uruchamiane są instrukcje zmieniające pozycję kursora. Nim to jednak nastąpi, dotychczasowy ślad kursora na formie musi zostać usunięty. Polega to na pobraniu z pół współrzędnych bieżącego elementu edytora trzech war-

tości: początkowej współrzędnej poziomej i dwóch współrzędnych pionowych rzeczywistych. O tym, z którego pola współrzędnych poziomych ma zostać pobrana wartość, decyduje usytuowanie kursora z lewej lub prawej strony znaku, dlatego gdy zmienna edytora `stat_kur` ustawiona jest w pozycji `false`, wartość ta pobrana zostaje z pola `lx`, w przeciwnym przypadku z pola `px` bieżącego elementu. Współrzędne pionowe pochodzą z jego pól `y1` i `y2`, jednakże gdy w polu `znak` znajduje się znak dzielenia, współrzędne pionowe wyliczane są w ten sposób, że do osobnej zmiennej trafia ¼ wysokości pomiędzy górną a dolną współrzędną poziomu, na którym znak ten się znajduje. Wyliczenie pierwszej wartości współrzędnej pionowej polega na dodaniu do współrzędnej `ly` ¼ wysokości poziomu; natomiast drugiej, na odjęciu od współrzędnej `py` ¼ wysokości poziomu. Uzyskane w ten sposób wartości współrzędnych odpowiadają rzeczywistej pozycji kursora usytuowanego przy kresce ułamkowej. Teraz już wystarczy ustawić kolor rysowanej linii na kolor tła formy grubość linii na jeden piksel i narysować linię w miejscu wskazanym przez wyliczone współrzędne. Po narysowaniu linii, ślad kursora przestaje być widoczny.

```
if stat_kur then kx:=wsb^.px else kx:=wsb^.lx; - jeśli zmienna stat_kur..
                                    ustawiona jest w pozycji true, wpisz do zmiennej kx...
                                    wartość z pola px bieżącego elementu,...
                                    w przeciwnym przypadku, wpisz do tej zmiennej...
                                    wartość z pola lx

if ord(wsb^.znak)=47 then - czy w polu znak bieżącego elementu znajduje się...
                                  znak dzielenia?

begin - jeśli tak, to
     pom:=trunc((wsb^.y1-wsb^.ly)/2); - w zmiennej pom wylicz ¼ wysokości...
                                          poziomu, czyli połowę wysokości między...
                                          rzeczywistą współrzędną kreski ułamkowej...
                                          a współrzędną poziomu ly
     ky1:=wsb^.ly+pom; - wylicz w zmiennej ky1 górną współrzędną pionową...
                              kreski ułamkowej

     ky2:=wsb^.py-pom - wylicz w zmiennej ky2 dolną współrzędną pionową...
                             kreski ułamkowej

end else - jeśli nie, jest to każdy inny znak prócz znaku dzielenia, czyli...
    begin
         ky1:=wsb^.y1; - wpisz do zmiennej ky1 wartość górnej współrzędnej pionowej
         ky2:=wsb^.y2 - wpisz do zmiennej ky2 wartość dolnej współrzędnej pionowej
    end;
Canvas.Pen.color:=Form6.F3D; - ustaw kolor rysowanej linii na kolor tła formy...
                    roboczej, zawarty w zmiennej F3D należącej do modułu regulacji

Canvas.Pen.width:=1; - ustaw grubość rysowanej linii na jeden piksel
Canvas.moveto(kx,ky1); - ustaw początek rysowanej linii, podając w poleceniu...
                       współrzędna poziomą kx i górną współrzędną pionową ky1

Canvas.lineto(kx,ky2); - nałóż na dotychczasowy ślad kursora linię w kolorze tła formy
```

Po usunięciu śladu kursora, można przenieść kursor w inne miejsce. Do tego celu służą klawisze kierunkowe klawiatury, których wartości rozpoznawane są w instrukcjach wyboru `case`, na podstawie ich wirtualnych kodów.

Po rozpoznaniu wartości parametru `Key` odpowiadającej wciśniętemu klawiszowi strzałki skierowanej w lewą stronę, wykonują się instrukcje, które podzielone zostały na dwie części, jedna dotyczy ustawienia kursora z lewej strony znaku a druga z prawej. Jeśli kursor ustawiony jest z prawej strony, zmiana elementu bieżącego na poprzedni w wiązaniu kursorowym może nastąpić tylko wtedy, gdy poprzednik ten istnieje oraz gdy w jego polu `znak`

nie znajduje się litera należąca do nazwy funkcji ‘log’ ani znak akcji oraz gdy znak ten nie jest ukryty. W każdym innym przypadku, zmienna edytora `stat_kur` ustawiona zostaje w pozycji `false`, co powoduje ustawienie kursora przed znakiem znajdującym się w polu `znak` bieżącego elementu. Jeśli kursor ustawiony jest z lewej strony, zmiana elementu bieżącego nastąpi wtedy, gdy element ten istnieje oraz gdy w jego polu `znak` znajduje się litera tworząca z pozostałymi znakami nazwę funkcji ‘log‘, jak również wtedy, gdy w tym polu nie znajduje się znak potęgowania. Obecność znaku potęgowania w polu `znak` poprzednika powoduje przyjęcie przez zmienną `wsb` adresu struktury otwierającej, pomijając element z ukrytym znakiem potęgowania. Zmienna `stat_kur` ustawiona zostaje w pozycji `true`, lokalizując kursor za znakiem. Nie spełnienie opisanych warunków, pozostawia kursor w niezmienionym miejscu.

```
case Key of
     VK_Left: - jeśli wykryto użycie klawisza strzałki skierowanej w lewą stronę, to...
     if stat_kur then - czy kursor ustawiony jest z prawej strony znaku?
     begin
          if ssShift in Shift then - czy został naciśnięty klawisz Shift?
          begin
               . . .
          end;
          if wsb=wsp then stat_kur:=false - jeśli tak, to gdy element...
                                            bieżący jest pierwszym składnikiem listy,...
                                            ustaw zmienną stat_kur w pozycji false
          else
          begin - jeśli bieżący element nie jest pierwszym składnikiem listy, to...
               ax1:=wsb^.kur_l; - do zmiennej ax1 wczytaj adres poprzednika...
                                  bieżącego elementu w wiązaniu kursorowym
               if ax1<>nil then - czy poprzednik istnieje?
                  if not czy_log(wsb) then - jeśli tak, to czy znak wpisany...
                       w pole znak bieżącego elementu jest częścią nazwy ‘log’?
                     if czy_log(ax1) then stat_kur:=false - jeśli nie...
                           jest częścią nazwy ‘log’, to gdy znak zawarty w polu znak...
                           poprzednika jest częścią nazwy ‘log’, ustaw kursor...
                           przed znakiem
                     else -gdy znak poprzednika nie jest częścią nazwy ‘log’, to...
                         if ord(ax1^.znak) in dzed then
                            stat_kur:=false - gdy znak w polu znak...
                                    poprzednika należy do znaków akcji,...
                                    ustaw kursor przed znakiem
                         else - gdy znak poprzednika nie należy…
                                 do znaków akcji, to...
                              if ax1^.druk then wsb:=ax1 - jeśli znak…
                                poprzednika nie jest ukryty, wpisz do wskaźnika…
                                edytora wsb adres poprzednika
                              else stat_kur:=false - gdy znak jest…
                                                     ukryty ustaw kursor przed znakiem
                  else wsb:=ax1 - gdy znak bieżącego elementu należy do nazwy...
                                  ‘log’, wpisz do wskaźnika edytora adres poprzednika
               else stat_kur:=false - jeśli poprzednik nie istnieje,...
                                      ustaw kursor przed znakiem
          end
     end else - jeśli nie, kursor ustawiony jest z lewej strony znaku, więc...
     begin
```

```
            if ssShift in Shift then - czy został naciśnięty klawisz Shift?
            begin
                 . . .
            end;
            ax1:=wsb^.kur_l; - do zmiennej ax1 wczytaj adres poprzednika...
                               bieżącego elementu w wiązaniu kursorowym

            if ax1<>nil then - czy poprzednik istnieje?
            begin - jeśli tak, to...
                 if czy_log(ax1) then - czy w polu znak poprzednika...
                                        znajduje się litera, będąca częścią nazwy 'log'?

                 begin - jeśli tak, to...
                      stat_kur:=true; - ustaw kursor za znakiem
                      wsb:=ax1 - wskaźnikowi edytora wsb wpisz adres poprzednika
                 end else - jeśli znak poprzednika nie jest częścią nazwy 'log', to...
                     if ax1^.znak='^' then - czy w polu znak poprzednika...
                                             znajduje się znak potęgowania?

                     begin - jeśli tak, to...
                          ax1:=ax1^.kur_l; - pozyskaj w zmiennej ax1 adres...
                          poprzednika w wiązaniu kursorowym (struktura otwierająca)
                          stat_kur:=true; - ustaw kursor za znakiem
                          wsb:=ax1 - wskaźnikowi wsb wpisz adres struktury...
                                  otwierającej należącej do znaku potęgowania

                     end else wsb:=ax1 - jeśli w polu znak poprzednika...
                                         nie ma znaku potęgowania, wpisz do...
                                         wskaźnika edytora wsb adres poprzednika
            end
        end;
```

Po rozpoznaniu wartości parametru `Key`, odpowiadającej wciśniętemu klawiszowi strzałki skierowanej w prawą stronę i ustawionym kursorem przed znakiem, kursor zostaje przeniesiony do prawej strony znaku poprzez ustawienie zmiennej `stat_kur` w pozycji `true`. Jeśli kursor już znajduje się z prawej strony znaku, zmiana elementu bieżącego następuje wtedy, gdy następny element w wiązaniu kursorowym istnieje a jego znak nie jest ukryty. Jeśli w polu `znak` następnika znajduje się znak potęgowania, do wskaźnika edytora wpisany zostaje adres struktury zamykającej należącej do otoczenia tego znaku potęgowania, pomijając tym samym ukryty znak potęgowania. Kursor jest w tym przypadku ustawiany przed znakiem.

```
VK_Right: - jeśli wykryto użycie klawisza strzałki skierowanej w prawą stronę, to...
if not stat_kur then - czy kursor ustawiony jest przed znakiem?
   if ssShift in Shift then - jeśli tak, to czy został naciśnięty klawisz Shift?
   begin
        . . .
   end else stat_kur:=true - jeśli znak nie jest zaznaczony,...
                             ustaw kursor za znakiem

else - jeśli kursor ustawiony jest za znakiem, to...
begin
     if ssShift in Shift then - czy został naciśnięty klawisz Shift?
     begin
          . . .
     end;
     ax1:=wsb^.kur_p; - do zmiennej ax1 wczytaj adres następnika...
                        bieżącego elementu w wiązaniu kursorowym

     if ax1<>nil then - czy następnik istnieje?
```

```
            if ax1^.druk then - jeśli tak, to czy znak następnika nie jest ukryty?
               if czy_log(wsb) then - jeśli nie, to czy znak wpisany w pole...
                znak bieżącego elementu zawiera literę, będącą częścią nazwy 'log'?
               begin - jeśli tak, to...
                    if not czy_log(ax1) then stat_kur:=false; - jeśli...
                                następnik zawiera znak, będący częścią nazwy 'log' ...
                                to ustaw kursor przed znakiem
                    wsb:=ax1 - wpisz do wskaźnika edytora wsb adres następnika
               end else wsb:=ax1 - jeśli litera zawarta w polu znak bieżącego...
                                   elementu nie jest częścią nazwy 'log', to wpisz...
                                   do wskaźnika edytora wsb adres następnika
            else jeśli znak w polu znak następnika jest ukryty, to...
            if ax1^.znak='^' then - czy jest to znak potęgowania?
            begin - jeśli tak, to...
                 ax1:=ax1^.kur_p; - pozyskaj w zmiennej ax1 adres następnika...
                                    w wiązaniu kursorowym (struktura zamykająca)
                 stat_kur:=false; - ustaw kursor przed znakiem
                 wsb:=ax1 - wpisz do wskaźnika edytora wsb adres następnika
            end
end;
```

Rozpoznanie w instrukcji wyboru wartości odpowiadającej użytemu klawiszowi strzałki skierowanej do góry oraz stwierdzeniu istnienia składnika związanego z bieżącym elementem za pomocą pola `kur_g`, elementem bieżącym staje się składnik, którego adres znajduje się w tym polu. Nie trzeba sprawdzać, czy znak zawarty w tym składniku nie jest ukryty, gdyż elementy z ukrytymi znakami nie są wiązane z innymi składnikami poprzez pola `kur_g` lub `kur_d`.

```
VK_Up: - jeśli wykryto użycie klawisza strzałki skierowanej do góry, to...
if wsb^.kur_g<>nil then - czy w polu kur_g elementu bieżącego...
                          nie ma adresu pustego?
begin - jeśli nie ma adresu pustego to...
     ax1:=wsb^.kur_g - wpisz wskaźnikowi ax1 adres...
                       zawarty w polu kur_g bieżącego elementu
     if ssShift in Shift then - czy został naciśnięty klawisz Shift?
     begin
          . . .
     end;
     wsb:=ax1; - wpisz wskaźnikowi edytora wsb adres zapamiętany w zmiennej ax1
     ax1:=wsb^.kur_p; - do zmiennej ax1 wpisz adres następnika bieżącego elementu...
                        w wiązaniu kursorowym
     if ax1<>nil then - czy następnik istnieje?
        if ax1^.znak='^' then stat_kur:=false - jeśli w polu znak...
             następnika znajduje się znak potęgowania, ustaw kursor przed znakiem
        else stat_kur:=true - w przeciwnym przypadku ustaw kursor za znakiem
     else stat_kur:=true; - jeśli następnik nie istnieje, ustaw kursor za znakiem
end;
```

Jeśli w instrukcji wyboru wykryta została wartość odpowiadająca użytemu klawiszowi strzałki skierowanej do dołu, sprawdzana jest zawartość pola `kur_d` bieżącego elementu. Jeśli pole to nie zawiera adresu pustego, wskaźnik edytora przyjmuje adres zawarty w tym polu.

```
VK_Down: - jeśli wykryto użycie klawisza strzałki skierowanej do dołu, to...
if wsb^.kur_d<>nil then - czy w polu kur_d elementu bieżącego...
                          nie ma adresu pustego?
begin - jeśli nie, to...
     wsb:=wsb^.kur_d; - wpisz wskaźnikowi ax1 adres...
                        zawarty w polu kur_d bieżącego elementu
     if ssShift in Shift then - czy został naciśnięty klawisz Shift?
     begin
          . . .
     end;
     wsb:=ax1; - wpisz wskaźnikowi edytora wsb adres zapamiętany w zmiennej ax1
     ax1:=wsb^.kur_l; - do zmiennej ax1 wpisz adres poprzednika bieżącego...
                        elementu w wiązaniu kursorowym
     if ax1<>nil then - czy poprzednik istnieje?
        if ord(ax1^.znak) in dzed then stat_kur:=false jeśli w polu...
                znak poprzednika znajduje się znak akcji, ustaw kursor przed znakiem
        else - jeśli nie ma znaku akcji, to...
        if czy_log(ax1) then stat_kur:=false - jeśli w polu znak...
                             poprzednika znajduje się litera, będąca częścią...
                             nazwy 'log', ustaw kursor przed znakiem
        else stat_kur:=true - w przeciwnym przypadku ustaw kursor za znakiem
     else stat_kur:=true; - jeśli poprzednik nie istnieje, ustaw kursor za znakiem
end;
```

Pozostałe wartości użytych klawiszy nie wpływają na zmianę pozycji kursora. Mają jednak inne znaczenie:

```
VK_Back: - naciśnięto klawisz BACKSPACE
if zaz_odz then - czy zmienna zaz_odz jest ustawiona w pozycji true?
begin - jeśli tak, to aktywne jest zaznaczenie, więc...
     Form2.Odznacz; - odznacz zaznaczone znaki
     Form2.Odswiez_funkcje; - wywołaj procedurę Odswiez_funkcje
     zaz_odz:=false; - wyłącz zaznaczenie przez ustawienie zmiennej zaz_odz...
                       w pozycję false
     zaz_x:=0 - wpisz zmiennej zaz_x liczbę zero (nieokreślony kierunek zaznaczania)
end else - jeśli zaznaczenie nie jest aktywne, to...
    begin
         Form2.Usun_znak(true); - wywołaj procedurę usun_znak z opcją true
         Form2.Odswiez_funkcje - wywołaj procedurę Odswiez_funkcje
    end;
VK_Delete: - naciśnięto klawisz DELETE
if zaz_odz then - czy zmienna zaz_odz jest ustawiona w pozycji true?
begin
     . . .
end else - jeśli nie, to zaznaczenie jest nieaktywne, więc…
    begin
         Form2.Usun_znak(false); - wywołaj procedurę usun_znak z opcją…
                                   false
         Form2.Odswiez_funkcje - wywołaj procedurę Odswiez_funkcje
    end;
VK_Return: - naciśnięcie klawisza ENTER uruchamia analizę wpisanej funkcji lub pochodnej
```

XIV. 6. Położenie znaków na formie – procedura `Wyswietl`

Zadaniem procedury jest położenie na formie znaków zawartych w polach `znak` wszystkich utworzonych składników edytora. Składniki te posiadają pełny zestaw współrzędnych lokalnych, to znaczy, że każda wartość współrzędnej wyliczona została względem punktu zero, który znajduje się w lewym, górnym rogu formy. Na podstawie tych współrzędnych, każdy znak znajdzie się na formie we właściwym miejscu. Polecenia wstawiające znaki na formie należą do klasy `Canvas`, która umożliwia dowolny dobór kroju czcionki oraz jej wielkości i koloru. Możliwość ta została udostępniona użytkownikowi poprzez formę ustawień, dzięki czemu ustawiane parametry czcionki zostały zapisane w zmiennych globalnych modułu ustawień, z których korzysta opisywana procedura. Przed uruchomieniem pętli, w której odbywa się analiza każdego składnika edytora, zatrzymywane jest pulsowanie kursora oraz, po wyczyszczeniu formy z treści graficznych za pomocą bezparametrowej procedury `Refresh`, ustawiana zostaje wysokość czcionki poprzez przypisanie właściwości `Height` należącej do klasy `TFont` wartości pochodzącej ze zmiennej globalnej, odpowiadającej ustawionej wysokości czcionki w bloku regulacji.

```
procedure TForm2.Wyswietl;
var w1:lisc;
    koc,kot,kor:TColor;
    wa1,x,y,x1,y1,x2,y2:integer;
    wtxt:string;
    wtab:array[0..1] of char;
begin
     Form2.Timer1.Enabled:=false; - zatrzymaj pulsowanie kursora
     Form2.Refresh; - wyczyść zawartość formy z treści graficznych
     Canvas.Font.Height:=Form6.RZ; - ustaw wysokość czcionki

     Canvas.TextOut(AXX,AY,tekst2); - połóż na formie przedrostek pochodnej...
                                      zawarty w stałej tekst2

       (współrzędna pozioma AXX jest stałą modułu zainicjowaną wartością 20,...
       natomiast współrzędna pionowa AY jest zmienną modułu, której wartość...
       została przypisana na etapie przygotowywania formy do pracy przez procedurę...
       Poczatek)

     w1:=wsp; - przypisz wskaźnikowi w1 adres pierwszego składnika listy
     while w1<>nil do - wykonaj krok pętli, póki wskaźnik w1 nie ma adresu pustego
     begin
          if w1^.zazn then - czy pole zazn badanego elementu zostało ustawione...
                             w pozycji true?

          begin - jeśli tak, to znak jest zaznaczony, czyli...
               koc:=kol_c; - zmiennej lokalnej koc przypisz wartość koloru...
                             zaznaczenia czcionki

               kot:=kol_t; - zmiennej lokalnej kot przypisz wartość koloru...
                             zaznaczenia wypełnienia czcionki

               kor:=kot - zmiennej lokalnej kor przypisz wartość koloru...
                          zaznaczenia znaków akcji

          end else - jeśli nie, znak nie jest zaznaczony, więc...
              begin
                   koc:=Form6.F2; - zmiennej lokalnej koc przypisz wartość...
                                    koloru czcionki ze zmiennej bloku regulacji

                   kot:=Form6.F3; zmiennej lokalnej kot przypisz wartość…
```

```
                                        koloru tła formy...
                                  (wypełnienie czcionki będzie identyczne z kolorem formy)
                        kor:=koc  -  zmiennej lokalnej kor przypisz wartość...
                                     koloru czcionki
              end;
        wa1:=ord(w1^.znak);  -  do zmiennej wa1 wpisz wartość znaku...
              zawartego w polu znak analizowanego w pętli składnika listy
        case wa1 of
              . . .
```

Po położeniu na formie tekstu ‘f’ =‘ zawartego w stałej tekst2 i przypisaniu lokalnemu wskaźnikowi w1 adresu pierwszego składnika listy, pętla zostaje uruchomiona. Analiza składników edytora w pętli polega na identyfikacji znaków wpisanych pola znak każdego składnika, począwszy od pierwszego a skończywszy na ostatnim. Znaki te, a właściwie ich wartości w kodzie ASCII, identyfikowane są w instrukcji wyboru case.

Na początku każdego kroku pętli sprawdzana jest informacja z pola zazn badanego elementu, czy znak ten nie został przez użytkownika zaznaczony. Na podstawie wartości z tego pola, lokalnym zmiennym procedury zostają przypisane wartości kolorów, pochodzące ze zmiennych lokalnych klasy TForm2 (kol_c i kol_t) lub zmiennych globalnych bloku regulacji, odpowiadających zestawom kolorów dla zaznaczonych i odznaczonych znaków. Tuż przed położeniem znaku na formie, kolor znaku, jego grubość oraz wypełnienie, muszą zostać ustawione przez przypisanie polom Color oraz Width należących do klasy TPen oraz Style należącej do klasy TBrush, wartości od lokalnych zmiennych.

Po rozpoznaniu wartości 47 odpowiadającej znakowi dzielenia, lokalnym zmiennym przypisane zostają wartości współrzędnych z pól lx, px i y1. Są to współrzędne poziome z lewej i prawej strony znaku oraz współrzędna pionowa rzeczywista kreski ułamkowej. Na podstawie tych współrzędnych rysowana jest pozioma linia w kolorze, którego wartość została zapamiętana w zmiennej lokalnej kor koloru czcionki. Jej grubość jest stała i wynosi dwa piksele.

```
47:
begin
      x1:=w1^.lx;  -  do zmiennej x1 wpisz wartość współrzędnej poziomej z pola lx
      x2:=w1^.px;  -  do zmiennej x2 wpisz wartość współrzędnej poziomej z pola px
      y1:=w1^.y1;  do zmiennej y1 wpisz wartość współrzędnej pionowej z pola y1
      Canvas.Pen.color:=kor;  -  ustaw kolor rysowanej linii na podstawie...
                                    zmiennej koloru kor
      Canvas.Pen.width:=2;  -  ustaw grubość rysowanej linii na 2 piksele
      Canvas.MoveTo(x1,y1);  -  zaznacz początek rysowanej linii w miejscu...
                                    określonym przez współrzędne x1 i y1
      Canvas.LineTo(x2,y1)  -  narysuj linię od zaznaczonego początku do końca,...
                                    określonego przez współrzędne x2 i y1
end;
```

Po rozpoznaniu wartości 92 odpowiadającej znakowi pierwiastkowania, zadanie jest nieco trudniejsze, ponieważ na podstawie początkowych i końcowych współrzędnych tego znaku trzeba wyliczyć współrzędne wszystkim czterem częściom, z których składa się kompletny znak pierwiastkowania oraz narysować go na formie. Pierwsza część to pozioma kreska, nad którą zwykle znajduje się stopień pierwiastka. Jej długość równa jest 1/8 wysokości znaku pierwiastkowana, powiększona o szerokość pionowej kreski pierwiastka. Początek kreski ustala początkowa współrzędna pozioma zawarta w polu lx elementu znaku pierwiastko-

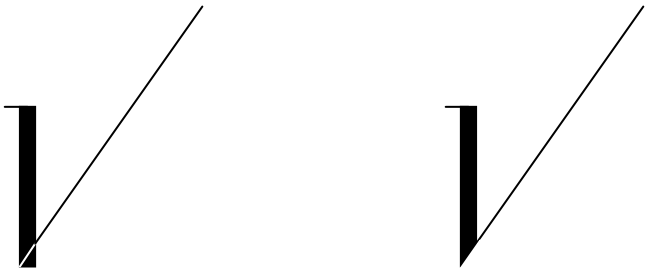
wania, zaś jej wysokość to współrzędna zawarta w połowie między górną współrzędną pionową y1 a dolną y2 tego elementu.

```
92:
begin
     x1:=w1^.lx+trunc((w1^.y2-w1^.y1)/8); - w zmiennej x1 wylicz...
          wstępną wartość końcowej współrzędnej poziomej pierwszej części pierwiastka,...
          równą sumie wartości zawartej w polu lx elementu znaku pierwiastkowania...
          oraz jego wysokości podzielonej przez osiem
     y1:=w1^.y1+trunc((w1^.y2-w1^.y1)/2); - w zmiennej y1 wylicz...
                                  wartość współrzędnej pionowej, równą sumie...
                                  początkowej współrzędnej pionowej, zawartej...
                                  w polu y1 elementu znaku pierwiastkowania...
                                  i połowie jego wysokości
     x:=round((w1^.y2-w1^.y1)/12); - w zmiennej x wylicz grubość pionowej...
          kreski (druga część pierwiastka) równą 1/12 wysokości znaku pierwiastkowania
     Canvas.Pen.Color:=kor; - określ kolor rysowanej linii
     Canvas.Pen.Width:=2; - określ grubość rysowanej linii
     y2:=y1+round(x/4); - w zmiennej y2 wylicz ostateczną wartość współrzędnej...
                          pionowej pierwszej części pierwiastka, równej sumie...
                          wartości wyliczonej w zmiennej y1 i niewielkiej korekcji,...
                          (linie należące do pierwszej i drugiej części pierwiastka...
                           muszą na siebie zachodzić, by nie było przerwy między nimi)
     Canvas.MoveTo(w1^.lx,y2); - określ punkt początkowy rysowanej linii
     Canvas.LineTo(x1+x,y2); - narysuj linię (pierwsza część pierwiastka)
```

Druga część znaku pierwiastka, czyli pionowa kreska, której początkową współrzędną pionową określa zmienna y2, zaś końcową – wartość zawarta w polu y2 elementu znaku pierwiastkowania, rysowana jest wielokrotnie w pętli for w odstępach jednego piksela, uzyskując linię o grubości określonej w zmiennej x. Linia ta nie jest jeszcze gotowa, gdyż jej pożądanym kształtem ma być trapez. Nim spodziewany kształt zostanie uzyskany, najpierw rysowana jest ukośna kreska biegnąca od najniższego punktu pierwiastka do najwyższego. Obie współrzędne pionowe są już znane, gdyż pochodzą z pól y1 i y2 elementu znaku pierwiastkowania. Znana jest też początkowa współrzędna pozioma, która znajduje się w zmiennej x1. Brakuje końcowej współrzędnej poziomej, która zostaje wyliczona w zmiennej x2. Jej wartością jest suma początkowej współrzędnej poziomej znaku pierwiastkowania i połowy jego wysokości.

```
Canvas.Pen.Width:=1; - określ grubość rysowanej linii na jeden piksel
for y:=0 to x do - wykonaj krok pętli, póki wartość w zmiennej y nie będzie większa...
                   od wartości w zmiennej x
begin
     Canvas.MoveTo(x1+y,y2); - określ punk początkowy rysowanej linii
     Canvas.LineTo(x1+y,w1^.y2) - narysuj linię od punktu początkowego...
                                  (druga część pierwiastka)
end;
x2:=w1^.lx+trunc((w1^.y2-w1^.y1)/2); - wylicz w zmiennej x2 końcową...
                        współrzędną poziomą dla trzeciej części znaku pierwiastka
Canvas.Pen.Width:=2; - określ grubość rysowanej linii na dwa piksele
Canvas.MoveTo(x1,w1^.y2); - określ punkt początkowy rysowanej linii
Canvas.LineTo(x2,w1^.y1); - narysuj linię (trzecia część znaku pierwiastka)
```

Uzyskana, ukośna kreska ma swój początek w lewym, dolnym rogu pionowej kreski w kształcie prostokąta. Wyjaśnia to lewa strona rysunku 59.



Rys. 59. Wygląd niekompletnego znaku pierwiastka, z uwidocznieniem zbędnego fragmentu pionowej kreski z lewej strony i po usunięciu tego fragmentu z prawej

Uzyskany efekt wizualny jest daleki od oczekiwań, dlatego zbędny fragment prostokąta musi być usunięty. Najprostszym sposobem okazało się zamazanie go ciągiem ukośnych linii rysowanych w kolorze tła formy. Ponieważ komplet współrzędnych ukośnej kreski został zachowany, można je wykorzystać do ponownego narysowania ukośnych linii, ale w kolorze tła formy i w odstępie jednego piksela od już narysowanej linii. Uzyskany efekt pokazuje prawa strona rysunku 59.

```
Canvas.Pen.Color:=Form6.F3D; – ustaw kolor rysowanej linii na kolor tła formy
Canvas.Pen.Width:=1; – ustaw grubość rysowanej linii na jeden piksel
for y:=1 to x do – wykonaj krok pętli, póki wartość w zmiennej y nie będzie większa...
                  od wartości w zmiennej x
begin
     Canvas.MoveTo(x1+y,w1^.y2); – określ punk początkowy rysowanej linii
     Canvas.LineTo(x2+y,w1^.y1) – narysuj linię w kolorze tła formy
end;
```

Warto zauważyć, że powyższa pętla wykonuje się od początkowej wartości zmiennej `y` równej jeden sprawiając, że pierwsza rysowana linia w kolorze tła formy jest oddalona o odległość jednego piksela od tej, która ma być widoczna.

By znak pierwiastka był kompletny, pozostało narysowanie ostatniej, czwartej jej części, czyli kreski podpierwiastkowej. Jej współrzędne są już znane, pozostaje już tylko określenie koloru rysowanej linii i jej grubości oraz narysowanie linii na formie.

```
Canvas.Pen.Color:=kor; – ustaw kolor rysowanej linii
Canvas.Pen.Width:=2; – ustaw grubość rysowanej linii
Canvas.MoveTo(x2,w1^.y1); – określ punkt początkowy
Canvas.LineTo(w1^.px,w1^.y1) narysuj linię (czwarta część znaku pierwiastka)
```

Gdy rozpoznany znak należy do znaków działania, czyli dodawania, mnożenia lub odejmowania, dla każdego z nich wyliczona została szerokość z pewnym zapasem. Obecność zapasu szerokości pozwala na powstanie niewielkiego odstępu od poprzedniego i następnego znaku. Wpływa to niewątpliwie na poprawę estetyki funkcji, ale pod warunkiem, że zapas ten będzie równo rozdzielony między lewą a prawą stroną znaku działania, dlatego równa połowa tego zapasu dodana została do współrzędnej `lx` elementu z wpisanym znakiem działania, otrzymując nową początkową współrzędną poziomą dla tego znaku, a tym samym równy odstęp od sąsiadujących znaków. Sam zapas wynika z różnicy między wyliczonymi wartościa-

mi, zawartymi w polach `px` a `lx` elementu edytora oraz rzeczywistą szerokością znaku, wyliczoną przez funkcję `TextWidth`.

Jeśli rozpoznany został znak mnożenia, jego podstawowy symbol w postaci gwiazdki zostaje zamieniony na kropkę, której kod ASCII wynosi 183.

```
wa1:=ord(w1^.znak); - do zmiennej wa1 wpisz wartość znaku...
     zawartego w polu znak analizowanego w pętli składnika listy
case wa1 of
. . .
else
begin
     if wa1 in dzia then - czy wartość znaku należy do znaków działania
     begin - jeśli tak, to...
          if wa1=42 then wtab[0]:=chr(183) - jeśli jest to znak mnożenia...
               to wpisz do łańcucha wtab znak kropki, której kod ASCII wynosi 183
             else wtab[0]:=w1^.znak; - każdy inny znak działania wpisz...
                                       do łańcucha wtab
          wtab[1]:=chr(0); - dopisz do łańcucha wtab znak pusty
          wtxt:=strpas(wtab); - dokonaj konwersji łańcucha wtab typu Char...
                                do łańcucha wtxt typu string
          y1:=w1^.y2-w1^.y1; - w zmiennej y1 wylicz wysokość znaku
          Canvas.Font.Height:=y1; - ustaw wysokość czcionki na podstawie...
                                    wartości w zmiennej y1
          Canvas.Font.Color:=koc; - określ kolor czcionki
          Canvas.Brush.Color:=kot; - określ kolor wypełnienia czcionki
          x1:=w1^.px-w1^.lx; - w zmiennej x1 wylicz różnicę między wartościami...
                               w polach px a lx składnika listy z wpisanym znakiem działania
          x2:=Canvas.TextWidth(wtxt); - w zmiennej x2 wylicz rzeczywistą...
                                        szerokość znaku, podając w parametrze funkcji TextWidth...
                                        łańcuch typu string z zawartym w niej znakiem działania

          x1:=w1^.lx+trunc((x1-x2)/2); - wylicz w zmiennej x1 wartość...
                                         nowej współrzędnej poziomej, dodając do...
                                         dotychczasowej wartości współrzędnej...
                                         połowę naddatku szerokości
          y1:=w1^.y1; - wpisz do zmiennej y1 wartość górnej współrzędnej pionowej
          Canvas.TextOut(x1,y1,wtxt) - wstaw znak na formie w miejscu...
                                       wskazanym przez zmienne x1 i y1
     end else
```

Gdy rozpoznany znak nie należy do znaków działania i nie jest to znak ukryty, tak jak w poprzednim przykładzie, znak wprowadzony zostaje do łańcucha `wtab` typu `Char`, po czym jego zawartość zostaje przekonwertowana do łańcucha typu `string`. Wysokość czcionki zostaje ustawiona na podstawie różnicy między dolną a górną współrzędną rzeczywistą znaku, po czym znak zostaje umieszczony na formie Jeśli wartością znaku jest liczba 112, tuż przed położeniem znaku na formie, zmieniony zostaje krój czcionki na '`Symbol`', co spowoduje zamianę polskiej litery 'p' na grecką 'π'. Po wyświetleniu znaku greckiego, krój czcionki zostaje przywrócony.

```
if wa1 in dzia then - czy wartość znaku należy do znaków działania?
begin

end else - jeśli nie, to...
    if w1^.druk then - czy znak nie jest ukryty?
    begin - jeśli nie, to...
         wtab[0]:=w1^.znak; - wpisz do łańcucha wtab znak zawarty w polu znak
         wtab[1]:=chr(0); - dopisz do łańcucha wtab znak pusty
         wtxt:=strpas(wtab); - dokonaj konwersji łańcucha wtab typu Char...
                               do łańcucha wtxt typu string

         y1:=w1^.y2-w1^.y1; - w zmiennej y1 wylicz wysokość znaku
         Canvas.Font.Height:=y1; - ustaw wysokość czcionki na podstawie...
                                   wartości w zmiennej y1

         Canvas.Font.Color:=koc; - określ kolor czcionki
         Canvas.Brush.Color:=kot; - określ kolor wypełnienia czcionki
         x1:=w1^.lx; - wpisz do zmiennej x1 wartość współrzędnej poziomej znaku
         y1:=w1^.y1; - wpisz do zmiennej y1 wartość współrzędnej pionowej znaku
         if wa1=112 then - czy wartość znaku odpowiada literze 'p'?
         begin - jeśli tak, to...
              Canvas.Font.Name:='Symbol'; - ustaw krój czcionki na Symbol
              Canvas.TextOut(x1,y1,wtxt); - wstaw znak na formie w miejscu...
                                            wskazanym przez zmienne x1 i y1

              Canvas.Font.Name:=Form6.F1 - ustaw krój czcionki na podstawie...
                                           zmiennej F1 należącej do modułu ustawień

         end else Canvas.TextOut(x1,y1,wtxt) -jeśli nie jest to wartość...
                                              litery 'p', to wstaw znak na formie w miejscu...
                                              wskazanym przez zmienne x1 i y1
    end;
```

Po analizie wszystkich elementów edytora, pętla zostaje opuszczona a kursor uruchomiony.

```
Form2.Timer1.Enabled:=false; - zatrzymaj pulsowanie kursora
. . .
w1:=wsp; - przypisz zmiennej w1 adres pierwszego składnika listy
while w1<>nil do - wykonaj krok pętli, póki wskaźnik w1 nie ma adresu pustego
begin
     . . .
     w1:=w1^.prawa - we wskaźniku w1 pozyskaj adres następnika w wiązaniu głównym
end;
Form2.Timer1.Enabled:=true - uruchom pulsowanie kursora
```

XIV. 7. Ustalenie szerokości znaku – procedura `Szerokosc`

Procedura wylicza rzeczywistą szerokość znaku, jaka wynika z ustawionej wysokości i kroju czcionki oraz samego znaku. Uzyskana wielkość, po zsumowaniu z początkową współrzędną znaku zawartą w polu `lx` elementu edytora, zostaje wpisana do pola `px` tego elementu. W przypadku znaku działania, czyli dodawania, odejmowania lub mnożenia, do jego szerokości dodana zostaje szerokość znaku kropki o kodzie ASCII równym 46. Uzyskana w ten sposób suma szerokości dwóch znaków trafia do pola `px` elementu edytora z wpisanym znakiem działania. Celem sztucznego powiększenia szerokości znaku działania jest

utworzenie dodatkowego odstępu do sąsiadujących znaków z lewej i prawej strony, co niewątpliwie wpływa na estetykę funkcji na ekranie. Dla znaku mnożenia w postaci gwiazdki zostaje wyliczona szerokość znaku kropki o kodzie ASCII równym 183.

```
procedure TForm2.Szerokosc(const azak:lisc);
var znaki:array[0..2] of char;
    zn1:char;
    ciag:string;
    s2,sp1:integer;
begin
     sp1:=azak^.lx; - zapamiętaj w zmiennej sp1 wartość początkowej współrzędnej...
                      poziomej znaku, zawartą w polu lx elementu edytora
     zn1:=azak^.znak; - w zmiennej znakowej zn1 zapamiętaj znak zawarty w polu...
                        znak elementu edytora
     s2:=ord(zn1); - w zmiennej s2 zapamiętaj wartość znaku w kodzie ASCII
     if s2=42 then zn1:=chr(183); - jeśli wartością znaku jest liczba 42,...
                      ponieważ odpowiada mnożeniu, wpisz do zmiennej znakowej...
                      zn1 znak kropki o kodzie ASCII równym 183
     Canvas.Font.Height:=azak^.y2- azak^.y1; - ustaw wysokość czcionki
     if s2 in dzia then - czy wartość znaku należy do znaków działania?
     begin - jeśli tak, to...
          znaki[0]:=chr(46); - wpisz do łańcucha znakowego znaki znak kropki
          znaki[1]:=zn1; - dopisz do łańcucha znak działania
          znaki[2]:=chr(0); - dopisz do łańcucha znak pusty
         ciag:=strpas(znaki); - dokonaj konwersji łańcucha znaki typu Char...
                              do łańcucha ciag typu string
          sp1:=sp1+Canvas.TextWidth(ciag) - w zmiennej sp1 uzyskaj sumę...
                          początkowej współrzędnej poziomej i wyliczonej przez...
                          funkcję TextWidth długości łańcucha ciag
     end else - jeśli nie jest to znak działania, to...
         if s2<>32 then - czy wartość znaku jest różna od 32?
         begin - jeśli tak, to nie jest to znak spacji, więc...
              znaki[0]:=zn1; - wpisz do łańcucha znakowego znaki znak...
                               zapamiętany w zmiennej zn1
              znaki[1]:=chr(0); - dopisz do łańcucha znak pusty
             ciag:=strpas(znaki); - dokonaj konwersji łańcucha znaki...
                                  typu Char do łańcucha ciag typu string
              sp1:=sp1+Canvas.TextWidth(ciag); - w zmiennej sp1...
                          uzyskaj sumę początkowej współrzędnej poziomej...
                          i wyliczonej przez funkcję TextWidth długości...
                          łańcucha ciag
         end;
     azak^.px:=sp1 - wpisz do pola px elementu edytora otrzymanego w parametrze,...
                     wartość końcowej współrzędnej poziomej, zapamiętanej w zmiennej sp1
end;
```

Analizując powyższy kod, można mieć obawy, czy gdyby do procedury dotarł element edytora z wpisanym znakiem akcji, nie będzie podejmowana próba poprawienia już wcześniej wyliczonej długości kreski ułamkowej czy pierwiastka. Tak się składa, że opisywana procedura wywoływana jest w określonych okolicznościach, ściśle kontrolowanych przez program, dlatego nie ma potrzeby stosowania warunków wykrywających pojawienie się znaku akcji, gdyż taki przypadek nie ma prawa się wydarzyć.

XIV. 8. Przesunięcie znaków o szerokość nowego znaku – procedura `Przesun`

Zadaniem procedury jest takie skorygowanie wartości w polach współrzędnych poziomych, należących do elementów związanych ze sobą wiązaniami kursorowymi, że pozostałe znaki usytuowane z prawej strony znaku zawartego w elemencie wskazywanym przez parametr nie będą na niego zachodzić ani być od niego oddalone. Efektem tej korekcji jest odsunięcie lub przysunięcie znaków z prawej strony znaku odniesienia, czyli tego zawartego w polu `znak` elementu wskazywanego przez parametr.

Do oceny wielkości przesunięcia istotne jest istnienie różnicy między wartością końcowej współrzędnej poziomej zawartej w polu `px` elementu otrzymanego w parametrze a wartością w polu `lx` następnego elementu związanego wiązaniem kursorowym z prawej strony. Jeśli istnieje różnica między tymi wartościami, prócz przesunięcia następnego znaku do znaku odniesienia, trzeba jeszcze o tę samą różnicę przesunąć pozostałe znaki, które znajdują się na tym samym poziomie, co znak odniesienia. W celu znalezienia ostatniego elementu edytora przeznaczonego do przesunięcia, wywoływana jest pętla, w której szukany jest taki składnik, który nie posiada swego następnika w wiązaniu kursorowym. Jego adres zapamiętany zostaje we wskaźniku `uk`. Jeśli wskaźnik ten nie ma adresu pustego, oznacza to, że obecny jest przynajmniej jeden element, który może wymagać korekcji współrzędnych poziomych. Do wyliczenia korekcji brane są pod uwagę dwie możliwości:

- gdy pozycja znaku odniesienia zmieniła się, wielkością przesunięcia następnych znaków jest różnica między wartością w polu `px` elementu odniesienia a polem `lx` następnego elementu. O takim sposobie wyliczania przesunięcia informuje procedurę wartość `true` zawarta w drugim parametrze;
- gdy między istniejące znaki funkcji dołączony zostaje nowy znak, wielkością przesunięcia jest wtedy szerokość nowego znaku. O tym sposobie wyliczania wielkości przesunięcia informuje wartość `false` drugiego parametru.

Wszystkie te elementy edytora, którym należy zmodyfikować współrzędne poziome, poprawiane są w pętli, począwszy od następnika elementu odniesienia w wiązaniu głównym a skończywszy na tym, którego adres zapamiętany został we wskaźniku `uk`.

```
Procedure TForm2.Przesun(const bzak:lisc;tryb:boolean);
var uk,u1:lisc;
    kor:integer;
begin
     uk:=nil; - przypisz zmiennej lokalnej uk identyfikator adresu pustego
     u1:=bzak^.kur_p; - zmiennej u1 wpisz adres następnika elementu odniesienia...
                        w wiązaniu kursorowym
     while u1<>nil do - wykonaj krok pętli, póki zmienna u1 nie ma adresu pustego
     begin
          uk:=u1; - przypisz zmiennej uk adres elementu analizowanego w pętli
          u1:=u1^.kur_p - w zmiennej u1 pozyskaj adres następnika...
                          w wiązaniu kursorowym
     end;
     if uk<>nil then - czy wskaźnik uk nie ma adresu pustego?
     begin - jeśli nie, to...
          u1:=bzak^.kur_p; - zmiennej u1 wpisz adres następnika elementu...
                             odniesienia w wiązaniu kursorowym
          if tryb then kor:=bzak^.px-u1^.lx - jeśli parametr tryb...
                       ustawiony został w pozycji true, w zmiennej kor wylicz...
                       różnicę między wartościami w polu px elementu odniesienia...
```

```
                                  i lx następnego elementu
                    else kor:=bzak^.px-bzak^.lx; - jeśli parametr tryb...
                                  ustawiony został w pozycji false, w zmiennej kor wylicz...
                                  różnicę między polami px a lx elementu odniesienia
               u1:=bzak^.prawa; - zmiennej u1 wpisz adres następnika elementu...
                                  odniesienia w wiązaniu głównym
               while u1<>nil do - wykonaj krok pętli, póki zmienna u1...
                                  nie ma adresu pustego
               begin
                     u1^.lx:=u1^.lx+kor;  } do pól współrzędnych poziomych ...
                     u1^.px:=u1^.px+kor;  } wpisz sumę dotychczasowej wartości...
                                            i wyliczonej korekcji w zmiennej kor
                     if u1=uk then break; - gdy zmienna u1 uzyska adres...
                                            zawarty w zmiennej uk, opuść pętlę
                     u1:=u1^.prawa - w zmiennej u1 pozyskaj adres następnika...
                                     w wiązaniu głównym
               end;
          end;
     end;
```

XIV. 9. Zaznaczanie

W tym rozdziale przedstawiono mechanizm zaznaczania wybranej przez użytkownika części wprowadzonej funkcji czy pochodnej. Początek tego mechanizmu opisano w rozdziale V.1 opisującym przygotowanie formy edukacji do pracy, a w nim wybrania przez program kolorów zaznaczania. Ważnym elementem edytora umożliwiającym zaznaczanie jest obecność logicznego pola `zazn` zawartego w strukturze `lisc`. Pole to zostaje ustawione w pozycji `true`, gdy znak wpisany w pole `znak` elementu edytora został zaznaczony.

Zaznaczanie możliwe jest zarówno poprzez przesuwanie myszką nad treścią funkcji czy pochodnej, mając wciśnięty lewy przycisk myszki, jak i przez naciśnięcie jednego z klawiszy kierunkowych klawiatury przy wcześniejszym naciśnięciu i przytrzymaniu klawisza `Shift`. Obydwie możliwości zostaną opisane w osobnych podrozdziałach.

XIV. 9.1. Zaznaczanie myszką

Do zaznaczania myszką wykorzystane zostało zdarzenie uruchamiane podczas przesuwania myszką po formie. W wyniku tego zdarzenia zostaje uruchomiona procedura `Form-MouseMove`, która zawiera w swych parametrach informację o naciśnięciu jednego z klawiszy specjalnych klawiatury lub przycisku myszki komputerowej oraz wartości współrzędnych aktualnej pozycji myszki względem lewego, górnego narożnika formy. W pierwszej instrukcji procedury sprawdzany jest fakt naciśnięcia lewego przycisku myszki – jeśli przycisk został naciśnięty, wywoływana zostaje funkcja `Nowy_biez`, która na podstawie wartości współrzędnych aktualnej pozycji myszki wybiera nowy element edytora, będący elementem zaznaczenia. Adres tego elementu zostaje zapamiętany w lokalnej zmiennej `nwsp`. Ważną częścią zaznaczania jest teraz określenie jego kierunku. Do zapamiętania kierunku zaznaczania służy zmienna prywatna `zaz_x`, należąca do klasy `TForm2`, która przyjmuje wartości od zera do dwóch. Wartość zero jest informacją, że kierunek nie został jeszcze określony, dlatego w kolejnych instrukcjach zmiennej tej przypisana zostaje liczba jeden lub dwa, zależnie od tego,

czy wartość współrzędnej poziomej nowego elementu `lisc` jest większa lub mniejsza od tej samej współrzędnej elementu bieżącego wskazywanego przez wskaźnik edytora `wsb`. Może się zdarzyć, że obydwa wskaźniki: `wsb` i `nwsp`, mają takie same adresy, wówczas do określenia kierunku zaznaczania wykorzystana zostaje różnica statusu kursora przed i po wywołaniu funkcji `Nowy_biez` – jeśli przed wywołaniem funkcji kursor znajdował się przed znakiem, a po jej opuszczeniu – za znakiem, kierunek zaznaczania przebiega w prawą stronę, więc zmienna `zaz_x` przyjmuje wartość jeden, w przeciwnym przypadku kierunek jest odwrotny, więc zmienna ta przyjmuje wartość dwa.

```
procedure TForm2.FormMouseMove(Sender: TObject;
          Shift: TShiftState; X,Y: Integer);
var nwsp:lisc;

     procedure Zaznacz(pocz,kon:lisc;kier:byte);

begin
     if ssLeft in Shift then - czy naciśnięty został lewy przycisk myszki?
     begin - jeśli tak, to...
          nwsp:=Form2.Nowy_biez(X,Y); na podstawie współrzędnych X i Y ...
           pozycji myszki znajdź element zaznaczenia i zapamiętaj go w zmiennej nwsp

          if nwsp<>nil then - czy element zaznaczenia został znaleziony?
          begin - jeśli tak, to…
               if zaz_x=0 then - czy kierunek zaznaczania został ustalony?
               begin - jeśli nie, to…
                    if nwsp^.lx>wsb^.lx then zaz_x:=1 –jeśli...
                     wartość współrzędnej poziomej lx elementu zaznaczenia...
                     jest większa od wartości współrzędnej elementu bieżącego,...
                     wpisz zmiennej zax_x jedynkę (zaznaczanie w prawo)
                    else if nwsp^.lx<wsb^.lx then zaz_x:=2
                    jeśli wartość współrzędnej poziomej elem. zaznaczenia...
                    jest mniejsza od tej samej wartości elementu bieżącego,...
                    wpisz zmiennej zaz_x liczbę dwa (zaznaczenie w lewo)

                        else if pocz_kur<>stat_kur then
                      jeśli element zaznaczenia i bieżący to ten sam element,...
                      to czy status kursora przed i po wywołaniu funkcji...
                      Nowy_biez różni się?

                                  if stat_kur then zaz_x:=1
                                  else zaz_x:=2;
                                  jeśli tak, to wpisz do zmiennej zaz_x...
                                  wartość jeden, gdy po opuszczeniu…
                                  funkcji Nowy_biez kursor znajduje się…
                                  za znakiem lub dwa, w przeciwnym…
                                  przypadku

                    zaz_odz:=true; - zaznaczenie jest aktywne, dlatego…
                                ustaw zmienną zaz_odz w pozycji true

                    Form2.ShowHint:=false - zablokuj automatyczne...
                              pojawianie się podpowiedzi w tzw. chmurce,...
                              by nie przeszkadzała podczas zaznaczania
               end;
               if zaz_x>0 then - czy ustalony został kierunek zaznaczania?
               begin - jeśli tak, to...
                    Zaznacz(wsb,nwsp,zaz_x); - wywołaj procedurę...
                                                Zaznacz

                    Form2.Wyswietl - wywołaj procedurę Wyswietl
               end;
```

```
                end;
            end;
        end;
```

Od tej chwili, jeżeli zmieni się kierunek zaznaczania, każdy znak zaznaczany w odwrotnym kierunku zostanie odznaczony. Zaznaczanie i odznaczanie odbywa się w wewnętrznej procedurze `Zaznacz`, której w parametrach przekazane zostają adres elementu bieżącego, nowego elementu zaznaczenia oraz ustalony kierunek zaznaczania. By efekt zaznaczenia był widoczny na formie, po opuszczeniu procedury wywoływana jest procedura `Wyswietl`.

XIV. 9.1.1. Procedura `Zaznacz`

Zaznaczenie znaku polega na ustawieniu w pozycji `true` pola `zazn`, należącego do elementu edytora `lisc`, w którego polu `znak` znajduje się zaznaczany znak. W pierwszych instrukcjach procedury ustalany zostaje pierwszy i ostatni element zaznaczenia. Wyznaczenie tych elementów jest zależne od kierunku zaznaczania – jest on dostępny w procedurze poprzez parametr o nazwie `kier` oraz statusu kursora. W przypadku zaznaczania „w prawo”, o czym informuje parametr `kier` o wartości jeden, pierwszym zaznaczanym elementem jest ten, wskazywany przez parametr `pocz` lub jego następnik w wiązaniu kursorowym, zależnie od tego czy kursor w chwili rozpoczęcia zaznaczania był usytuowany z lewej lub prawej strony znaku bieżącego, natomiast ostatnim elementem zaznaczania jest ten, wskazywany przez parametr `kon` lub jego poprzednik w wiązaniu kursorowym, zależnie od tego czy kursor w momencie wyłonienia następnego elementu zaznaczenia był umieszczony z prawej lub lewej strony zaznaczanego znaku. W przypadku zaznaczania „w lewo”, o czym informuje parametr `kier` o wartości dwa, wybór pierwszego i ostatniego elementu zaznaczenia jest odwrotny do poprzedniego przypadku zaznaczania „w prawo”. Może się zdarzyć, że obydwa parametry będą wskazywały na ten sam element, dlatego kwestię wyboru początkowego elementu zaznaczenia powinien rozstrzygnąć fakt przemieszczenia się kursora w ramach zaznaczanego znaku.

Zapamiętanie pozycji kursora w chwili rozpoczęcia zaznaczania odbywa się w wyniku kliknięcia myszką w miejscu zaznaczanego tekstu i wywołania tym kliknięciem procedury zdarzenia `FormMouseDown`.

Ponieważ adres bieżącego oraz nowego elementu zaznaczenia zostały procedurze przekazane poprzez wartość, można potraktować parametry `pocz` i `kon` jak lokalne zmienne procedury. Pozwala to na bezpieczne modyfikowanie tych parametrów nowymi wartościami, bez wpływu na parametry aktualne.

Gdy element został wybrany, w pętli `while..do` zaznaczane zostają te elementy edytora, które zawarte zostały między tymi, na które wskazują obydwa parametry lokalne `pocz` i `kon` – pozostałe elementy zostają odznaczane. Ponieważ kierunek pracy pętli zawsze jest jednakowy – od pierwszego składnika edytora do ostatniego, rozpoczęcie zaznaczania wymaga porównania aktualnego adresu znajdującego się w zmiennej sterującej pętlą z adresem jednego z dwóch parametrów lokalnych. Gdy pierwszy napotkany adres zawiera parametr `kon`, następuje zamiana adresów tak, by adres ostatniego zaznaczanego składnika, na pewno znalazł się w parametrze `kon`. Do rozróżnienia w pętli początku i końca zaznaczania, pomocną okazała się lokalna zmienna logiczna `zap`, która przyjmuje wartość `true` przy rozpoczęciu zaznaczania i `false` przy jego końcu.

```
procedure Zaznacz(pocz,kon:lisc;kier:byte);
var zwp:lisc;
    zap:boolean;
begin
     if kier=1 then - czy parametr kier ma wartość jeden?
     begin - jeśli tak, zaznaczanie odbywa się w prawo, więc…
          if pocz_kur then - czy w momencie rozpoczęcia zaznaczania,...
                                kursor znajdował się za znakiem?
             if pocz^.kur_p<>nil then pocz:=pocz^.kur_p;
                jeśli tak, to gdy istnieje następnik w wiązaniu kursorowym...
                bieżącego elementu, wpisz jego adres do parametru pocz...
          if not stat_kur then - czy w momencie wyłonienia...
                                 nowego elementu zaznaczenia, kursor...
                                 znajduje się przed zaznaczanym znakiem?
             if kon^.kur_l<>nil then kon:=kon^.kur_l;
             jeśli tak, to gdy istnieje poprzednik w wiązaniu kursorowym...
             nowego elementu zaznaczenia, wpisz jego adres do parametru kon
          if pocz=kon then - czy obydwa parametry wskazują na...
                                ten sam element edytora lisc?
             if pocz_kur<>stat_kur then pocz^.zazn:=true
             else pocz^.zazn:=false
             jeśli tak, to gdy zmienne wskazujące na ustawienie kursora...
             w chwili rozpoczęcia zaznaczania i po wyłonieniu nowego...
             elementu zaznaczenia różnią się, ustaw pole zazn w pozycji...
             true składnikowi, na który wskazuje parametr pocz,...
             w przeciwnym przypadku ustaw ten parametr w pozycji false
     end else - jeśli nie, parametr kier ma wartość dwa, co oznacza,...
                że zaznaczanie odbywa się w lewo, więc...
         begin
              if not pocz_kur then - czy w momencie rozpoczęcia...
                                     zaznaczania kursor znajdował się przed znakiem?
                 if pocz^.kur_l<>nil then pocz:=pocz^.kur_l;
                 jeśli tak, to gdy istnieje poprzednik w wiązaniu kursorowym...
                 bieżącego elementu, wpisz jego adres do parametru pocz...
              if stat_kur then - czy w momencie wyłonienia...
                                 nowego elementu zaznaczenia, kursor...
                                 znajduje się za zaznaczanym znakiem?
                 if kon^.kur_p<>nil then kon:=kon^.kur_p;
                 jeśli tak, to gdy istnieje następnik w wiązaniu kursorowym...
                 nowego elementu zaznaczenia, wpisz jego adres...
                 do parametru kon
              if pocz=kon then - (jak wyżej)
                 if pocz_kur<>stat_kur then
                    pocz^.zazn:=false
                 else pocz^.zazn:=true
                (odwrotnie jak w analogicznym przypadku powyżej)
         end;
     zwp:=wsp; - wpisz do zmiennej zwp adres pierwszego elementu listy
     zap:=false; - zainicjuj zmienną logiczną zap wartością false
     while zwp<>nil do - wykonaj krok pętli, póki zmienna zwp...
                         nie otrzyma adresu pustego
     begin
          zwp^.zazn:=false; - ustaw pole zazn elementu, na który...
                              wskazuje zmienna zwp, w pozycji false
          if not zap then - czy zmienna zap ma wartość false?
```

```
                if zwp=kon then - jeśli tak, to czy adres wpisany do...
                 zmiennej zwp i parametr kon wskazują na ten sam element?
                begin - jeśli tak, to…
                     zap:=true; - ustaw zmienną zap w pozycji true
                     kon:=pocz - wpisz do parametru kon adres,...
                                 jaki zawiera parametr pocz
                end else - jeśli nie, to…
                    if zwp=pocz then zap:=true; - jeśli adres...
                            wpisany do zmiennej zwp i parametr pocz...
                            wskazują na ten sam element, ustaw zmienną...
                            zap w pozycji true
            if zap then zwp^.zazn:=true; - jeśli zmienna zap...
            ustawiona jest w pozycji true, ustaw pole zazn w elemencie...
            wskazywanym przez zmienną zwp w pozycji true
            if zap then - czy zmienna zap jest ustawiona w pozycji true?
               if zwp=kon then zap:=false; jeśli tak, to jeśli …
                     zmienna zwp i parametr kon wskazują na ten sam...
                     element, ustaw zmienna zap w pozycji false
            zwp:=zwp^.prawa; - w zmiennej zwp pozyskaj adres...
                               następnika w wiązaniu głównym
        end;
end;
```

XIV. 9.1.2. Całkowite odznaczenie przez kliknięcie

W wyniku kliknięcia myszką w formę w miejscu wstawianych znaków funkcji czy pochodnej, wywołane jest zdarzenie, w wyniku którego uruchomiona zostaje procedura `FormMouseDown`. Jeżeli została wcześniej ustawiona zmienna logiczna `zaz_odz` w pozycji `true`, uruchomiona jest ta część kodu procedury, której zadaniem jest odznaczenie zaznaczonych znaków, czyli ustawienia pola `zazn` w pozycji `false` wszystkim składnikom edytora. By efekt odznaczenia był widoczny, po opuszczeniu procedury `Odznacz`, wywołana zostaje procedura `Wyswietl`.

```
procedure TForm2.FormMouseDown(Sender: TObject;
                               Button: TMouseButton;
                               Shift: TShiftState;
                               X, Y: Integer);
. . .
begin
     . . .
     pocz_kur:=stat_kur; - zapamiętaj w zmiennej prywatnej klasy...
                 TForm2 pozycję kursora w momencie kliknięcia....
                 Jeśli po kliknięciu rozpocznie się zaznaczanie, zmienna...
                 ta będzie pamiętała pozycję kursora w momencie...
                 rozpoczęcia zaznaczania
     nwsp:=Form2.Nowy_biez(X,Y); - znajdź nowy element kliknięcia...
                                   i zapamiętaj jego adres w zmiennej nwsp
     if nwsp<>nil then - czy nowy element został znaleziony?
     begin - jeśli tak, to...
          if zaz_odz then - czy zmienna zaz_odz została ustawiona...
                            w pozycji true?
          begin - jeśli tak, to zaznaczanie jest aktywne, więc...
```

```
                zaz_odz:=false; - ustaw zmienną zaz_odz...
                                  w pozycji false
                zaz_x:=0; - wpisz do zmiennej kierunku zaznaczania zero
                Form2.Odznacz; - ustaw pola zazn wszystkim...
                                 elementom edytora w pozycji false
                Form2.Wyswietl - wywołaj procedurę Wyswietl
        end;
```

XIV. 9.2. Zaznaczanie klawiszami sterowania kursorem

Zaznaczenie lub odznaczenie zaznaczonego znaku możliwe jest także przez naciśnięcie jednego z klawiszy sterowania kursorem. Wywołane jest wówczas zdarzenie w wyniku którego uruchomiona zostaje procedura `FormKeyDown`. Zaznaczenie znaku jest możliwe po spełnieniu dwóch warunków:

- naciśnięcie jednego z klawiszy sterowania kursorem musi być poprzedzone naciśnięciem i przytrzymaniem naciśniętego klawisza `Shift`,
- istniejący, zaznaczany znak nie został jeszcze zaznaczony.

Jeśli znak został wcześniej zaznaczony, jego powtórne zaznaczanie spowoduje odznaczenie tego znaku.

Użycie jednego z klawiszy sterowania kursorem, przy zwolnionym przycisku `Shift`, spowoduje odznaczenie wszystkich zaznaczonych znaków.

Obsługa zaznaczania lub odznaczania tekstu, po użyciu klawisza sterowania kursorem „w lewo” lub „w prawo”, podzielona została na dwie części – oddzielnie dla każdego z wymienionych kierunków zaznaczania. Każda część dotyczy usytuowania kursora z lewej lub prawej strony znaku bieżącego, czyli tego, który wpisany został w pole `znak` elementu wskazywanego przez wskaźnik edytora `wsb`.

Podczas zaznaczania przez użycie klawisza „w lewo”, mając ustawiony kursor za zaznaczanym znakiem lub przez użycie klawisza „w prawo” przy ustawionym kursorze przed tym znakiem, pole `zazn` należące do elementu wskazywanego przez wskaźnik `wsb` zostaje ustawione w pozycji przeciwnej do tej, w jakiej się aktualnie znajduje.

Użycie do zaznaczenia klawisza „w lewo” przy kursorze ustawionym przed znakiem bieżącym, a wiec gdy zmienna `stat_kur` ustawiona jest w pozycji `false`, zaznaczanym lub odznaczanym znakiem jest ten, znajdujący się z lewej strony kursora, a więc pole `zazn` poprzednika elementu bieżącego zostaje zmodyfikowane. Podobnie, gdy do zaznaczenia użytkownik użyje klawisza „w prawo” w sytuacji, gdy kursor znajduje się za znakiem bieżącym, modyfikowane jest wtedy pole `zazn` następnika elementu bieżącego. Poniżej został zaprezentowany sposób zaznaczania lub odznaczania znaku przez użycie klawisza „w lewo”.

```
case Key of
     VK_Left: - gdy naciśnięty został klawisz klawiatury „w lewo”, to…
     if stat_kur then - czy kursor ustawiony jest za znakiem bieżącym?
     begin - jeśli tak, to…
          if ssShift in Shift then - czy naciśnięty został także...
                                     klawisz Shift?
          begin - jeśli tak, to...
               if wsb^.zazn then wsb^.zazn:=false
               else wsb^.zazn:=true; - przestaw pole zazn należące do...
                                       bieżącego elementu
               zaz_odz:=true; - ustaw zmienną zaz_odz w pozycji true
               Form2.Wyswietl - wywołaj procedurę Wyswietl
```

```
        end else - jeśli nie został naciśnięty klawisz Shift, to...
             if zaz_odz then - czy zaznaczenie jest aktywne?
             begin - jeśli tak, to...
                  zaz_odz:=false; - ustaw zmienną zaz_odz w pozycji...
                                    false
                  Form2.Odznacz; - ustaw pola zazn w pozycji false...
                                   we wszystkich składnikach listy lisc
                  Form2.Wyswietl - wywołaj procedurę Wyswietl
             end;
        . . .
end else - jeśli nie, kursor ustawiony jest przed znakiem bieżącym, więc...
    begin
         if ssShift in Shift then - czy naciśnięty został także...
                                    klawisz Shift?
         begin - jeśli tak, to…
              ax1:=wsb^.kur_l; - do zmiennej ax1 wczytaj adres...
                poprzednika bieżącego elementu w wiązaniu kursorowym

              if ax1<>nil then - czy poprzednik istnieje?
              begin - jeśli tak, to…
                   if ax1^.zazn then ax1^.zazn:=false
                   else ax1^.zazn:=true; - przestaw pole zazn...
                                           należące do poprzednika
                   zaz_odz:=true; - ustaw zmienną zaz_odz...
                                    w pozycji true
                   Form2.Wyswietl -wywołaj procedurę Wyswietl
              end
         end else - jeśli nie został naciśnięty klawisz Shift, to...
             if zaz_odz then
             begin
                  zaz_odz:=false;   } (jak wyżej)
                  Form2.Odznacz;
                  Form2.Wyswietl
             end;
```

Zaznaczanie przy użyciu klawisza „w górę” lub „w dół” przebiega identycznie jak to zaprezentowane powyżej, z tą różnicą, że zaznaczanym lub odznaczanym znakiem jest ten umieszczony odpowiednio nad lub pod znakiem bieżącym.

Niezależnie od tego, czy znak został zaznaczony czy odznaczony, zmienna logiczna `zaz_odz` jest ustawiana w pozycji `true`, ustanawiając oraz podtrzymując aktywny stan zaznaczenia.

XIV. 9.3. Usuwanie zaznaczonych znaków

Podczas zaznaczania tekstu zarówno myszką komputerową, jak i klawiszami sterowania kursorem, zaznaczane mogą być także znaki ukryte. W porównaniu z tradycyjnym usuwaniem znaku, jest to ważna informacja – bezwzględnie konieczne stało się dołączenie do procedury `us_pot` dodatkowej instrukcji zabezpieczającej ją przed wykonaniem niedozwolonej operacji na wskaźnikach, w konsekwencji prowadzącej do zawieszenia się programu. Chodzi o obecność we wskaźniku edytora `wsb` adresu składnika z wpisanym w polu `znak` znakiem potęgowania. Poza zaznaczaniem, wskaźnik `wsb` nie może przyjąć adresu takiego elementu, gdyż nie pozwolą na to zabezpieczenia funkcjonujące przy poruszaniu się kursorem. Jeśli

więc podczas usuwania zaznaczonego tekstu wywołana zostanie procedura us_pot (procedura ta jest wewnętrzną procedurą usun_znak, której opis znajduje się w rozdziale XIV.3.3), we wskaźniku edytora wsb wykryta zostanie obecność adresu usuwanego składnika z wpisanym znakiem potęgowania, wskaźnik ten musi przyjąć adres swego poprzednika w wiązaniu głównym. Można mieć pewność, że poprzednik ten istnieje, gdyż nie dopuszczą do jego usunięcia zabezpieczenia spoza procedury.

```
procedure us_pot(const podn5:lisc);
var up1,up2,up3:lisc;
. . .
up3:=podn5^.lewa; - do wskaźnika up3 wczytaj adres poprzednika,...
      w wiązaniu głównym, elementu ze znakiem potęgowania, dostępnego przez...
       parametr podn5

if wsb=podn5 then wsb:=up3; - jeśli adres, zawarty w parametrze podn5...
    i w głównym wskaźniku edytora wsb, jest taki sam, przypisz wskaźnikowi edytora...
    adres jego poprzednika w wiązaniu głównym
```

Pozostałe znaki ukryte usuwane są w sposób tradycyjny, a więc tak jak każdy inny znak nie będący znakiem akcji.

Do usunięcia zaznaczonego tekstu służy klawisz klawiatury DELETE – jego naciśnięcie generuje zdarzenie, w wyniku którego wywołana zostaje procedura FormKeyDown. W jednym z parametrów procedury znajduje się wartość w kodzie ASCII naciśniętego klawisza – jeśli odpowiada on klawiszowi DELETE przy aktywnym zaznaczeniu, uruchamiany jest kod programu usuwający zaznaczone znaki. Usuwanie odbywa się w pętli while..do, w której adres zaznaczonego elementu, a więc takiego, w którym pole zazn ustawione zostało w pozycji true, przekazany jest wskaźnikowi wsb. Tylko w ten sposób można wskazać procedurze usun_znak składnik listy przeznaczony do usunięcia. Nim procedura zostanie wywołana, pole zazn tego składnika musi być przestawione w pozycję false – jest to niezbędne, gdyż może okazać się, że składnik nie zostanie usunięty a jedynie znak z jego pola znak. Pozostawienie pola zazn w pozycji true może więc spowodować cykliczne wywołania procedury usun_znak, po każdorazowym znalezieniu tego samego elementu i w konsekwencji zapętlenie się pętli. Procedura usun_znak zostaje wywołana opcją false i ustawionym kursorze przed znakiem, co daje pewność, że znak zawarty w elemencie bieżącym zostanie usunięty. Zaraz po opuszczeniu procedury, zmienna sterująca pętlą otrzymuje adres pierwszego elementu listy, umożliwiając przeszukiwanie pól zazn od początku. Instrukcja Continue pomija polecenie, w którym do zmiennej sterującej wczytany zostaje adres następnego elementu. Daje to pewność, że pierwszy element listy nie zostanie pominięty. Po opuszczeniu pętli, czyli usunięciu wszystkich zaznaczonych znaków, zmiennej zaz_x przypisana zostaje wartość zero, a zmienna logiczna zaz_odz ustawiona zostaje w pozycji false, wyłączając tym stan zaznaczenia.

```
case Key of
. . .
     VK_Delete: - czy naciśnięty został klawisz DELETE?
     if zaz_odz then - jeśli tak, to czy zmienna zaz_odz ustawiona jest...
                        w pozycji true?

     begin - jeśli tak, to stan zaznaczenia jest aktywny, więc…
          ax1:=wsp; - lokalnej zmiennej ax1 przypisz adres...
                      pierwszego elementu listy

          while ax1<>nil do - wykonaj krok pętli, póki zmienna ax1...
                              nie ma adresu pustego
```

```
        begin
             if ax1^.zazn then - czy w elemencie wskazywanym przez...
                wskaźnik ax1, pole zazn ustawione zostało w pozycji true?
             begin - jeśli tak, to znak jest zaznaczony, czyli...
                  ax1^.zazn:=false; - przestaw pole zazn...
                                      w pozycję false
                  wsb:=ax1; - przypisz zmiennej edytora wsb adres...
                              elementu przeznaczonego do usunięcia
                  stat_kur:=false; - ustaw kursor przed znakiem
                  Form2.Usun_znak(false); - wywołaj procedurę...
                   usun_znak z opcją false (usunięcie znaku za kursorem)
                  ax1:=wsp; - przypisz zmiennej ax1 adres...
                              pierwszego składnika listy
                  Continue - pomiń dalsze instrukcje w pętli
             end;
             ax1:=ax1^.prawa - w zmiennej ax1 pozyskaj adres następnika
        end;
        Form2.Odswiez_funkcje; - wywołaj procedurę Odswiez_funkcje,
        by uwidocznić na formie funkcję pozostałą po usunięciu zaznaczonych znaków
        zaz_x:=0; - wpisz do zmiennej zaz_x wartość zero...
                    (nieokreślony kierunek zaznaczania)
        zaz_odz:=false - ustaw zmienną zaz_odz w pozycji false...
                         (wyłączony stan zaznaczenia)
end else
```

XV. Edytor równań – bez edycji

W tym rozdziale omówiono jedynie różnice między edytorem z edycją i bez edycji, gdyż ich budowa oraz rozwiązania programowe są do siebie bardzo podobne. Różnice jakie między nimi istnieją, mają jednak duże znaczenie, dlatego zostaną wskazane i opisane. Pierwszą z nich jest uproszczenie głównej struktury edytora, polegające na rezygnacji z pól `kur_g` i `kur_d`, gdyż edytor bez edycji nie posiada kursora, więc pola te nie miałyby tu zastosowania. Brak kursora sugerowałby także rezygnację z pól `kur_l` i `kur_p`, jednak pola te potrzebne są do utworzenia otoczenia znaków akcji, czyli takiego sposobu wiązań elementów edytora wokół tych znaków, jakie funkcjonują w edytorze z edycją – bez tych pól byłoby to niemożliwe. Edytor bez edycji nie posiada również procedury `usun_znak`, która w edytorze z edycją umożliwia modyfikację pisanej funkcji czy pochodnej. Skoro nie ma kursora, nie ma też procedur reagujących na zdarzenia wywołane klawiaturą. Pozostałe zmiany znajdują się w wewnętrznych procedurach, które opisano poniżej.

XV. 1. Procedura `dzielenie`

Zarówno w tej procedurze, jak i w każdej następnej, należącej do procedury `Odswiez_funkcje`, możliwa była rezygnacja z wielu zabezpieczeń programowych, które chroniłyby strukturę edytora przed błędnymi wiązaniami jej składników ze sobą. Rezygnację z zabezpieczeń umożliwiło jedynie sekwencyjne wprowadzanie znaków do edytora, bez możliwości ingerencji użytkownika w treść układanej funkcji przez edytor. To pozwoliło na rezygnację z przygotowywania elementów edytora z wpisanym znakiem spacji, które w edytorze z edycją umożliwiało późniejsze dopisanie znaku z klawiatury. Takie uproszczenia doprowadziły do skrócenia kodu procedur, takich jak: `dzielenie`, `pierwiastek` czy `potelog`, mniej więcej o 2/3 w stosunku do tych samych procedur należących do edytora z edycją.

Poniżej zamieszczono fragment kodu zawierający pętlę wydzielającą licznik ułamka. W pętli zliczane są nawiasy – odbywa się to w ten sposób, że rozpoznanie w polu `znak` elementu edytora nawiasu zamykającego, zwiększa wartość w zmiennej `n1` o jeden, natomiast rozpoznanie nawiasu otwierającego – zmniejsza w tej zmiennej wartość o jeden. Opuszczenie pętli następuje, gdy zidentyfikowany zostanie znak działania, czyli dodawania, odejmowania lub mnożenia, przy równości nawiasów otwierających i zamykających lub gdy wskaźnik `ad1`, przyjmujący w pętli adres poprzedniego elementu edytora w wiązaniu głównym, otrzyma adres pusty. W pętli znajdowana jest także największa wartość współrzędnej pionowej, pochodząca z pól `y2` elementów licznika, co pozwoli na wyznaczenie położenia licznika na formie.

```
ad3:=akcja; - do wskaźnika ad3 wpisz adres elementu znaku dzielenia,...
                zawartego w parametrze akcja
ad1:=ad3^.lewa; - do wskaźnika ad1 wczytaj adres poprzednika elementu...
                znaku dzielenia w wiązaniu głównym
if ad1=nil then exit; - jeśli poprzednik nie istnieje, opuść procedurę
ad2:=nil; - wskaźnik ad2 zainicjuj identyfikatorem adresu pustego
l1:=-maxint; - zmienną l1 zainicjuj minimalną wartością dla typu integer
n1:=0; - zmienną n1 zainicjuj wartością zero
```

```
repeat
        k1:=ord(ad1^.znak); - do zmiennej k1 wpisz wartość znaku zawartego...
                              w polu znak elementu analizowanego w pętli

        if k1 in zam then inc(n1); - jeśli wartość zawarta w zmiennej k1...
                                     należy do wartości nawiasów zamykających,...
                                     zawartych w tablicy stałej zam, zwiększ...
                                     wartość w zmiennej n1 o jeden

        if k1 in otw then dec(n1); - jeśli wartość zawarta w zmiennej k1...
                                     należy do wartości nawiasów otwierających,...
                                     zawartych w tablicy stałej otw, zmniejsz...
                                     wartość w zmiennej n1 o jeden

        if n1<=0 then - czy wartość w zmiennej n1 jest mniejsza lub równa zero?
        begin - jeśli tak, to...
             if k1 in dzia then break; - jeśli wartość w zmiennej k1...
                                         należy do wartości znaków działania, zawartych...
                                         w tablicy stałej dzia, to opuść pętlę

             if k1=47 then break; - jeśli wartość w zmiennej k1...
                                    odpowiada znakowi dzielenia, to opuść pętlę

             if n1<0 then break - jeśli wartość w zmiennej n1 jest mniejsza...
                                  od zera, opuść pętlę
        end;
        ad2:=ad1; - we wskaźniku ad2 zapamiętaj adres zawarty we wskaźniku ad1,...
                    nim we wskaźniku ad1 pojawi się nowy adres

        if ad2^.y2>l1 then l1:=ad2^.y2; - jeśli wartość w polu y2...
                                          pewnego elementu licznika, jest większa...
                                          od wartości w zmiennej l1, wpisz do tej...
                                          zmiennej wartość z tego pola

        ad1:=ad1^.lewa - we wskaźniku ad1 pozyskaj adres poprzednika...
                         w wiązaniu głównym

until ad1=nil; - opuść pętlę, gdy wskaźnik ad1 otrzyma adres pusty
```

Na podstawie powyższego kodu można wywnioskować, że pętla wykonuje się przy każdym wywołaniu procedury `dzielenie`, jedynym warunkiem rozpoczęcia jej pracy jest obecność poprzednika w stosunku do elementu znaku dzielenia. Kolejny kod zamieszczony poniżej ustala strukturę otwierającą obecnego ułamka. Jak będzie można się przekonać, sprawdzane jest jedynie jej istnienie. Jeśli element ten został wcześniej utworzony, będzie odpowiednio związany ze składnikiem kreski ułamkowej i pierwszym elementem licznika. Uderzającą różnicą w porównaniu z edytorem z edycją jest brak tworzenia tej struktury niejako na wyrost, co oznacza, że jeśli nie została ona wcześniej utworzona, po prostu jej nie będzie.

```
ad4:=ad3^.lewa; - do wskaźnika ad4 wpisz adres ostatniego elementu licznika
if ad4^.kur_p=ad3 then - czy element znaku dzielenia (ad3) jest związany z ostatnim...
                         elementem licznika wiązaniem kursorowym?

begin - jeśli tak, to licznik nie został jeszcze poprawnie związany...
        z pozostałymi częściami ułamka, więc...

     ad2^.kur_l:=nil; - zamknij pierwszy element licznika z lewej strony
     ad4^.kur_p:=nil; - zamknij ostatni element licznika z prawej strony
     ad3^.lx:=ad2^.lx; - ustaw początkową współrzędną poziomą elementu znaku...
     dzielenia względem początkowej współrzędnej poziomej pierwszego elementu licznika

     ukryj_nawiasy(ad2,ad4); - ukryj skrajną parę nawiasów licznika
```

```
      if ad1<>nil then  czy istnieje element przed ułamkiem?
         if ad1^.ly=ad3^.ly then - jeśli tak, to czy współrzędne poziomu...
                                   tego elementu i znaku dzielenia mają te same wartości?
         begin - jeśli tak, to zwiąż element przed ułamkiem jak strukturę otwierającą
               ad1^.kur_p:=ad3; - wpisz do pola kur_p elementu znajdującego się...
                                  przed ułamkiem adres składnika znaku dzielenia
               ad3^.kur_l:=ad1; - wpisz do pola kur_l elementu znaku dzielenia...
                                  adres składnika znajdującego się przed ułamkiem
               ad3^.lx:=ad1^.px - wpisz do pola lx początkowej współrzędnej...
                                  poziomej elementu znaku dzielenia wartość...
                                  z pola lx składnika znajdującego się przed ułamkiem
         end
end;
apl:=ad2; - zapamiętaj we wskaźniku apl adres pierwszego elementu licznika
```

W podobny sposób ustalany jest mianownik ułamka, pod warunkiem, że istnieje następnik w wiązaniu głównym elementu znaku dzielenia. Obecność następnika uruchamia pętlę, w której prócz wydzielania mianownika spośród pozostałych elementów funkcji, szukana jest najmniejsza wartość współrzędnej pionowej, pochodząca z pól `y1` pewnych elementów mianownika.

```
ad1:=ad3^.prawa; - do wskaźnika ad1 wpisz adres następnika w wiązaniu głównym...
                   elementu znaku dzielenia
if ad1<>nil then - czy istnieje następnik?
begin - jeśli tak, to jest on pierwszym elementem mianownika, więc...
      ad1^.kur_l:=nil; - zamknij go z lewej strony
      ad2:=nil; - zainicjuj wskaźnik ad2 identyfikatorem adresu pustego
      m1:=maxint; - zainicjuj zmienną m1 maksymalną wartością dla typu integer
      n1:=0; - zainicjuj zmienną n1 wartością zero
      repeat
           k1:=ord(ad1^.znak); - do zmiennej k1 wpisz wartość znaku zawartego...
                                 w polu znak elementu analizowanego w pętli
           if k1 in otw then inc(n1); - dla wartości nawiasu otwierającego...
                                        zwiększ wartość w zmiennej n1 o jeden
           if k1 in zam then dec(n1); - dla wartości nawiasu zamykającego...
                                        zmniejsz wartość w zmiennej n1 o jeden
           if n1<=0 then - czy licznik nawiasów (zmienna n1) ma wartość...
                           mniejszą lub równą zero?
           begin - jeśli tak, to...
                if k1 in dzia then break; - jeśli zmienna k1 posiada wartość...
                                            znaku działania ('+','– ','*'), opuść pętlę
                if n1<0 then break - jeśli zmienna n1 ma wartość mniejszą...
                                     od zera, opuść pętlę
           end;
           ad2:=ad1; - we wskaźniku ad2 zapamiętaj adres zawarty we wskaźniku ad1,...
                       nim we wskaźniku tym pojawi się nowy adres
           if ad2^.y1<m1 then m1:=ad2^.y1; - jeśli wartość w polu y1...
                             pewnego elementu mianownika jest mniejsza od wartości...
                             w zmiennej m1, wpisz do tej zmiennej wartość z tego pola
           ad1:=ad1^.prawa - pozyskaj we wskaźniku ad1 adres następnika
      until ad1=nil; - opuść pętlę, gdy wskaźnik ad1 otrzyma adres pusty
```

Jeśli po opuszczeniu powyższej pętli, wskaźnik `ad1` nie będzie miała adresu pustego, oznacza to, że za ułamkiem znajduje się następny element. Jeśli w jego polu `kur_l` znajduje się adres ostatniego elementu mianownika, ważne jest, by znak zawarty w polu `znak` tego elementu był na tym samym poziomie co kreska ułamkowa – ewentualna różnica musi zostać usunięta.

```
if ad1<>nil then - czy wskaźnik ad1 nie ma adresu pustego?
   if ad1^.kur_l=ad2 then - jeśli tak, to czy w polu kur_l elementu wskazywanego...
                            przez ten wskaźnik znajduje się adres ostatniego elementu mianownika?
   begin - jeśli tak, to...
        k1:=ad3^.ly-ad1^.ly; - w zmiennej k1 wylicz różnicę między...
                               górną współrzędną poziomu elementów: kreski ułamkowej...
                               i elementu wskazywanego przez wskaźnik ad1
        n1:=ad3^.py-ad1^.py; - w zmiennej n1 wylicz różnicę między...
                               dolną współrzędną poziomu elementów jak wyżej
        if k1<>0 then - czy istnieje różnica?
        with ad1^ do - jeśli tak, to w instrukcji wiążącej skoryguj współrzędne pionowe...
                       struktury zamykającej
        begin
            ly:=ly+k1;
            py:=py+n1;  } skoryguj pola współrzędnych pionowych struktury zamykającej..
            y1:=y1+k1;  } jako sumę dotychczasowych wartości i wyliczonych różnic
            y2:=y2+n1
        end
   end else
else ad3^.kur_p:=nil; - jeśli wskaźnik ad1 ma adres pusty, zamknij element...
                        znaku dzielenia z prawej strony
```

Analizując powyższy kod, nasuwa się pytanie: dlaczego struktura zamykająca nie została związana z ostatnim składnikiem mianownika i elementem znaku dzielenia? Wiązanie to będzie zrealizowane po ustaleniu długości kreski ułamkowej. Nim to jednak nastąpi, wszystkie składniki licznika i mianownika muszą zostać poprawnie ułożone, a więc właściwie zwymiarowane. Proces ten jest identyczny z tym, jaki zastosowany został w procedurze o tej samej nazwie, należącej do edytora z edycją, dlatego jego opis zostanie tu pominięty.

Gdy kreska ułamkowa posiada już właściwą długość, w jednej instrukcji warunkowej odbywa się wiązanie istniejącej struktury zamykającej z elementem znaku dzielenia oraz skorygowane są współrzędne poziome tej struktury względem współrzędnej końcowej kreski ułamkowej.

```
if ad1<>nil then - czy istnieje struktura zamykająca?
   if ad1^.kur_l=ad2 then - jeśli tak, to czy w jej polu kur_l znajduje się...
                            adres ostatniego elementu mianownika?
   begin - jeśli tak, to...
        ad1^.kur_l:=ad3; - wpisz w pole kur_l struktury zamykającej...
                           adres elementu znaku dzielenia
        ad3^.kur_p:=ad1; - w pole kur_p elementu znaku dzielenia wpisz...
                           adres struktury zamykającej
        n1:=ad3^.px-ad1^.lx; - w zmiennej n1 wylicz różnicę między wartościami...
        współrzędnych poziomych: px elementu znaku dzielenia i lx struktury zamykającej
        ad1^.lx:=ad1^.lx+n1; } skoryguj współrzędne poziome struktury zamykającej...
        ad1^.px:=ad1^.px+n1  } o wyliczoną różnicę w zmiennej n1
   end;
```

Warto zauważyć, że nie ma potrzeby przesuwania znaków zawartych w kolejnych następnikach struktury zamykającej, gdyż nie istnieje jeszcze następnik tej struktury, więc nie ma znaku do przesunięcia. Przy kolejnych wywołaniach procedury nie będzie spełniony drugi warunek, dlatego mimo istnienia już następnika struktury zamykającej, instrukcje zaprezentowane powyżej nie wykonają się.

Procedurze pozostało jeszcze wyśrodkowanie licznika i mianownika względem kreski ułamkowej. By nie powielać opisu tych instrukcji – które znajdują się również w procedurze o tej samej nazwie, ale należącej do edytora z edycją – w tym rozdziale je pominięto.

XV. 2. Procedura `pierwiastek`

Rozpoczęcie pracy procedury uzależnione jest od obecności następnika w wiązaniu głównym elementu znaku pierwiastkowania. Gdy następnik ten istnieje, ale w jego polu `znak` znajduje się cyfra, jest to element stopnia pierwiastka, w przeciwnym przypadku jest to pierwszy element funkcji podpierwiastkowej. Brak elementu stopnia pierwiastka nie będzie w tej procedurze uzupełniony, jak miało to miejsce w procedurze należącej do edytora z edycją. Powód jest ten sam, stopień pierwiastka zawsze jest ustalony jeszcze przed wywołaniem tej procedury, zatem jeśli jest on drugiego stopnia, taki pozostanie.

```
pi2:=adres^.prawa; - wpisz do wskaźnika pi2 adres następnika w wiązaniu głównym...
                     elementu znaku pierwiastkowania
pi3:=adres^.kur_p; - wpisz do wskaźnika pi3 adres następnika w wiązaniu…
                     kursorowym elementu znaku pierwiastkowania
if pi2<>nil then - czy następnik w wiązaniu głównym istnieje?
   if pi2=pi3 then - jeśli tak, to czy następnik w wiązaniu głównym i kursorowym,...
                     to ten sam element?
   begin - jeśli tak, to...
        n1:=ord(pi2^.znak); - do zmiennej n1 wczytaj wartość znaku wpisanego...
                              w pole znak następnika w wiązaniu głównym
        if (n1>48) and (n1<58) then - czy wartość znaku odpowiada cyfrze?
        begin - jeśli tak, to jest to element stopnia pierwiastka, więc...
             pi1:=pi2; - zapamiętaj jego adres we wskaźniku pi1
             pi2:=pi2^.prawa - we wskaźniku pi2 pozyskaj adres następnika...
                               w wiązaniu głównym
        end else pi1:=nil; - w przeciwnym razie, do wskaźnika pi1 wpisz...
                             identyfikator adresu pustego
```

Istnienie elementu funkcji podpierwiastkowej, której adres powinien znajdować się we wskaźniku `pi2`, uruchamia pętlę ustalającą funkcję podpierwiastkową. W pętli, poza zliczaniem nawiasów i opuszczeniem jej, gdy liczba nawiasów zamykających będzie większa od otwierających lub równa obu rodzajom nawiasów, spośród elementów funkcji podpierwiastkowej szukane są: najmniejsza i największa wartość współrzędnej pionowej, które wykorzystane będą do wymiarowania znaku pierwiastka i ułożenia funkcji podpierwiastkowej pod pierwiastkiem.

```
if pi2<>nil then - czy wskaźnik pi2 nie ma adresu pustego?
begin - jeśli nie, to istnieje przynajmniej jeden element funkcji podpierwiastkowej, czyli...
     pi3:=pi2; - zachowaj we wskaźniku pi3 adres pierwszego elementu...
                 funkcji podpierwiastkowej
     pi4:=nil; - wskaźnik pi4 zainicjuj identyfikatorem adresu pustego
```

```
n1:=0; - zmienną n1 (licznik nawiasów) zainicjuj wartością zero
s1:=maxint; - zmienną s1 zainicjuj maksymalną wartością,...
              odpowiednią dla typu integer
s2:=0; - zmienną s2 zainicjuj wartością zero
repeat
      k2:=ord(pi3^.znak); - do zmiennej k2 wczytaj wartość znaku...
         zawartego w polu znak kolejnego elementu analizowanego w pętli
      if k2 in otw then inc(n1); - gdy wartość znaku odpowiada...
          nawiasowi otwierającemu, zwiększ wartość w zmiennej n1 o jeden
      if k2 in zam then dec(n1); - gdy wartość znaku odpowiada...
          nawiasowi zamykającemu, zmniejsz wartość w zmiennej n1 o jeden
      if n1<=0 then - czy wartość w zmiennej n1 jest mniejsza...
                      lub równa zero?
      begin - jeśli tak, to...
           if k2 in dzia then break; - jeśli wartość odczytanego...
                                znaku należy do znaków działania, opuść pętlę
           if n1<0 then break - jeśli zmienna n1 ma wartość...
                                mniejszą od zera, opuść pętlę
      end;
      pi4:=pi3; - we wskaźniku pi4 zapamiętaj adres zawarty we wskaźniku...
                  pi3, nim we wskaźniku tym pojawi się nowy adres
      if pi4^.y1<s1 then s1:=pi4^.y1; - jeśli wartość zawarta...
                      w polu y1 elementu funkcji podpierwiastkowej,...
                      jest mniejsza od wartości w zmiennej s1, wpisz...
                      do tej zmiennej wartość z tego pola
      if pi4^.y2>s2 then s2:=pi4^.y2; - jeśli wartość zawarta...
                      w polu y2 elementu funkcji podpierwiastkowej,...
                      jest większa od wartości w zmiennej s2, wpisz...
                      do tej zmiennej wartość z tego pola
      pi3:=pi3^.prawa - we wskaźniku pi3 pozyskaj adres następnika
until (pi3=nil) or (n1=0); - opuść pętlę, gdy wskaźnik pi3 uzyska...
                             adres pusty lub gdy zmienna n1 ma wartość zero
```

Po opuszczeniu pętli, symbol pierwiastka i wszystkie jego elementy poddane są wymiarowaniu. Sposób wymiarowania jest identyczny z tym, zastosowanym w edytorze z edycją, z tą różnicą, że instrukcje modyfikujące współrzędne elementu stopnia pierwiastka oraz struktury zamykającej muszą być zamknięte w instrukcjach warunkowych, sprawdzających istnienie tych elementów. Bez tych instrukcji, próba wpisania wartości w pole nieistniejącego elementu wywoła błąd systemu operacyjnego.

XV. 3. Procedura `potelog`

Konstrukcja tej procedury jest nieco inna niż jej imienniczki znajdującej się w edytorze z edycją, dlatego opisana zostanie w całości.

W deklaracji stałych znajduje się łańcuch o nazwie `liter5` składający się z wartości znaków reprezentujących litery oraz znaki: dzielenia, pierwiastkowania i potęgowania. Jego obecność ma na celu dostosowanie się do innych możliwości wydzielania funkcji potęgującej lub podstawy logarytmu, jakie oferuje struktura edytora bez edycji. Okrojone możliwości, z jakimi musi poradzić sobie procedura, wynikają z możliwych braków kompletnego otoczenia znaków akcji, które mogą być częścią funkcji potęgującej. Każda procedura wchodząca

w skład edytora z edycją musiała zapewnić komplet wiązań z obowiązkowymi strukturami: otwierającą i zamykającą. Procedury należące do edytora bez edycji nie dają takiej gwarancji, ponieważ struktury te mogą po prostu nie istnieć. Nie można więc budować warunków, których spełnienie lub nie zależy od niepewnych elementów, dlatego sposób wydzielania funkcji potęgującej lub podstawy logarytmu musiał zostać zmieniony.

W początkowych instrukcjach procedury sprawdzany jest drugi, otrzymany parametr logiczny. Jeśli posiada wartość `true`, oznacza to, że w pierwszym parametrze znajduje się adres elementu, który ma wpisany znak potęgowania. Znak ten zostaje ukryty, zaś jego szerokość – zredukowana do zera. Dalsza praca procedury możliwa jest wtedy, gdy istnieje następnik w wiązaniu głównym elementu, na który wskazuje otrzymany parametr. Obecność następnika pozwala na uruchomienie pętli, w której określona zostaje funkcja potęgująca lub podstawa logarytmu. Nim zostanie ona uruchomiona, sprawdzany jest znak zawarty w tym następniku. Jeśli jest to nawias otwierający a procedura zajmuje się logarytmem, zmienna logiczną `pod5` ustawiana zostaje w pozycji `true`, co zabezpieczy argument logarytmu dziesiętnego przed bezprawnym ukryciem nawiasów oraz przed zmniejszeniem go przez procedurę `Skalowanie`.

Pętla tradycyjnie rozpoczyna się od sczytania znaku zawartego w polu `znak` kolejnego elementu edytora analizowanego w pętli i zapamiętania go w zmiennej liczbowej jako wartość w kodzie ASCII. W pętli zliczane są nawiasy w ten sposób, że gdy wartość odczytanego znaku należy do nawiasów otwierających, zwiększana jest wartość w zmiennej `po1` o jeden. Dodatkowo, jeśli istnieje poprzedni element a zmienna logiczna `czy1` ustawiona jest w pozycji `false`, a ponadto, gdy licznik nawiasów będzie równy zero, pętla zostaje opuszczona. Obecność nawiasu otwierającego tuż za cyfrą może sugerować początek argumentu funkcji logarytmicznej, dlatego nawias ten powinien znaleźć się poza jego podstawą. Może się zdarzyć, że funkcja potęgująca będzie się rozpoczynała z dwóch lub większej liczby nawiasów otwierających. Gdyby nie ostatni warunek opuszczenia pętli zależnej od zerowej wartości licznika nawiasów, następny nawias oraz pozostała część funkcji znalazłyby się poza funkcją potęgującą.

Gdy wartość odczytanego znaku należy do nawiasów zamykających, w zmiennej `po1` zostaje zmniejszona wartość o jeden.

Obecność znaku mnożenia lub dzielenia w sytuacji, gdy nie jest on zamknięty w nawiasie, powoduje przerwanie pętli. W ten sposób znaki te znajdą się poza funkcją potęgującą. W przypadku znaku dzielenia, procedura `dzielenie` będzie mogła umieścić kompletną funkcję potęgującą w liczniku ułamka.

Sczytanie nie zamkniętego w nawiasie znaku dodawania lub odejmowania, przy dodatkowym warunki, że poprzedni znak nie jest potęgowaniem, mnożeniem ani dzieleniem, spowoduje opuszczenie pętli. Każdy znak dodawania lub odejmowania nie ujęty w nawiasie, musi znaleźć się poza funkcją potęgującą. Wyjątkiem jest dopuszczenie tych znaków rozpoczynających np. funkcję potęgującą.

Jeśli wartość sczytanego znaku zawiera się w tablicy `liter5`, zmienna `czy1` zostaje ustawiona w pozycji `true`. Ustawienie tej zmiennej w tej pozycji pozwala zachować w funkcji potęgującej funkcję trygonometryczną. W ten sposób, pojawienie się nawiasu otwierającego, rozpoczynającego argument tej funkcji, nie dopuści do przerwania pętli, co pozwoli na zachowanie funkcji trygonometrycznej w całości.

```
procedure potelog(const adres4:lisc;tryb4:boolean);
const liter5=[47,92,94,97,99,103,105,108,110,111,114,115,116];
var pl1,pl2,pl3,pl4:lisc;
    po1,po2,po5:integer;
    czy1,pod5:boolean;
begin
```

```
if tryb4 then - czy parametr tryb4 ustawiony jest w pozycji true?
   if adres4^.druk then - jeśli tak, to czy pole druk, elementu...
       wskazywanego przez pierwszy parametr jest ustawione w pozycji true?
   begin - jeśli tak, to...
        adres4^.druk:=false; - ustaw pole druk tego elementu...
                                w pozycji false
        adres4^.px:=adres4^.lx - wpisz w pole px tego elementu...
           wartość zawartą w polu lx (redukcja szerokości znaku do zera)
    end;
po5:=adres4^.px; - w zmiennej po5 zapamiętaj wartość końcowej...
       współrzędnej poziomej elementu wskazywanego przez parametr
pl4:=adres4^.prawa; - do zmiennej pl4 wczytaj adres następnika...
                      elementu wskazywanego przez parametr
pod5:=false; - ustaw zmienną logiczną pod5 w pozycji false
if pl4<>nil then - czy następnik istnieje?
begin - jeśli tak, to...
     if not tryb4 then - czy ustawienie parametru tryb4 w pozycji...
          false podpowiada procedurze, że ma zająć się logarytmem?
        if ord(pl4^.znak) in otw then pod5:=true; - jeśli tak, to...
```

gdy element wskazywany przez wskaźnik pl4 zawiera w swym polu znak nawias otwierający, ustaw zmienną logiczną pod5 w pozycji true

```
     pl2:=nil; - zainicjuj zmienną pl2 identyfikatorem adresu pustego
     czy1:=false; - ustaw zmienną czy1 w pozycji false
     po1:=0; - zainicjuj zmienną po1 wartością zero
     repeat
        po2:=ord(pl4^.znak); - zapamiętaj w zmiennej po2 wartość...
                               odczytanego znaku
        if po2 in otw then - czy wartość znaku należy do...
                            nawiasów otwierających?
          begin - jeśli tak, to...
           if pl2<>nil then - czy istnieje poprzedni element...
                      funkcji potęgującej lub podstawy logarytmu?
             if not czy1 then - czy zmienna czy1 została...
                                ustawiona w pozycji true?
               if po1=0 then break; - jeśli tak, to jeśli…
                      licznik nawiasów posiada wartość zero,...
                      opuść pętlę
           inc(po1) - zwiększ wartość w zmiennej po1 o jeden
        end;
        if po2 in zam then dec(po1); - jeśli wartość znaku...
                                       należy do nawiasów zamykających,...
                                       zmniejsz wartość w zmiennej po1 o jeden
        if po1<=0 then - czy wartość w zmiennej po1 jest mniejsza...
                         lub równa zero?
        begin - jeśli tak, to...
           if pl2<>nil then - czy istnieje poprzedni element...
                      funkcji potęgującej lub podstawy logarytmu?
```

```
                    begin - jeśli istnieje, to...
                        if (po2=42) or (po2=47) then break;
                              jeśli wartość odczytanego znaku odpowiada...
                              mnożeniu lub dzieleniu, opuść pętlę
                        if (po2=43) or (po2=45) then -czy...
                                 wartość odczytanego znaku należy do...
                                 dodawania lub odejmowania?
                        begin - jeśli tak, to...
                              po1:=ord(pl2^.znak); - odczytaj...
                                wartość znaku w poprzednim elemencie
                              if (po1<>42) and (po1<>47) and
                                  (po1<>94) then break - jeżeli...
                                 w poprzednim elemencie nie ma znaku...
                                 mnożenia, dzielenia czy potęgowania,...
                                 opuść pętlę
                              else po1:=0 - w przeciwnym przypadku...
                                wpisz do zmiennej po1 wartość zero
                        end;
                    end;
                    if po1<0 then break; - jeśli zmienna po1 ma...
                                 wartość mniejszą od zera, opuść pętlę
                end;
                pl2:=pl4; - zachowaj we wskaźniku pl2 adres pewnego elementu...
                          funkcji potęgującej lub podstawy logarytmu
                if po2 in liter5 then czy1:=true; - jeśli wartość...
                          odczytanego znaku zawiera się w tablicy liter5,...
                          ustaw zmienną czy1 w pozycji true
                pl4:=pl4^.prawa - we wskaźniku pl4 pozyskaj adres następnika...
            until pl4=nil; - opuść pętlę, gdy wskaźnik pl4 otrzyma adres pusty
```

Po opuszczeniu pętli, wskaźnik `pl2` powinien wskazywać na ostatni element funkcji potęgującej lub podstawy logarytmu. Jeśli funkcja potęgująca zamknięta jest w parze nawiasów, zostają one ukryte przez procedurę `ukryj_nawiasy`. Ponieważ procedurze tej należy przekazać adresy: pierwszego i ostatniego elementu funkcji, do kompletu brakuje adresu pierwszego elementu funkcji, dlatego do wskaźnika `pl1` wczytany zostaje adres następnika elementu wskazywanego przez parametr, będący pierwszym elementem funkcji. Teraz można już wywołać procedurę, przekazując jej komplet wymaganych adresów.

Gdy wskaźnik `pi4` zawiera adres elementu edytora, oznacza to, że istnieje element, który nie należy do funkcji potęgującej ani do podstawy logarytmu. Jeżeli element ten jest w dalszym ciągu związany wiązaniem kursorowym z ostatnim elementem funkcji potęgującej lub podstawy logarytmu, otrzymuje on wartości wszystkich współrzędnych pionowych od elementu wskazywanego przez pierwszy otrzymany parametr. Ponadto składniki te wiązane są wzajemnie wiązaniem kursorowym, czyli zgodnie z wiązaniem struktury zamykającej.

```
if not pod5 then - czy zmienna pod5 jest wciąż ustawiona w pozycji false?
begin - jeśli tak, to...
     pl1:=adres4^.prawa; - do wskaźnika pl1 wpisz adres pierwszego elementu...
                           funkcji potęgującej
     ukryj_nawiasy(pl1,pl2); - ukryj skrajną parę nawiasów funkcji potęgującej
end;
if pl4<>nil then - czy istnieje struktura zamykająca?
   if pl2^.kur_p=pl4 then - jeśli tak, to czy w polu kur_p ostatniego elementu...
        funkcji potęgującej /podstawy logarytmu widnieje adres struktury zamykającej?
```

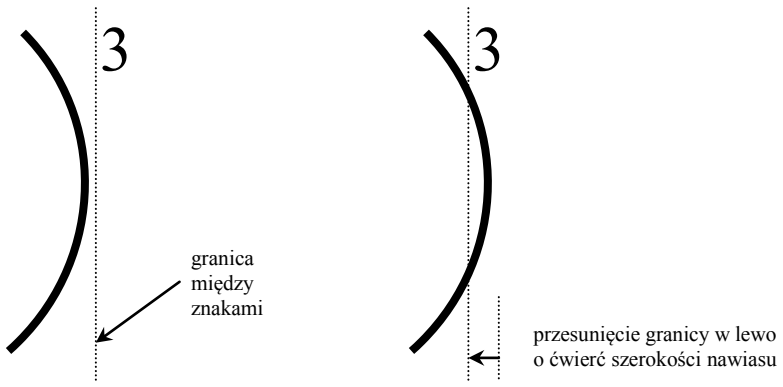
```
begin - jeśli tak, to...
      with pl4^ do
      begin
            ly:=adres4^.ly;  } wpisz do pól współrzędnych pionowych...
            py:=adres4^.py;  } struktury zamykającej wartości współrzędnych...
            y1:=adres4^.y1;  } zawarte w elemencie wskazywanym przez...
            y2:=adres4^.y2   } parametr adres4
      end;
      Form4.Szerokosc(pl4); - popraw szerokość znaku zawartego...
                              w strukturze zamykającej
      adres4^.kur_p:=pl4;  } związ wzajemnie wiązaniem kursorowym...
      pl4^.kur_l:=adres4;  } element adres4 i strukturę zamykającą
end else pl2^.kur_p:=nil - w przeciwnym przypadku, zamknij...
  z prawej strony ostatni element funkcji potęgującej/podstawy logarytmu
```

Przy braku następnika elementu, na który wskazuje otrzymany parametr, procedura musi być opuszczona instrukcją `exit`.

```
pl4:=adres4^.prawa; - do wskaźnika pl4 wczytaj adres następnika...
                      elementu wskazywanego przez parametr adres4
if pl4<>nil then - czy następnik istnieje?
begin
     . . .
end else exit; - jeśli nie istnieje, opuść procedurę
```

Po wydzieleniu funkcji potęgującej czy podstawy logarytmu, pora na ich skalowanie i pozycjonowanie. Proces ten uzależniony jest od ustawienia parametru `tryb4` – pozycja `true` dotyczy funkcji potęgującej.

Na początku sprawdzany jest znak znajdujący się przed znakiem potęgowania. Jeśli znak taki istnieje i jest nim nawias zamykający okrągły lub sześcienny, współrzędna pozioma początku funkcji potęgującej będzie nieznacznie zmniejszona powodując, że funkcja ta będzie przesunięta bliżej tego nawiasu, pozbywając się niepotrzebnej przerwy między zaokrągleniem nawiasu a pierwszym znakiem funkcji.



Rys. 60. Efekt przesunięcia funkcji potęgującej w stronę okrągłego nawiasu zamykającego

Następnie wywołana zostaje procedura `Skalowanie`, która zmniejszy rozmiar znakom zawartym w polach `znak` należących do elementów edytora. Procedura ta otrzymuje w parametrach adres elementu znaku potęgowania i ostatniego elementu funkcji potęgującej.

Po powrocie z procedury, znaki funkcji potęgującej są przesuwane w poziomie o różnicę między wartością początkowej współrzędnej poziomej pierwszego elementu funkcji potęgującej i współrzędną znaku potęgowania. Ponadto, spośród składników tej funkcji, szukana jest największa wartość współrzędnej pionowej. Na jej podstawie oraz aktualnej wysokości znaku znajdującego się przed znakiem potęgowania, wyliczona jest wielkość przesunięcia tej funkcji w pionie.

```
pl3:=adres4^.prawa; - do wskaźnika pl3 wpisz adres pierwszego elementu...
                      funkcji potęgującej/podstawy logarytmu
po1:=0; - zainicjuj zmienną po1 wartością zero
if tryb4 then - czy parametr tryb4 został ustawiony w pozycji true?
begin - jeśli tak, to ustaw i zmniejsz funkcję potęgującą
     pl1:=adres4^.kur_l; - do wskaźnika pl1 wpisz adres poprzednika...
                           elementu znaku potęgowania
     if (pl1^.znak=')') or (pl1^.znak='}') then - czy w polu znak...
               poprzednika znajduje się nawias zamykający okrągły lub sześcienny?
     begin - jeśli tak, to...
          po1:=trunc((pl1^.px-pl1^.lx)/4); - w zmiennej po1 wylicz ¼...
                                szerokości znaku stojącego przed znakiem potęgowania
          po5:=adres4^.lx-po1 - w zmiennej po5 wylicz nową wartość...
                     początkowej współrzędnej poziomej funkcji potęgującej
     end;
     Form4.Skalowanie(adres4,pl2); - zmniejsz znaki funkcji potęgującej
     pl1:=pl3; - we wskaźniku pl1 zachowaj adres pierwszego elementu...
                 funkcji potęgującej
     po1:=pl1^.y1; - zainicjuj zmienną po1 wartością minimalnej współrzędnej...
                     pionowej pierwszego elementu funkcji potęgującej
     po2:=pl1^.lx-po5; - w zmiennej po2 wylicz wielkość przesunięcia...
                         w poziomie znaków funkcji potęgującej
     repeat
          if pl1^.y2>po1 then po1:=pl1^.y2; - jeśli wartość...
          współrzędnej maksymalnej z pola y2 elementu funkcji potęgującej...
          jest większa od wartości w zmiennej po1, wpisz do tej zmiennej...
          wartość z tego pola
          pl1^.lx:=pl1^.lx-po2;  } skoryguj pola współrzędnych poziomych...
          pl1^.px:=pl1^.px-po2;  } o wartość zawartą w zmiennej po2
          pl1:=pl1^.prawa - we wskaźniku pi1 pozyskaj adres następnika
     until (pl1=pl4) or (pl1=nil); - opuść pętlę, gdy wskaźnik pi1...
                        uzyska adres struktury zamykającej (pl4) lub pusty
     pl1:=adres4^.kur_l; - do wskaźnika pl1 wpisz adres poprzednika...
                           elementu znaku potęgowania
     if Form6.QP=0.0 then po1:=0 else - jeśli zmienna globalna QP...
                           modułu regulacji ma wartości zero, to wpisz...
                           do zmiennej po1 wartość zero
     begin - w przeciwnym razie...
          po2:=pl1^.py-pl1^.ly; - w zmiennej po2 wylicz wysokość...
                                  poziomu, na którym znajduje się znak...
                                  stojący przed znakiem potęgowania
```

```
            po2:=pl1^.y1+trunc(po2/Form6.QP); - zmodyfikuj wartość...
                    w zmiennej po2 jako sumę minimalnej współrzędnej pionowej...
                    rzeczywistej znaku przed znakiem potęgowania i ilorazu...
                    wyliczonej wysokości poziomu przez stopień pozycjonowania
            po1:=po1-po2 - w zmiennej po1 wylicz ostateczną wielkość...
                           pozycjonowania funkcji potęgującej
      end
end else
```

Ustawiony parametr `tryb4` w pozycji `false` dotyczy pozycjonowania i skalowania podstawy logarytmu. Sposób jest porównywalny z tym opisanym powyżej, z tą różnicą, że w pętli szukana jest najmniejsza wartość współrzędnej pionowej pochodzącej z pól `y1` elementów podstawy logarytmu, natomiast wielkość przesunięcia podstawy w pionie jest wyliczana względem maksymalnej współrzędnej pionowej struktury otwierającej, której adres znajduje się w parametrze `adres4`. Samo skalowanie i pozycjonowanie podstawy logarytmu uzależnione jest od tego, czy zmienna logiczna `pod5` nie została ustawiona w pozycji `true`. Gdyby tak się stało, otrzymany logarytm nie posiada podstawy. W takiej sytuacji, następnik struktury otwierającej, dostępny przez wskaźnik `pl3`, powinien posiadać w polu `znak` nawias otwierający. Gdyby nie to zabezpieczenie, nawias ten oraz pozostała część argumentu funkcji uległyby skalowaniu i pozycjonowaniu.

```
if tryb4 then - czy parametr tryb4 został ustawiony w pozycji true?
begin
     . . .
end else - jeśli nie, ustawiony jest w pozycji false, czyli...
    if not pod5 then - czy zmienna pod5 jest wciąż ustawiona w pozycji false?
    begin - jeśli tak, to...
          Form4.Skalowanie(adres4,pl2); - zmniejsz znaki podstawy logarytmu
          pl1:=pl3; - we wskaźniku pl1 zachowaj adres pierwszego elementu...
                      podstawy logarytmu
          po1:=adres4^.y2; - zainicjuj zmienną po1 wartością maksymalnej...
                             współrzędnej pionowej struktury otwierającej logarytm
          repeat
               if pl1^.y1<po1 then po1:=pl1^.y1; - jeśli wartość...
               współrzędnej minimalnej z pola y1 elementu podstawy logarytmu...
               jest mniejsza od wartości w zmiennej po1, wpisz do tej zmiennej...
               wartość z tego pola
               pl1:=pl1^.prawa - we wskaźniku pl1 pozyskaj adres następnika
          until (pl1=pl4) or (pl1=nil); - opuść pętlę, gdy wskaźnik pi1...
                                          uzyska adres struktury zamykającej (pl4) lub pusty
          if pl1<>nil then - czy istnieje element argumentu funkcji?
          begin - jeśli tak, to wylicz wielkość przesunięcia podstawy logarytmu
               po2:=adres4^.py-adres4^.ly;
               po2:=adres4^.y2-trunc(po2/Form6.QP);
               po1:=po1-po2
          end else po1:=0
    end;
```

Wartość w zmiennej `po1` różna od zera oznacza konieczność pozycjonowania, czyli poprawiania współrzędnych pionowych elementów podstawy logarytmu lub funkcji potęgującej.

```
if po1<>0 then - czy wartość w zmiennej po1 jest różna od zera?
begin - jeśli tak, to...
     pl1:=pl3; - zachowaj we wskaźniku pl1 adres...
                 pierwszego elementu funkcji potęgującej/podstawy logarytmu
     repeat
           with pl1^ do
           begin
                y1:=y1-po1;  } wpisz w pola współrzędnych pionowych...
                y2:=y2-po1;  } poprawianych elementów...
                ly:=ly-po1;  } różnicę dotychczasowej wartości w tych...
                py:=py-po1   } polach i wartości w zmiennej po1
           end;
           pl1:=pl1^.prawa; - we wskaźniku pl1 pozyskaj adres następnika
     until (pl1=pl4) or (pl1=nil); - opuść pętlę, gdy wskaźnik pi1...
                      uzyska adres struktury zamykającej (pl4) lub pusty
end;
```

Jeśli struktura zamykająca istnieje, o czym świadczy niepusty adres we wskaźniku `pl4`, znak w niej zawarty musi mieć swoją początkową współrzędną poziomą równą końcowej współrzędnej ostatniego znaku, należącego do funkcji potęgującej lub podstawy logarytmu. Wartością tej współrzędnej jest suma początkowej współrzędnej poziomej pierwszego elementu funkcji lub podstawy oraz ich długości. Początkowa współrzędna tych elementów została już zapamiętana w zmiennej `po5`, dlatego jej wartość posłużyła do wyliczenia wymaganej wartości. Ewentualna różnica między tą sumą a wartością w polu `lx` struktury zamykającej posłuży do skorygowanie współrzędnych poziomych zarówno tej struktury, jak i następnych elementów.

```
if pl4<>nil then - czy wskaźnik pl4 nie ma adresu pustego?
   if pl4^.kur_l=adres4 then - jeśli nie, to czy w polu kur_l elementu,...
        na który wskazuje ten wskaźnik, znajduje się adres struktury otwierającej?
   begin - jeśli tak, to element ten jest strukturą zamykającą, więc...
        po1:=po5+dlugosc(pl3,pl4); - w zmiennej po1 wylicz poprawną...
                        wartość, początkowej współrzędnej poziomej znaku...
        po1:=po1-pl4^.lx; w zmiennej po1 wylicz różnicę między wartością...
             współrzędnej, zawartej w polu lx tej struktury a poprawną wartością
        if po1<>0 then - czy wartość w zmiennej po1 jest różna od zera?
        begin - jeśli tak, to zmodyfikuj współrzędne poziome struktury zamykającej...
             po2:=pl4^.px-pl4^.lx; - w zmiennej po2 wylicz szerokość znaku
             pl4^.lx:=pl4^.lx+po1; - w pole lx wpisz sumę...
            dotychczasowej wartości w tym polu i wyliczonej różnicy
             pl4^.px:=pl4^.lx+po2; - w pole px wpisz sumę poprawionej...
                                     wartości w polu lx i szerokości znaku
             Form4.Przesun(pl4,true) - przesuń w prawo pozostałe znaki
        end;
   end;
```

XV. 4. Procedura `Wstaw_znak`

W edytorze bez edycji, jedynie w tej procedurze lokowane są w pamięci kolejne elementy edytora. Każdy z tych elementów wstawiany jest zawsze na końcu listy `lisc`, dlatego nie ma w procedurze gałęzi instrukcji, umożliwiających wstawianie elementu w dowolne miejsce listy, jak miało to miejsce w tej samej procedurze należącej do edytora z edycją. Zre-

zygnowano również z zabezpieczania struktury przed rozbiciem otoczenia znaku akcji, gdyż takie zabezpieczenia w edytorze bez edycji są niepotrzebne. Jedynie przed ulokowaniem w pamięci nowego składnika listy, sprawdzane jest istnienie poprzednika, jeśli istnieje, poprzednik ten jest kompletnie wiązany z nowym elementem. Edytor bez edycji nie posiada funkcji `czy_log`, gdyż wobec sekwencyjnego wprowadzania znaków, nie istnieje niebezpieczeństwo rozbicia nazwy funkcji 'log'.

Choć procedura ta jest bardzo uproszczoną wersją procedury z edytora z edycją, to zastosowane w niej rozwiązania są w zupełności wystarczające do tego, by edytor działał poprawnie.

```
procedure TForm4.Wstaw_znak (const wzn1:char);
var z1:lisc;
begin
     z1:=wsb; - wpisz do wskaźnika z1 adres ostatniego elementu listy
     wsb:=nil; - zainicjuj główny wskaźnik edytora adresem pustym
     new(wsb); - utwórz w pamięci nowy element lisc
     if z1<>nil then - czy istnieje poprzedni element?
     begin - jeśli tak, to...
          with wsb^ do
          begin
               prawa:=nil; - następnik w wiązaniu głównym nie istnieje,...
                dlatego pole prawa zainicjuj identyfikatorem adresu pustego
               lewa:=z1; - w pole lewa wpisz adres poprzednika...
                           w wiązaniu głównym
               kur_p:=nil; - następnik w wiązaniu kursorowym nie istnieje,...
                  dlatego pole kur_p zainicjuj identyfikatorem adresu pustego
               kur_l:=z1; - w pole kur_l wpisz adres poprzednika...
                            w wiązaniu kursorowym
               lx:=z1^.px;
               ly:=z1^.ly;   wpisz w pola współrzędnych poziomych...
               py:=z1^.py;   i pionowych wartości zawarte w polach ...
               y1:=ly;       współrzędnych poprzedniego elementu
               y2:=py;
               znak:=wzn1 - w pole znak wpisz znak zawarty w parametrze
          end;
          Form4.Szerokosc(wsb); - wylicz szerokość znaku i na...
                   jego podstawie uzupełnij pole px nowego elementu..
                   wartością współrzędnej poziomej
          z1^.prawa:=wsb; } wpisz w pola: prawa i kur_p poprzednika...
          z1^.kur_p:=wsb; } adres nowego elementu
     end else - jeśli nie, nowy element jest pierwszym składnikiem listy, więc...
          begin
               with wsb^ do
               begin
                    lewa:=nil;  } zainicjuj wszystkie pola przeznaczone do...
                    prawa:=nil; } wiązań z elementami listy...
                    kur_l:=nil; } identyfikatorami adresu pustego
                    kur_p:=nil;
                    lx:=AX; - w pole lx wpisz wartość poziomej...
                              współrzędnej startowej ze zmiennej AX
                    ly:=AY; - w pole ly wpisz wartość pionowej...
                              współrzędnej startowej ze zmiennej AY
```

```
                    py:=ly+Form6.RZ; - w pole py wpisz sumę wartości...
                                       z pola ly i zmiennej globalnej RZ modułu regulacji,...
                                       w której zawarta jest wysokość czcionki
                    y1:=ly;
                    y2:=py;
                    znak:=wzn1
                end;
                Form4.Szerokosc(wsb);
                wsp:=wsb; - jest to pierwszy składnik listy, więc...
                            jego adres musi zostać zapamiętany przez wskaźnik edytora wsp
            end;
        wsb^.druk:=true; - znak ma być widoczny na formie, dlatego...
                           ustaw pole druk w pozycji true
        Form4.Odswiez_funkcje - ustaw znaki funkcji w sposób matematyczny...
                                i umieść funkcję na formie
end;
```

XVI. Źródła

1. Leksiński W., Macukow B., Żakowski W., *Matematyka dla maturzystów – definicje, twierdzenia, wzory, przykłady*, Wydawnictwo Naukowo Techniczne, Warszawa 1994.
2. Marciniak A., *Turbo Pascal 7.0 z elementami programowania*. Część 1, PWN, Warszawa 2000, PWN 2000,
3. Pamuła T., *Aplikacje w Delphi*, wydanie II , Wydawnictwo „Helion", Gliwice 2007.
4. Siewierski L., *Ćwiczenia z analizy matematycznej z zastosowaniami*. Tom 1, PWN, Warszawa 1982.
5. Snarska A., *Ćwiczenia z Delphi 3.0, 4.0, 5.0*, Wydawnictwo „Mikom", Warszawa 2000.
6. https://4programmers.net.Delphi
7. http://pl.123rf.com
8. http://staff.amu.edu.pl/~zcht/pliki/htmlhelp.rtf

Mgr inż. Sławomir Guzewski – informatyk, specjalizujący się w informatyce stosowanej i sieciach komputerowych. Absolwent Wydziału Inżynierii Mechanicznej i Informatyki Politechniki Częstochowskiej.

Z recenzji:

„Program (…) został napisany w języku Object Pascal z wykorzystaniem narzędzia programistycznego Delphi4, które służy do tworzenia aplikacji uruchomianych w środowisku Windows. (…) Autor bardzo szczegółowo opisuje działanie programu i krok po kroku pokazuje, jak program działa w praktyce. (…) W książce zamieszczono i bardzo szczegółowo omówiono ogromną ilość przykładów, za pomocą których Czytelnik zgłębia praktyczne działanie programu edukacyjnego. (…) Oprócz nauczania obliczania pochodnych funkcji, program umożliwia również obliczanie wartości pochodnych we wskazanych konkretnych punktach. (…) Dodatkową (i bardzo ważną) zaletą prezentowanej książki jest przedstawienie detali działania programu na licznych rysunkach oraz diagramach. Są one bardzo pomocne w zrozumieniu działania programu (…)" .

Prof. zw. dr hab. Józef Banaś
Politechnika Rzeszowska im. Ignacego Łukasiewicza

ISBN 978-83-66187-14-6 (oprawa miękka)
ISBN 978-83-66187-15-3 (PDF)
ISBN 978-83-66187-16-0 (CD)