D. Powtórz punkt B (jako aktualny stan podaj np. 1000). Po zatrzymaniu programu spróbuj zmienić wartość zmiennej *Stan Aktualny* poleceniem Evaluate/Modify z okna Debug (np. na 1500) i wykonaj program do końca. Sprawdź, której wartości zmiennej odpowiada wynik końcowy: wprowadzonej czy zmienionej. Możesz też umieścić wcześniej zmienną w oknie Watches i sprawdzić, czy jej wartość się zmieni.

10.5. Zmienne i stałe

10.5.1. Nazwy zmiennych

Nazwa w systemie Turbo Pascal (patrz rozdz. 10.2.1) może się składać z 63 znaków: liter, cyfr i znaku podkreślenia, przy czym pierwszy znak musi być literą. Zmienne nie mogą nosić nazw używanych dla słów kluczowych, natomiast można im nadawać takie nazwy, jakie są przewidziane dla funkcji i procedur (w tym przypadku program tłumaczący będzie daną nazwę kojarzył wyłącznie z zadeklarowaną zmienną, a już nie z funkcją ani z procedurą). Do poprawnych nazw zmiennych w Turbo Pascalu należą zatem:

```
ala   J23   x1Y2   SIN   WRITELN   czytelnik_wiek
```

chociaż SIN jest nazwą funkcji obliczającej wartość funkcji sinus, a WRITELN nazwą procedury wyświetlającej na ekranie teksty i liczby, natomiast nazwy:

```
1ala  x1-Y2  begin
```

nie są poprawne, pierwsza z nich zaczyna się bowiem od cyfry, druga zawiera niedozwolony w nazwie znak *minus*, a trzecia jest słowem kluczowym BEGIN.

Uwaga. Ponieważ może sprawiać Ci trudność rozróżnianie, które ze słów używanych w programach jest słowem kluczowym, a które nazwą procedury, to przyjmij zasadę, że unikasz tworzenia takich nazw, o których wiesz, że są używane w systemie Turbo Pascal. Nie musisz jednak obawiać się, że spowodujesz błąd używając nieświadomie nazwy jakiejś funkcji.

10.5.2. Typy zmiennych

W języku Pascal wszystkie zmienne muszą być deklarowane i dla każdej z nich musi być podany typ.

W programach przykładowych stosowałeś trzy typy: całkowity (*integer*), rzeczywisty (*real*) i łańcuchowy (*string*). Liczba typów zmiennych przewi-

dzianych w systemie Turbo Pascal jest znacznie większa. Niektóre typy zmiennych liczbowych są wymienione w tab. 10.1, a innych w tab. 10.2.

Tabela 10.1. Podstawowe typy zmiennych liczbowych

Typ zmiennej		Bajty	Wartość minimalna	Wartość maksymalna
Integer	Całkowita	2	-32768	32767
Longint	Całkowita długa	4	-214783648	2147483647
Shortint	Całkowita krótka	1	-128	127
Word	Słowo	2	0	65535
Byte	Bajt	1	0	255
Real	Rzeczywista	4	$-3.402 * 10^{38}$	$3.402 * 10^{38}$
Double	Rzeczywista długa podwójnej precyzji	8	$-1.798 * 10^{308}$	$1.798 * 10^{308}$
Extended	Rozszerzona	10	$-1.1 * 10^{4932}$	$1.1 * 10^{4932}$
Complex	Zespolona	8	$-9.2 * 10^{18}$	$9.2 * 10^{18}$

Tabela 10.2. Podstawowe typy zmiennych nieliczbowych prostych

Typ zmiennej		Bajty	Wartości
Boolean	Logiczna	1	True (Prawda); False (Fałsz)
String	Łańcuch	zmienna	0 ÷ 255 znaków ASCII
Char	Znak	1	znak ASCII

Ponadto użytkownik może tworzyć własne typy, co dodatkowo zwiększa możliwości języka.

Zmienne służą przede wszystkim do przedstawiania liczb (zmienne liczbowe) i tekstów (zmienne łańcuchowe). Oprócz pojedynczych zmiennych mogą być też całe grupy zmiennych, o określonej strukturze. Typy zmiennych dzielimy w związku z tym na **proste**, odpowiadające pojedynczym zmiennym, i **złożone**.

Zmienne proste dzielą się na porządkowe, rzeczywiste, łańcuchowe i wskaźnikowe.

Pojęcie typu porządkowego, a raczej grupy typów, wiąże się z możliwością wskazania dla każdego elementu — elementu następnego lub poprzedniego, lub obu. Zauważ, że można to uczynić dla liczb całkowitych (na przykład dla 13 następnym elementem jest 14, a poprzednim 12), można również dla liter, natomiast nie można dla liczb rzeczywistych.

Typy porządkowe dzielą się na standardowe i niestandardowe. Do standardowych należą liczby całkowite, których jest pięć typów: *shortint, integer, longint, byte, word*, zmienne logiczne (*boolean*) i znakowe (*char*); wszystkie te typy są wymienione w tab. 10.1 i 10.2.

Do niestandardowych typów porządkowych należą typy wyliczeniowe i okrojone. Na typach wyliczeniowych nie wykonuje się operacji arytmetycznych. Przykładem typu wyliczeniowego są nazwy stron świata lub kolory.

Typ okrojony ogranicza zakres zmienności zmiennej. Na przykład wartość zmiennej całkowitej mającej oznaczać dni miesiąca można ograniczyć do przedziału od 1 do 31.

Do podstawowych typów rzeczywistych należą *real* i *double*, oznaczające liczbę rzeczywistą pojedynczej i podwójnej precyzji (oba wymienione w tabeli 10.1).

Zmienne łańcuchowe zawierają ciągi (łańcuchy) znaków: liter, cyfr, znaków przestankowych, znaków spacji, znaków specjalnych ($, %, & i in.). Maksymalna długość łańcucha wynosi 255 znaków.

Typ wskaźnikowy służy do wskazywania miejsca pamięci (tzw. adresu), gdzie jest umieszczona konkretna zmienna.

Do zmiennych złożonych należą tablice (*array*), rekordy (*record*), takie jak w bazach danych, pliki (*file*), zbiory (*set*) oraz wskaźniki (*pointer*). Tablice zawierają elementy jednego typu, natomiast rekordy mogą zawierać elementy różnych typów.

10.5.3. Deklarowanie zmiennych prostych

Z deklarowaniem zmiennych prostych zetknąłeś się już w przykładowych programach. Deklaracje zmiennych rozpoczyna się słowem kluczowym VAR. Deklaracja zmiennych jednego typu określonego w języku Turbo Pascal, albo zdefiniowanego przez użytkownika, ma postać:

```
lista_nazw_zmiennych: typ
```

przy czym nazwy zmiennych występujących w liście są od siebie oddzielone przecinkami, a dodatkowo mogą być oddzielone spacjami. Podczas wprowadzania większej liczby zmiennych można powtarzać deklaracje zmiennych tego samego typu, np.

```
var i,j, k1,k2,k3:     integer;
    x, y1,y2,y3, z:    real;
    licznik:           integer;
    dokladnosc:        real;
    imie, nazwisko:    string;
```

Wybierając typ zmiennych, których zamierzasz używać w swoim programie, kieruj się wiadomościami i wskazówkami z rozdz. 8.5.2.

10.5.4. Nadawanie wartości zmiennym

Zmiennym możesz w programie nadawać wartości. W arkuszu kalkulacyjnym czyniłeś to wpisując liczby do konkretnej komórki, tu natomiast posłużysz się instrukcją przypisania zapisywaną przy użyciu symbolu skła-

dającego się z dwukropka i znaku równości (:=). Na przykład

```
x := 2.5
```

jest instrukcją nadania zmiennej x wartości liczbowej 2.5, natomiast

```
napis := 'Witam Ciebie ponownie!'
```

jest instrukcją nadania zmiennej *napis* wartości w postaci ciągu 22 znaków objętego apostrofami (od wielkiej litery W do wykrzyknika, łącznie ze spacjami).

Operację przypisania wartości należy rozumieć następująco: zmiennej występującej po lewej stronie symbolu przypisania zostaje nadana nowa wartość obliczona z wyrażenia umieszczonego po prawej stronie symbolu (patrz też rozdz. 8.4). Możliwość wykonania operacji przypisania zależy oczywiście od typu zmiennej występującej po lewej stronie symbolu := (gdyby np. zmienna x była zadeklarowana jako *string*, to podana operacja przypisania nie mogłaby być wykonana).

Wartość wyrażenia zapisanego po prawej stronie symbolu jest obliczana przed wykonaniem operacji przypisania, zatem zapis

```
x := x + 5
```

oznacza zwiększenie wartości zmiennej x o 5 (najpierw będzie obliczona suma dotychczasowej wartości zmiennej x oraz liczby 5 i jej wartość zostanie przypisana zmiennej x jako jej nowa wartość.

Operacja przypisywania wartości zmiennej może być wykonywana wielokrotnie (podobnie jak w arkuszu kalkulacyjnym, gdzie do tej samej komórki tablicy można było wpisywać kolejno różne liczby). Wartość przypisana zmiennej pozostaje niezmieniona do chwili wykonania następnej operacji przypisania.

Uwaga. W chwili uruchomienia programu zmienne mogą mieć wartości przypadkowe. Przyjmij zatem zasadę, że każdej zmiennej nadajesz najpierw wartość, zanim użyjesz jej w obliczeniach.

Ćwiczenie 10-6

A. Napisz program TP-4 lub odczytaj go z pliku TP04.PAS:

Program TP-4

```
program TP04;
  {tu jest miejsce na Twoj komentarz}
var   LiczbaCalk:  integer;
begin
  LiczbaCalk := 16;
```

```
  WriteLn ('Liczba = ',LiczbaCalk);
  LiczbaCalk := 4;
  LiczbaCalk := 7;
  WriteLn ('Liczba = ',LiczbaCalk);
  LiczbaCalk := LiczbaCalk + 1;
  WriteLn ('Liczba = ',LiczbaCalk);
  ReadLn
end.
```

Jeżeli program pisałeś, to zapisz go na dysku.

B. Przeanalizuj, co powinno pojawić się na ekranie jako wynik działania programu.
C. Wykonaj program. Czy wynik jest zgodny z przewidywaniem?

Nadając zmiennym wartości, powinieneś pamiętać o ich typie. Nie przypisuj zatem zmiennej liczbowej całkowitej *LiczbaCalk* wartości rzeczywistej 2.5 ani tym bardziej łańcucha znaków. Jeżeli w programie TP-4 zmieniłbyś prawą stronę instrukcji przypisania na liczbę rzeczywistą 2.5 lub łańcuch '44', to system podczas kompilacji programu zasygnalizuje niezgodność typów komunikatem:

```
Error 26:  Type mismatch.
```

Jeżeli przypisywana zmiennej liczba przekracza dopuszczalny zakres zmiennej danego typu, to błąd taki jest wykrywany podczas kompilacji. Gdy zmienisz w programie TP-4 przypisywaną zmiennej *LiczbaCalk* wartość na 40000,

```
LiczbaCalk := 40000
```

powinieneś otrzymać komunikat:

```
Error 76:  Constant out of range
```

oznaczający, że stała (*constant*), czyli liczba 40000, jest spoza zakresu.

Uwaga. Zakres zmiennych może zostać przekroczony podczas wykonywania programu **bez ostrzeżenia** (powodując błędy).

Ćwiczenie 10-7

Zmodyfikuj program TP-4: zamiast 16 przypisz zmiennej *LiczbaCalk* 25000, usuń instrukcję przypisania; jej liczby 4, a obie pozostałe instrukcje przypisania zmodyfikuj tak, aby wartość zmiennej była zwiększana o 20000 (tzn. użyj instrukcji w postaci $x := x + 20000$). Zapisz go pod nazwą TP-4M (właściwie TP04M). Jakie wyniki powinieneś otrzymać: czy 25000, 45000 i 65000? Jakie wyniki otrzymujesz?

Czy widzisz jakiś związek otrzymywanych wyników z zakresem liczb całkowitych?

10.5.5. Używanie stałych

Zapisywane w treści programu teksty (w apostrofach) i liczby system traktuje jako stałe. Używałeś już stałych tekstowych jako parametrów procedur WRITE i WRITELN (np. w programie TP-3). Zapisywałeś też liczby (stałe liczbowe) po prawej stronie symbolu przypisania.

Niekiedy jest wygodnie nadawać stałym nazwy. Gdybyś miał w kilku miejscach programu powtarzać wielocyfrową wartość jakiejś stałej fizycznej lub matematycznej, to byłoby to okazją do błędów (poza tym byłoby uciążliwe). Wartość tę można by wprawdzie nadać także zmiennej, i w wyrażeniach używać tej zmiennej, jednak wartość zmiennej można zmienić przez przeoczenie (lub uczynić ją niedostępną), a wartość stałej nie.

Stałe definiuje się przed określeniem typów zmiennych. Służy do tego słowo kluczowe CONST, po którym podaje się definicje stałych, używając znaku równości (a nie symbolu podstawienia). Program sam rozpoznaje typ stałych. W definicji stałych można używać stałych zdefiniowanych wcześniej.

W systemie Turbo Pascal istnieje funkcja PI, której wartością jest π. Używa się jej w obliczeniach w taki sposób, jak gdyby była to zdefiniowana wcześniej stała.

Program TP-5

```
program TP05;
  {Obliczanie obwodu kola}
const
   DwaPi = 2 * Pi;
var
   promien:   real;
begin
  Write ('Podaj promien kola ');
  ReadLn (promien);
  Write ('Obwod kola o promieniu ', promien :7:2);
  WriteLn (' wynosi: ', DwaPi * promien :11:5);
  ReadLn
end.
```

Stałym (czyli nazwom występującym w deklaracji stałych) nie możesz nadawać wartości w programie. Spróbuj przekonać się o tym dopisując przed instrukcją wywołania procedury WRITELN instrukcję przypisania: `DwaPi:= 2*Pi;` (lub `DwaPi:=6.28;`) i przeprowadzając kompilację; programu nie zapisuj.

Uwaga. Symbole: „:7:2" i „:11:5" występujące jako parametry procedur WRITE i WRITELN określają format zapisu liczb rzeczywistych: liczba za pierwszym dwukropkiem określa liczbę pozycji przeznaczonych na zapis liczby w postaci ułamka dziesiętnego (łącznie z kropką dziesiętną i znakiem minus), a liczba za drugim dwukropkiem określa liczbę miejsc po kropce dziesiętnej. Symbole takie umieszcza się bezpośrednio za wyrażeniem. Zapis liczby jest wówczas stałopozycyjny (stała jest pozycja kropki). Gdy wyrażenie ma wartość całkowitą, wówczas do określenia liczby pozycji używa się jednej liczby całkowitej poprzedzonej dwukropkiem (np. :5).

Jeżeli chcesz się przekonać o roli wymienionych symboli, to usuń je z programu i uruchom go ponownie. Jak są teraz drukowane liczby?

10.6. Działania na zmiennych

10.6.1. Operacje arytmetyczne

Na zmiennych i stałych liczbowych możesz wykonywać cztery działania arytmetyczne, podobnie jak w arkuszu kalkulacyjnym, nie ma natomiast potęgowania. Podnoszenie do potęgi całkowitej możesz zrealizować za pomocą wielokrotnego mnożenia lub dzielenia (podnoszenie do potęgi niecałkowitej wymaga użycia funkcji logarytmicznej i wykładniczej). Są też w języku Pascal zdefiniowane operatory DIV i MOD dotyczące dzielenia całkowitego (tab. 10.3).

Tabela 10.3. Działania na zmiennych liczbowych

Działanie	Symbol	Przykład	Wynik	Typ wyniku w przykładzie
Dodawanie	+	3 + 5.5	8.5	rzeczywisty
		3 + 5	8	całkowity
Odejmowanie	-	17 - 12	5	całkowity
Mnożenie	*	3 * 17	51	całkowity
		30 * 1.7	51.0	rzeczywisty
Dzielenie rzeczywiste	/	9/4	2.25	rzeczywisty
Dzielenie całkowite	DIV	11 DIV 4	2	całkowity
Reszta z dzielenia całkowitego	MOD	11 MOD 4	3	całkowity

Dla przypomnienia: w dzieleniu całkowitym dzieli się liczbę całkowitą przez liczbę całkowitą. Wynik dzielenia jest zawsze liczbą całkowitą — oznacza ona, ile razy dzielnik mieści się w całości w dzielnej; ponadto z dzielenia może pozostać reszta. Na przykład przy dzieleniu 13 przez 5 wynik dzielenia całkowitego (DIV) wynosi 2, a z dzielenia pozostaje reszta (MOD) równa 3.

W celu oswojenia się z zapisywaniem rzadziej używanych działań wykonaj ćwiczenie 10-8.

Ćwiczenie 10-8

A. Napisz program TP-6 lub odczytaj go z pliku TP06.PAS.

Program TP-6

```
program TP06;
  {operacje arytmetyczne}
begin
  WriteLn ('17 / 5   = ', 17 / 5 :7:5);
  Write ('17 DIV 5 = ', 17 DIV 5);
  WriteLn ('         17 MOD 5 = ', 17 MOD 5);
  Write ('2 do potegi 3 = ', 2*2*2);
  WriteLn('    3 do potegi 2 = ', 3*3);
  ReadLn
end.
```

B. Oblicz wyniki, jakie powinieneś otrzymać (jeżeli dobrze rozumiesz przedstawiony zapis).
C. Uruchom program i sprawdź, czy otrzymane wyniki są zgodne z Twoimi.
D. Popraw format zapisu wyniku dzielenia, jeżeli Ci nie odpowiada.

Uwaga. Typ wyniku podany w tab. 10.3 odnosi się do liczb podanych w przykładach, przy założeniu że ich typ odpowiada zapisowi. W istocie typ wyniku takich operacji jak dodawanie, odejmowanie czy mnożenie zależy od typu zmiennych i stałych będących elementami działania. Na przykład, jeżeli wszystkie elementy (zmienne lub stałe) są typu całkowitego, to i wynik wymienionych operacji jest typu całkowitego, jeżeli zaś choć jedna ze zmiennych lub stałych jest typu rzeczywistego, to i wynik jest typu rzeczywistego.

Wobec istnienia wielu typów zmiennych, kombinacji jest również wiele. Zazwyczaj typ wyniku jest taki jak zmienna zajmująca więcej bajtów, na przykład zmienna całkowita i całkowita długa (*integer* i *longint*) da wynik typu zmiennej całkowitej długiej. Szczegółowe dane na ten temat podano w opisie języka.

Ćwiczenie 10-9

Odczytaj zapisany przez Ciebie program TP-4M (TP04M.PAS) i zmień deklarację zmiennej *LiczbaCalk* z *integer* na *real*. Uruchom program i sprawdź, czy tym razem wyniki będą poprawne.

10.6.2. Wyrażenia

Posługując się arkuszem kalkulacyjnym zapisywałeś wyrażenia matematyczne takie jak spotykane w zadaniach z matematyki lub fizyki. Podobnie możesz czynić w systemie Turbo Pascal. Robiłeś to już zresztą w programach TP-3, TP-4 i TP-5, tyle że może bez dostatecznej pewności. Możesz używać wszystkich działań, a także nawiasów (okrągłych).

Kolejność wykonywania obliczeń jest taka jak w matematyce, tzn. najpierw mnożenie i dzielenie, a potem dodawanie i odejmowanie; działania równorzędne są wykonywane w kolejności od lewej do prawej; ponadto działania w nawiasach są wykonywane przed innymi.

Na przykład chcąc obliczyć drogę ze znanego Ci z fizyki wzoru na drogę w ruchu jednostajnie przyspieszonym $s = v_0 t + a\, t^2/2$ dla różnych wartości czasu $t \geq 0$, powinieneś utworzyć program podobny do TP-7.

Program TP-7

```
program TP07;
  {droga w ruchu jednostajnie przyspieszonym}
var
  v0, a, t, s:  real;
begin
  Write ('Podaj wartosc predkosci poczatkowej v0  ');
  ReadLn(v0);
  Write ('Podaj wartosc przyspieszenia a ');
  Readln(a);
  Write ('Podaj wartosc zmiennej t ');
  ReadLn (t);
  s := v0 * t + a * t * t / 2;
  WriteLn (' Przebyta droga s = ', s:12:5);
  ReadLn
end.
```

Przy zapisywaniu dzielenia wyrażeń pamiętaj o objęciu ich nawiasami. Wyrażenie $x = a * b/c * d$ nie jest równoważne wyrażeniu $x = (a * b)/(c * d)$, ponieważ działania mnożenia i dzielenia są wykonywane od lewej do prawej (żadne z nich nie ma pierwszeństwa przed drugim). W pierwszym ułamek ab/c zostanie pomnożony przez d, dając abd/c, w drugim zaś zostanie obliczone wyrażenie ab/cd.

Jeżeli nie jesteś pewien dlaczego, to napisz program i poeksperymentuj obejmując nawiasami tylko licznik lub tylko mianownik dla konkretnych wartości a, b, c i d.

Zrób jeszcze jeden eksperyment: spróbuj wykonać dzielenie przez 0 (co, jak wiesz, nie jest wykonalne) i sprawdź, jak system potraktował taką próbę.

10.6.3. Warunki logiczne

Warunki formułujesz podobnie jak w arkuszu kalkulacyjnym i programie zarządzania bazami danych, patrz. tab. 10.4. Zauważ, że do zapisywania nierówności słabych używasz dwóch symboli (nierówności ostrej i równości), i to w określonej kolejności.

Tabela 10.4. Relacje

Relacja		Przykłady	
Nazwa	Symbol	Relacja zachodzi	Relacja nie zachodzi
Równe	=	3 = 3 'Ala' = 'Ala'	5 = 5.01 'A' = 'a'
Mniejsze	<	3 < 5 'C' < 'F'	5 < 3 'g' < 'b'
Większe	>	5.01 > 5.001	-5 > -3
Nie mniejsze	>=	3 ≥ 3 'Ewa' ≥ 'Ela'	3 ≥ 3.001 'A' ≥ 'a'
Nie większe	<=	'Jan' ≤ 'Jan' 'Tak' ≤ 'tak'	'Janek' ≤ 'Jan' 'nie' ≤ 'Tak'
Nierówne	<>	'Tak' ≠ 'tak'	3 ≠ 3

10.6.4. Operacje logiczne

Operacje logiczne wykonywałeś już w arkuszu kalkulacyjnym i w programie obsługi baz danych. Podobne operacje mogą być wykonywane w języku Pascal. Ich argumentem są warunki logiczne, takie jak:

$$x < 5, \quad y = 3.2, \quad z >= 7, \quad imie = \text{'Ewa'}$$

Każde z podanych wyrażeń (nazywanych w matematyce relacjami) dla konkretnej wartości zmiennych x, y, z i *imie* jest albo prawdziwe, albo fałszywe.

Z wyrażeń takich za pomocą operatorów logicznych AND, OR i NOT (i, lub, nie), znanych Ci z arkuszy kalkulacyjnych, a także za pomocą innych operatorów, m.in. XOR, tworzy się wyrażenia rozbudowane, tj.:

- wyrażenie powstałe w wyniku poprzedzenia danego wyrażenia operatorem NOT, zwane **negacją**, jest prawdziwe, kiedy dane wyrażenie jest fałszywe;
- wyrażenie złożone z dwóch wyrażeń logicznych połączonych operatorem AND, zwane **iloczynem logicznym**, jest prawdziwe, kiedy oba wyrażenia składowe są prawdziwe;
- wyrażenie złożone z dwóch wyrażeń logicznych połączonych operatorem OR, zwane **sumą logiczną**, jest prawdziwe, kiedy choć jedno z wyrażeń składowych jest prawdziwe;

- wyrażenie złożone z dwóch wyrażeń logicznych połączonych operatorem XOR, zwane **różnicą symetryczną**, jest prawdziwe, kiedy jedno z wyrażeń składowych jest prawdziwe, a drugie fałszywe.

Tabela 10.5. Operatory logiczne

Argumenty		Wartości wyrażeń			
x	y	NOT x	x AND y	x OR y	x XOR y
fałsz	fałsz	prawda	fałsz	fałsz	fałsz
fałsz	prawda	prawda	fałsz	prawda	prawda
prawda	fałsz	fałsz	fałsz	prawda	prawda
prawda	prawda	fałsz	prawda	prawda	fałsz

Na przykład wyrażenie $(x <= 15)$ AND $(x > 7)$ jest prawdziwe wtedy i tylko wtedy, gdy zmienna x jest większa od 7 i jednocześnie nie jest większa od 15, a więc np. dla $x = 10$; wyrażenie $(x > 7)$ OR $(x > 3)$ jest prawdziwe, gdy $x > 3$, a więc np. dla $x = 4$; wyrażenie NOT $(x <= 15)$ jest prawdziwe, gdy $x > 15$, a więc np. dla $x = 20$.

Uwaga. Zauważ, że warunki zapisywane przy operatorach logicznych (stanowiące element składowy warunków złożonych) są umieszczone w nawiasach. Jest to konieczne; w przeciwnym razie system podczas kompilacji będzie sygnalizować błąd, i to w sposób, który nie wskazuje bezpośrednio przyczyny błędu. Warunki pojedyncze, np. $x <= 15$, nie muszą być zapisywane w nawiasach.

10.7. Instrukcje warunkowe i obliczenia wielokrotne

W niemal każdym algorytmie występują działania, które są podejmowane lub nie zależnie od tego, czy jest spełniony określony warunek. Do realizowania tych elementów algorytmów służą w językach programowania **instrukcje warunkowe**.

10.7.1. Instrukcja warunkowa IF

Podstawową instrukcją warunkową jest instrukcja IF (jeżeli), a raczej różne jej odmiany.

```
IF warunek THEN instrukcja 1 ELSE instrukcja 2
```

W instrukcji warunkowej IF najpierw jest badany warunek zapisany po tym słowie. Jeżeli jest spełniony, to wykonywana jest instrukcja 1 po słowie

THEN (to, w takim razie), a gdy nie jest spełniony, to jest wykonywana instrukcja 2 po słowie ELSE (w przeciwnym razie). Po wykonaniu „instrukcji 1" albo „instrukcji 2" system przechodzi do wykonania dalszej części programu.

Jest też prostsza odmiana instrukcji warunkowej IF nie zawierająca słowa ELSE.

```
IF warunek THEN instrukcja
```

Jeżeli warunek jest spełniony, to jest wykonywana instrukcja zapisana po słowie THEN, natomiast gdy nie jest spełniony, to nie jest wykonywana i system przechodzi do wykonywania dalszej części programu.

Zauważ, że gdybyś postawił średnik po instrukcji znajdującej się przed słowem ELSE, wówczas podczas kompilacji zostałby stwierdzony błąd, bowiem ten średnik oznaczałby dla kompilatora koniec polecenia IF THEN (bez ELSE), a słowo ELSE nie miałoby do czego pasować.

Uwaga. W ogólności po słowach THEN i ELSE zamiast pojedynczych instrukcji może być wiele różnych instrukcji objętych słowami BEGIN oraz END.

```
IF warunek THEN
  BEGIN
    lista instrukcji 1
  END
ELSE
  BEGIN
    lista instrukcji 2
  END
```

Instrukcji warunkowych możesz używać między innymi do sprawdzania poprawności wprowadzanych danych. Gdyby na przykład odczytywana liczba miała być nieujemna, wówczas mógłbyś za instrukcją przypisującą wartość zmiennej x umieścić w programie następującą instrukcję warunkową:

```
{odczytanie zmiennej x}
if x < 0 then
  begin
    WriteLn (x, ' < 0');
    Halt
  end;
{dalsza czesc programu ...}
```

Jeżeli wartość zmiennej x będzie ujemna, to zostanie wykonany blok instrukcji za słowem THEN, zawierający instrukcję WRITELN, sygnalizu-

jącą, że zmienna x przyjęła niedozwoloną wartość, oraz instrukcję wywołania procedury HALT, powodującej przerwanie wykonywania programu. W przeciwnym razie (to znaczy gdy odczytana zmienna $x \geq 0$) bezpośrednio po sprawdzeniu warunku program przejdzie do wykonywania dalszej części programu.

Podobnie możesz sprawdzać poprawność odpowiedzi danej przez użytkownika w odpowiedzi na zadane przez program pytanie.

Dzielenie całkowite umożliwia sprawdzanie, czy dana liczba jest (całkowitą) wielokrotnością innej liczby (różnej od zera). Jest tak wtedy, kiedy reszta z dzielenia całkowitego jest równa zeru. Instrukcja warunkowa jest przydatna do przedstawiania wyników badania.

Program TP-8

```
program TP08;
  {operacje arytmetyczne}
var  dzielna, dzielnik: integer;
begin
  Write('wprowadz dzielna calkowita ');
  ReadLn (dzielna);
  Write('wprowadz dzielnik calkowity ');
  ReadLn (dzielnik);
  if dzielna MOD dzielnik = 0 then
    WriteLn ('Liczba ', dzielna, ' dzieli sie przez ',
                                                  dzielnik)
  else
    WriteLn ('Liczba ', dzielna, ' nie dzieli sie przez ',
                                                  dzielnik);
  ReadLn
end.
```

Takimi działaniami posłużymy się pisząc program rozkładający liczbę na czynniki pierwsze.

Spróbuj teraz sam napisać program rozwiązywania równania liniowego $ax+b = c$ na podstawie algorytmu 2. Czy nie masz trudności z powiązaniem dwóch warunków: $a = 0$ oraz $b = c$? Do zapisania tego typu złożonych zależności można posłużyć się operatorami logicznymi.

Program TP-9

```
Program TP09;
 {realizujacy algorytm 2}
var
  a,b,c: real;
begin
```

```
  WriteLn ('Podaj kolejno liczby a, b, c oddzielajac je
                                                odstepami');
  WriteLn ('i nacisnij ENTER');
  ReadLn (a,b,c);
  WriteLn ('Rownanie ', a:6:2, ' x +', b:6:2, ' =', c:6:2);
  if a <> 0 then
            WriteLn ('ma rozwiazanie x = ', (c - b) / a:6:2);
  if (a = 0) and (b <> c) then
            WriteLn ('nie ma rozwiazan');
  if (a = 0) and (b = c) then
            WriteLn ('jest zawsze spelnione');
  ReadLn
end.
```

Zauważ, że warunek pojedynczy w pierwszej instrukcji IF jest zapisany bez użycia nawiasów, natomiast warunki w pozostałych dwóch instrukcjach IF są zapisane w nawiasach, ponieważ wchodzą w skład wyrażeń.

Do utworzenia programu realizującego algorytm 2 można także posłużyć się instrukcjami warunkowymi umieszczonymi wewnątrz innych instrukcji warunkowych.

10.7.2. Instrukcje warunkowe „zagnieżdżone"

Wśród instrukcji wykonywanych warunkowo mogą być zawarte następne instrukcje warunkowe. W ten sposób jedna instrukcja warunkowa jest „zagnieżdżona" wewnątrz drugiej. Można to czynić bezpośrednio, np.

```
if a = 0 then if b = c then WriteLn('sprzeczne')
  else WriteLn('prawdziwe');
```

a można też „zagnieżdżać" następne instrukcje warunkowe wewnątrz bloków objętych słowami BEGIN ELSE.

Programy takie jak TP-9, w których trzeba używać operatorów logicznych, żeby zrealizować schemat obliczeń określony algorytmem, możesz obecnie zapisać prościej.

Program TP-10

```
program TP10;
 {realizujacy algorytm 2}
var
  a,b,c: real;
begin
  WriteLn ('Podaj kolejno liczby a, b, c oddzielajac je
                                                odstepami');
```

```
  WriteLn ('i nacisnij ENTER');
  ReadLn (a,b,c);
  WriteLn ('Rownanie ', a:6:2, ' x +', b:6:2, ' =', c:6:2);
  if a <> 0 then
              WriteLn ('ma rozwiazanie x = ', (c - b) / a:6:2)
            else
             if b<>c then
               WriteLn ('nie ma rozwiazan')
             else
               WriteLn ('jest zawsze spelnione')
end.
```

Ćwiczenie 10-10

Spróbuj napisać część programu obliczania pierwiastków równania kwadratowego na podstawie algorytmu 3 do zadania 8-1, na razie bez wyznaczania samych pierwiastków (ponieważ nie umiesz jeszcze obliczać pierwiastka kwadratowego), natomiast z rozróżnieniem różnych możliwych przypadków. Posłuż się:

A. Zagnieżdżonymi instrukcjami warunkowymi.

B. Złożonymi warunkami logicznymi, zapisanymi przy użyciu operatorów logicznych.

10.7.3. Instrukcja wyboru CASE OF

W niektórych sytuacjach wygodnie jest rozdzielić bieg programu nie na dwie różne drogi, lecz od razu na większą liczbę. Zetknąłeś się z taką sytuacją w algorytmie 16 do zadania 8-12. Przy małej liczbie dróg można posługiwać się zagnieżdżonymi instrukcjami warunkowymi, lecz przy większej ich ilości wygodniej jest stosować konstrukcję CASE OF. Ma ona postać:

```
CASE wyrażenie OF
  lista wartości 1:  instrukcja 1
  lista wartości 2:  instrukcja 2
    ...
  lista wartości n:  instrukcja n
  ELSE instrukcja n+1
END
```

Bieg programu jest tu rozdzielony przez wartości wyrażenia zapisanego w pierwszej linii instrukcji (za słowem CASE), a nie przez formułowane warunki logiczne, jak w instrukcji warunkowej. Gdy „wyrażenie" przyjmuje jedną z wartości wymienionych jako „lista wartości 1", wówczas jest wykonywana „instrukcja 1", natomiast gdy przyjmuje wartość wymienioną

w „lista wartości 2", wówczas jest wykonywana „instrukcja 2", itd.; w przeciwnym razie — tzn. gdy wartość „wyrażenia" nie jest objęta żadną z list wartości, wówczas jest wykonywana „lista poleceń n+1".

Użycie słowa (kluczowego) ELSE i instrukcji $n+1$ nie jest konieczne, konieczne jest natomiast użycie słowa (kluczowego) END, żeby program tłumaczący wiedział, gdzie kończy się cała instrukcja CASE OF.

Elementy listy wartości można oddzielać przecinkami, a ich zakresy podawać z dwiema kropkami, np. `'a'..'p'`.

A oto fragment programu z przykładem użycia instrukcji CASE OF do rozróżnienia wyboru przedmiotu, którego miałby dotyczyć test. Pliki z pytaniami i odpowiedziami dotyczącymi różnych przedmiotów nauczania (historia, geografia), na przykład podobne do pliku DANE5.DAT, masz przygotowane. Instrukcje odczytywania danych z plików poznasz w dalszej części rozdziału. Wybieranie przedmiotu przez użytkownika może polegać na naciśnięciu klawisza literowego z nazwą przedmiotu. Zakładamy, że zmienna *haslo* jest równa odpowiedniej literze (odpowiadającej nazwie przedmiotu).

Program TP-11 .

```
program TP11;
   {fragment programu do samodzielnego uzupelnienia
                                          i rozwiniecia}
var haslo: string;
begin
     {miejsce na początek programu}
   Write('Wybierz nazwe przedmiotu (pierwsza litere): ');
   ReadLn(haslo);
   case haslo of
     'm','M':
        instrukcje odczytania pliku z pytaniami testowymi
                                          z matematyki;
     'f','F':
         instrukcje odczytania pliku z pytaniami testowymi
                                          z fizyki;
     ...
     else
       WriteLn('Dla tego przedmiotu nie ma przygotowanych pytan')
   end;
     {miejsce na dalszy ciag programu}
end.
```

Uwaga. Zmienna *haslo* może być także zadeklarowana jako zmienna znakowa (typ *char*), skoro ma zawierać jeden znak.

10.7.4. Obliczenia wielokrotne. Pętle

W niektórych algorytmach występują pętle służące do wielokrotnego powtarzania obliczeń. Do szczególnie użytecznych cech programu komputerowego należy łatwość ich realizowania za pomocą służących do tego instrukcji. Jest kilka odmian instrukcji, spośród których użytkownik może wybierać te, które w danej sytuacji są najwygodniejsze.

Instrukcje różnią się tym, czy warunek jest badany na początku pętli czy na jej końcu, oraz czy ma on znaczenie warunku kontynuacji obliczeń czy ich zakończenia.

Instrukcja WHILE – DO

Do zrealizowania pętli, w której warunek jest badany na jej początku, służy instrukcja WHILE, mająca postać:

```
WHILE warunek DO instrukcja
```

Jeżeli warunek za słowem WHILE (podczas gdy) jest spełniony, to jest wykonywana instrukcja zapisana za słowem DO (wykonaj), po czym warunek jest badany ponownie; jeżeli jest spełniony, to instrukcja jest ponownie wykonywana itd. Wykonywanie instrukcji zapisanej za słowem DO jest powtarzane tak długo, jak długo warunek pozostaje spełniony. Gdy podczas kolejnego sprawdzania okaże się, że warunek nie jest spełniony, wówczas system przechodzi do wykonywania dalszej części programu (i do badania warunku już nie powraca).

Można powiedzieć, że w pętli WHILE występuje warunek kontynuacji obliczeń. Zauważ, że ponieważ warunek jest badany na początku pętli, to może się zdarzyć, że instrukcja nie zostanie wykonana ani razu.

Instrukcja zapisana za słowem DO może oczywiście zawierać wiele instrukcji objętych słowami BEGIN i END (jak nawiasami).

Przyjrzyj się programowi TP-12, realizującemu przy użyciu pętli WHILE algorytm 4 do zadania 8-2 (możesz go odczytać z pliku TP12.PAS).

Program TP-12

```
program TP12;
 {realizujacy algorytm 4  -  petla WHILE}
var
  liczba, mnoznik: integer;
begin
  Write ('Podaj liczbe calkowita ');
  ReadLn (liczba);
  mnoznik := 2;
```

```
  while mnoznik <= 10 do
    begin
      Write ('Iloczyn ', liczba, ' przez ');
      WriteLn (mnoznik, ' = ', liczba * mnoznik);
      mnoznik := mnoznik + 1
    end;
  ReadLn
end.
```

Na początku programu zostaje napisany komunikat (WRITE) i odczytana zmienna *liczba* (READLN); zmiennej *mnoznik* zostaje nadana wartość początkowa 2. Przy takiej wartości zmiennej *mnoznik* warunek pętli WHILE jest spełniony, wobec czego następuje pierwsze wykonanie instrukcji za słowem DO. Pętlą są objęte instrukcje: WRITE, WRITELN oraz wykonywana po nich instrukcja zwiększająca wartość zmiennej *mnoznik* o 1. Za pierwszym razem będą wykonane obliczenia dla zmiennej *mnoznik* = 2 (i wypisane na ekranie wyniki), po czym zmienna zostanie zwiększona do 3. Zostanie zbadany warunek, a że dla *mnoznik* = 3 jest on spełniony, to obliczenia w pętli będą wykonane ponownie. Największą wartością zmiennej całkowitej *mnoznik*, dla której warunek kontynuacji obliczeń jest spełniony, jest 10. Wtedy instrukcje w pętli będą wykonane ostatni raz, ponieważ zmienna *mnoznik* zostanie zwiększona do 11 i warunek przestanie być spełniony. Jako następna będzie wykonana instrukcja READLN (służąca, jak pamiętasz, do „przytrzymania" ekranu wynikowego) i program zostanie zakończony.

Uwaga. Zwracaj uwagę na formułowanie warunku kontynuowania obliczeń w pętli (w każdym rodzaju pętli), ponieważ możesz przez nieuwagę spowodować, że obliczenia będą wykonywane bez końca. Jeżeli jednak dojdzie do tego, to nie podejmuj kroków desperackich w rodzaju wyłączania komputera, lecz naciśnij klawisze *Ctrl-Break* jeden lub dwa razy.

Ćwiczenie 10-11

A. Sprawdź, czy program TP-12 działa zgodnie z Twoimi oczekiwaniami. W jakim zakresie danych? (Czy po wprowadzeniu wartości 6000 też?) Sprawdź, czy zmiana deklaracji typu zmiennych *liczba* i *mnoznik* pomoże.

B. Zmień w programie TP-12 warunek kontynuacji obliczeń na *mnoznik* > 1 (program możesz zapisać pod zmienioną nazwą). Zastanów się, jaki powinien być przebieg obliczeń. Uruchom program i sprawdź, czy rezultat jest taki, jakiego oczekiwałeś. Przerwij obliczenia.

Instrukcja REPEAT – UNTIL

Słowa REPEAT (powtarzaj) i UNTIL (aż do) wyznaczają początek i koniec pętli służącej do wielokrotnego wykonywania listy instrukcji umieszczonej pomiędzy nimi. Podobnie jak w instrukcji WHILE liczba powtórzeń pętli nie jest znana z góry, lecz zależy od spełnienia warunku. Warunek podaje się za słowem UNTIL; określa on zakończenie obliczeń, np.

```
REPEAT
    lista instrukcji
UNTIL   i > 10
```

Instrukcje zawarte na liście są wykonane po raz pierwszy bez badania warunku. Jest on badany każdorazowo na końcu pętli. Jeżeli nie jest spełniony, to instrukcje zawarte na liście są wykonywane ponownie, w przeciwnym razie system uznaje wykonanie instrukcji pętli za zakończone i przechodzi do wykonywania instrukcji z dalszej części programu (za słowem UNTIL).

Ćwiczenie 10-12

A. Zmodyfikuj program TP-12, używając pętli REPEAT zamiast WHILE. Zapisz go pod zmienioną nazwą. Zwróć uwagę na potrzebę zmiany warunku, który ma stać się warunkiem zakończenia pętli.

B. Porównaj wprowadzone przez Ciebie zmiany z podanym zmienionym fragmentem (pełny program jest zapisany w pliku TP12R.PAS.)

```
repeat
   Write ('Iloczyn ', liczba, ' przez ');
   WriteLn (mnoznik, ' = ', liczba * mnoznik);
   mnoznik := mnoznik + 1
until mnoznik > 10;
```

Wybór rodzaju pętli

Spróbuj napisać program realizujący algorytm 5 do zadania 8-3 (wyznaczania sumy wprowadzanych liczb oraz liczby największej). Zastanów się, czy wygodniej jest Ci użyć pętli REPEAT czy pętli WHILE.

Porównaj teraz swój program z programem TP-13 (zapisanym w pliku TP13.PAS), w którym użyto pętli REPEAT.

Program TP-13

```
program TP13;
 {realizujacy algorytm 5  -  petla REPEAT}
var
  Suma, LiczbaWprow, Najwieksza: real;
```

```
  Licz: integer;
begin
  Suma := 0.0;
  Licz := 0;
  repeat
    WriteLn ('Podaj liczbe nieujemna');
    Write ('(liczba ujemna konczy dzialanie programu) ');
    ReadLn (LiczbaWprow);
    if LiczbaWprow >= 0 then
      begin
        Suma := Suma + LiczbaWprow;
        Licz := Licz + 1;
        if Licz = 1 THEN Najwieksza := LiczbaWprow;
        if LiczbaWprow > Najwieksza then Najwieksza :=
                                          LiczbaWprow;
        WriteLn ('Suma ', Licz, ' liczb = ', Suma);
        WriteLn ('najwieksza liczba = ', Najwieksza);
        WriteLn
      end
  until LiczbaWprow < 0
end.
```

Instrukcja FOR

Obliczenia w pętli bywają prowadzone określoną z góry ilość razy, na przykład drukowanie liczb mnożonych przez kolejne liczby całkowite od 2 do 10. W takiej sytuacji wygodniejsza w realizowaniu pętli od instrukcji REPEAT UNTIL jest instrukcja FOR. Ma ona postać:

```
FOR licznik := wartość początkowa TO wartość końcowa
                                          DO instrukcja
```

lub

```
FOR licznik := wartość początkowa DOWNTO wartość końcowa
                                          DO instrukcja
```

Najpierw zmienna służąca jako „licznik" pętli przybiera „wartość początkową". Potem badany jest warunek, czy wartość zmiennej „licznik" nie przekroczyła „wartości końcowej" (tzn. nie stała się od niej większa, a w wypadku DOWNTO – mniejsza). Jeżeli nie przekroczyła, to instrukcja umieszczona za słowem DO jest wykonywana. Wartość licznika jest następnie zwiększana o 1, a jeżeli w instrukcji FOR występuje słowo DOWNTO, to zmniejszana o 1. Wtedy instrukcja powraca do badania warunku.

Wykonywanie instrukcji w takiej pętli zostaje przerwane, gdy zmienna „licznik" przekracza „wartość końcową". Po przerwaniu pętli obliczeń, jako kolejna jest wykonywana instrukcja następna po instrukcji FOR.

Uwaga. Zmiennej użytej w roli licznika obiegów pętli, wymienionej za słowem FOR, nie należy nadawać wartości wewnątrz pętli, ponieważ może to doprowadzić do nieprawidłowego przebiegu obliczeń; w szczególności program może nie wyjść z pętli (pozostaje wówczas przerwanie jego wykonania naciśnięciem klawiszy *Ctrl-Break*).

Pętla FOR dobrze się nadaje do realizowania algorytmu 4, gdyż liczba powtórzeń jest dana z góry.

Program TP-14

```
program TP14;
 {realizujacy algorytm 4}
var
  liczba, mnoznik: integer;
begin
  Write ('Podaj liczbe calkowita ');
  ReadLn (liczba);
  for mnoznik := 2 to 10 do
    begin
      Write ('Iloczyn ', liczba, ' przez ');
      WriteLn (mnoznik, ' = ', liczba * mnoznik)
    end;
  ReadLn
end.
```

Pętlą FOR są objęte dwie instrukcje drukowania — te same, co w programie TP-12 (nie ma tu natomiast instrukcji zwiększającej wartość zmiennej *mnoznik*, ponieważ jest to realizowane w samym mechanizmie pętli). Pętla ta jest wykonywana 9 razy. Pełniąca funkcję licznika zmienna całkowita *mnoznik* na początku przyjmuje wartość 2 i po każdym obiegu pętli jest zwiększana o 1. Gdy przekroczy 10, wówczas obliczenia w pętli przestają być wykonywane, i jako następna jest wykonywana instrukcja READLN (ostatnia instrukcja programu przez końcowym END). Ostatnia wartość zmiennej *mnoznik*, dla której obliczenia w pętli są wykonywane, wynosi 10.

Pętli FOR można użyć do podnoszenia liczb do potęgi całkowitej przez wielokrotne mnożenie. Wykładnik potęgi określałby liczbę powtórzeń, a liczba potęgowana pełniłaby funkcję mnożnika. Zmienna służąca do obliczenia wyniku (mnożna) powinna mieć nadaną wartość początkową 1 przed rozpoczęciem obliczeń w pętli.

Ćwiczenie 10-13

Napisz program odczytujący liczbę wprowadzoną przez użytkownika i wyznaczający jej:

A. czwartą potęgę.
B. dziesiątą potęgę.

Instrukcję FOR będziesz mógł stosować m.in. do obliczania wartości funkcji $f(x)$ dla różnych wartości jej argumentu (w celu sporządzenia wykresu). W sytuacjach takich zmienną niezależną x zwiększa się zazwyczaj ze stałym krokiem, na przykład o 0,5, lub 0,1. Wówczas masz do wyboru dwa sposoby użycia pętli FOR:

- zastosować jako licznik zmienną całkowitą (zwiększaną o 1), i kolejne wartości zmiennej niezależnej x obliczać jako wyrażenia zależne od wartości licznika, np. $x = 2 + 0,1 * licznik$;
- za każdym obiegiem pętli zwiększać wartość zmiennej x o wielkość kroku.

W programie TP-15 użyto drugiego z wymienionych sposobów do obliczania wartości funkcji $f(x) = -2x^2 + 8x$ w przedziale od 2 do 3 z krokiem 0,01.

Program TP-15

```
program TP15;
  {wielokrotne dodawanie liczb rzeczywistych}
var
  licznik: integer;
  x, krok:  real;
begin
  x := 2;
  krok := 0.01;
  for licznik := 0 to 100 do
    begin
      x := x + krok;
      WriteLn (x:15:10, -2 * x * x + 8 * x :20:10)
    end;
  ReadLn
end.
```

Zwróć uwagę na wartości przyjmowane przez zmienną niezależną x. Nie są one dokładnie równe tym liczbom, jakim powinny. Po stukrotnym wykonaniu dodawania liczby 0,01 do 2 wynik nie jest równy dokładnie 3 (konkretne wartości mogą zależeć od komputera). Przy dodawaniu stu liczb błąd nie jest duży, ale przy dodawaniu większej ich liczby może urosnąć.

Ćwiczenie 10-14

A. Zmień w programie TP-15 sposób obliczania wartości zmiennej x z sumowania kroków na bezpośrednie wyznaczanie w funkcji zmien-

nej *licznik* ($x = 2 + licznik \times 0,01$). Możesz posłużyć się programem z pliku TP15M.PAS.

B. Porównaj oba sposoby, drukując obok siebie dwie wartości zmiennej x wyznaczone różnymi sposobami (zmodyfikuj jeden z programów, np. zamiast drukowania wartości funkcji kwadratowej drukuj wartość zmiennej x wyznaczaną drugim sposobem).

C. Zorientuj się, jak szybko i jak dokładnie Twój komputer dodaje liczby rzeczywiste zmniejszając krok do 0,001 lub nawet do 0,0001 i zwiększając odpowiednio liczbę przebiegów pętli (zmodyfikuj program TP-15 przesuwając instrukcję WRITELN poza pętlę).

Podane uwagi nie dotyczą wyłącznie pętli FOR, lecz ogólnie obliczeń w komputerze, i ich wpływu na obliczenia w pętlach. Wynika stąd bardzo ważny wniosek praktyczny odnośnie formułowania warunków na kończenie obliczeń w pętlach REPEAT UNTIL i WHILE DO, ponieważ nie może on mieć postaci równości liczb rzeczywistych.

Zapamiętaj!

Przy stosowaniu zmiennych rzeczywistych obliczenia nie są dokładne.
Warunek końcowy pętli powinien być formułowany w postaci nierówności.

Zagnieżdżone pętle

W liście instrukcji wewnątrz pętli mogą występować wszelkie instrukcje, a więc także instrukcje warunkowe i pętle. Jeżeli pojedyncza pętla umożliwia nadanie kolejnych wartości jednej zmiennej, to pętla umieszczona w pętli umożliwia systematyczne zmienianie dwóch zmiennych.

W programie TP-16 użyto pętli FOR „zagnieżdżonej" wewnątrz pętli FOR, do obliczania tabliczki mnożenia. Zakres obliczeń jest niepełny, ponieważ przy takiej formie wydruku 100 liczb nie zmieściłoby się na ekranie.

Program TP-16

```
Program TP16;
  {tabliczka mnozenia  (niepelna) - zagniezdzone petle FOR}
var
  i, j, iloczyn: integer;
begin
    for i := 2 to 5 do
```

```
    for j := 2 to 7 do
      begin
        iloczyn := i * j;
        WriteLn (i, ' x ', j, '   =   ', iloczyn)
      end;
  ReadLn
end.
```

Zewnętrzna pętla FOR zaczyna się w pierwszej linii programu za słowem BEGIN (po deklaracjach zmiennych); jako licznik służy w niej zmienna całkowita i. Wewnętrzna pętla FOR zaczyna się w następnej linii programu; jako licznik służy w niej zmienna j. Za pomocą zagnieżdżonych pętli FOR możesz np. nadawać lub odczytywać wartości z dwuwymiarowych tablic.

Zakres zmian licznika pętli nie musi być określony przez stałe (np. 2, 7), lecz także za pomocą zmiennych. Przy obliczaniu tabliczki mnożenia — jeżeli wiesz, że mnożenie jest przemienne, tzn. że $x \times y = y \times x$ — możesz w wewnętrznej pętli zmieniać j nie od 2 do 7, lecz od i do 7.

Ćwiczenie 10-15

A. Zmodyfikuj program TP-16 przez zmianę wartości początkowej zmiennej j z 2 na i (możesz odczytać program z pliku TP16M.PAS). Zastanów się, dla jakich wartości zmiennych i, j powinieneś otrzymać wyniki mnożenia; uruchom program i sprawdź.

B. Jeżeli masz niewykorzystane miejsce na ekranie, to spróbuj zmienić wartość końcową pętli, np. z 7 na 8.

C. Zmodyfikuj program tak, żeby drukował na ekranie pełną tabliczkę mnożenia w zakresie od 1 do 10 w formie tablicy zawierającej same iloczyny (po dziesięć w wierszu). Możesz porównać swój program z programem zapisanym w pliku TP16T.PAS.

10.7.5. Instrukcja skoku. Etykiety

Do zmiany kolejności wykonywania instrukcji programu służy instrukcja skoku GOTO (*go to* — idź do). Wymaga ona wskazania instrukcji, która ma być wykonana jako następna w kolejności.

Do wskazywania instrukcji w programie służą **etykiety** (*label*).

Etykieta musi być zadeklarowana (przed stałymi), umieszczona przy instrukcji docelowej (w tym samym wierszu — jako poprzedzający ją element, oddzielony dwukropkiem) i wymieniona w instrukcji GOTO.

Za pomocą instrukcji skoku można sterować obliczeniami (zapoznaj się z programem zamieszczonym w pliku TPSKOK.PAS). Ze względu na małą czytelność programu z instrukcjami skoku nie zaleca się jednak ich używania.

Uwaga. Pisząc swoje programy stosuj instrukcje strukturalne, takie jak pętle oraz instrukcje warunkowe (a nie stosuj instrukcji skoku).

10.7.6. Przykładowe programy

Przeanalizuj algorytm 7 do zadania 8-5. Zauważ, że znasz wszystkie elementy potrzebne do napisania programu (podzielność jednej liczby przez drugą badałeś w programie TP-8). Spróbuj napisać program realizujący ten algorytm. Sprawdź jego działanie. W razie trudności posłuż się programem TP-17.

Program TP-17

```
Program TP17;
  {rozkladanie liczby na czynniki pierwsze}
var
  n, czynnik, iloraz:  integer;
begin
  Write ('Wprowadz liczbe calkowita dodatnia ');
  ReadLn (n);
  if n < 1 then Halt;
  czynnik := 2;
  iloraz := n;
  while (iloraz > 1) and (czynnik <= n) do
    if iloraz mod czynnik = 0 then
      begin
        iloraz := iloraz DIV czynnik;
        Writeln ('Kolejny czynnik liczby ', n, ' = ', czynnik)
      end
    else
        czynnik := czynnik + 1;
  Readln
end.
```

Sprawdź działanie programu na różnych danych. Posłuż się pracą krokową oraz instrukcjami debugera, żeby „podejrzeć" działanie programu. Zmodyfikuj podany program tak, by realizował algorytm 8 do zadania 8-6 (wyznaczenia największego wspólnego dzielnika i najmniejszej wspólnej wielokrotności dwóch liczb naturalnych). Możesz też odczytać go z pliku TP18.PAS i przeanalizować jego działanie (posługując się pracą krokową).

Program TP-18

```
Program TP18;
  {wyznaczanie NWD i NWW dwoch liczb naturalnych}
var
```

```
  m,n,ilorazm,ilorazn,czynnik,nwd,nww: integer;
begin
  Write ('Wprowadz pierwsza liczbe calkowita dodatnia ');
  ReadLn (m);
  Write ('Wprowadz druga liczbe calkowita dodatnia    ');
  ReadLn (n);
  if (m < 1) or (n < 1) then Halt;
  czynnik := 2;
  ilorazm := m;
  ilorazn := n;
  nwd := 1;
  While (ilorazm > 1) and (ilorazn > 1) and (czynnik <= m) and
                                          (czynnik <= n) do
    if (ilorazm MOD czynnik = 0) and (ilorazn MOD
                                        czynnik = 0) then
      begin
        ilorazm := ilorazm DIV czynnik;
        ilorazn := ilorazn DIV czynnik;
        nwd := nwd * czynnik;
        WriteLn ('Kolejny czynnik liczb ',m,' i ',n,' = ',
                                                  czynnik)
      end
    else
        czynnik := czynnik + 1;
  nww := (m DIV nwd) * n;
  WriteLn ('NWD liczb ', m, ' i ', n, ' = ', nwd);
  WriteLn ('NWW liczb ', m, ' i ', n, ' = ', nww);
  ReadLn
end.
```

Możesz być zaskoczony sposobem zapisania wzoru na NWW. Czy nie prościej byłoby zapisać zgodnie ze wzorem z rozdz. 8 wyrażenie:

```
nww := m * n / nwd
```

Rzeczywiście byłoby to bardziej naturalne, zmiana została jednak podyktowana typem wyniku oraz zakresem liczb całkowitych. Spróbuj zmienić symbol DIV na symbol dzielenia rzeczywistego (kreskę ułamkową /) i skompilować program. Czy system dokonał kompilacji, czy też wskazał niezgodność typów w tej linii programu? Co do zakresu liczb, to już miałeś doświadczenia w innych programach. Uruchom program TP-18 dla danych 480 i 720; powinieneś otrzymać NWD = 240 i NWW = 1440. Zmień teraz w programie instrukcję obliczania NWW na podaną wyżej i uruchom program dla tych samych danych. Czy otrzymałeś wyniki? Czy NWW ma taką samą wartość jak poprzednio, czy też inną (74)? Który z wyników jest prawidłowy?

Błąd w obliczeniach jest związany z przekroczeniem zakresu zmiennych zadeklarowanych jako całkowite (*integer*). Nadmiar ten bierze się stąd, że po zmianie instrukcji najpierw jest obliczany iloczyn zmiennych m i n (wynoszący ponad 300 000, a więc więcej niż zakres liczb całkowitych), a dopiero później jest on dzielony przez NWD (równą 240). Jego powstanie jest związane z przebiegiem obliczeń, a nie z wynikiem, który mieści się w zakresie dopuszczalnym. Zauważ, że system nie sygnalizuje takiego błędu.

Ustrzeżesz się przed nim zmieniając typ przynajmniej jednej ze zmiennych m i n na całkowitą długą (*longint*). Jeżeli jednak będziesz posługiwał się programem TP-18 dla większych danych, to i tak może się zdarzyć, że wystąpi nadmiar. W takiej sytuacji zmień typ wszystkich zmiennych całkowitych na *longint* (zapisz program pod zmienioną nazwą).

Zmiana zakresu liczb nie rozwiązuje zupełnie problemu, lecz jedynie go przesuwa w stronę większych liczb. Możesz się o tym przekonać podając takie dane, których największa wspólna wielokrotność przekroczy zakres liczb całkowitych długich. Podaj np. liczby, z których jedna jest odpowiednio dużą potęgą dwóch, a druga trzech (wówczas NWW jest ich iloczynem, ponieważ nie mają wspólnego dzielnika większego od 1): 4096 i 6561, 16384 i 19683, 65536 i 59049. Czy dla ostatniej pary liczb otrzymujesz wynik prawidłowy?

Jak możesz ustrzec się przed błędem? Na przykład ogranicz zakres danych i ewentualnie sprawdzaj w programie, czy nie został przekroczony.

Ćwiczenie 10-16

Napisz program realizujący algorytm 9 (Euklidesa) i porównaj uzyskane wyniki z otrzymanymi za pomocą programu TP-18. Jeżeli różnica w czasie obliczeń jest zauważalna, to sprawdź, który jest szybszy. (Łatwiej zauważysz różnicę szybkości dla dużych danych, zatem dobierz odpowiedni typ zmiennych w obu programach.)

10.8. Tablice, rekordy, definiowane typy zmiennych

10.8.1. Tablice

Deklarowanie tablic

Tablice deklaruje się razem z innymi zmiennymi, określając typ danej zmiennej, będącej nazwą tablicy, za pomocą słowa ARRAY, po którym podaje się zakres **indeksu** lub indeksów w nawiasach kwadratowych, a następnie typ wielkości przechowywanych w tablicy. Zakres indeksu określa się