Program QB-5

```
REM Program QB-5   Obliczanie obwodu kola
CONST PI = 3.14159
INPUT "Podaj promien kola ", r
PRINT "Obwod kola o promieniu "; r; "wynosi"; 2 * PI * r
```

Rozwiązanie takie ma zalety związane z użyciem pamięci komputera (co ma znaczenie przy większych programach), przede wszystkim jednak zmniejsza niebezpieczeństwo popełnienia błędu przez niejednakowe napisanie wartości stałej w różnych miejscach programu oraz czyni program bardziej przejrzystym, ułatwiając jego kontrolę i wyszukiwanie ewentualnych błędów.

Ćwiczenie 9-7

Zmodyfikuj program QB-5 tak, by wyznaczał pole koła.

9.5. Działania na zmiennych liczbowych

9.5.1. Operacje arytmetyczne

Na zmiennych i stałych liczbowych możesz wykonywać różne działania matematyczne, podobnie jak w arkuszu kalkulacyjnym. Oprócz czterech działań arytmetycznych: dodawania, odejmowania, mnożenia i dzielenia, możesz wykonywać potęgowanie (x^y) oraz dzielenie całkowite (tab. 9.2).

Tabela 9.2. Działania na zmiennych liczbowych

Działanie	Symbol	Przykład	Wynik
Dodawanie	+	3 + 5.5	8.5
Odejmowanie	−	17 − 12	5
Mnożenie	*	3 * 17	51
Dzielenie	/	9/4	2.25
Potęgowanie	^	3^2	9 (3^2)
Dzielenie całkowite	\	11\4	2
Reszta z dzielenia całkowitego	MOD	11 MOD 4	3

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 wynosi 2, a z dzielenia pozostaje reszta równa 3. W celu oswojenia się z zapisywaniem rzadziej używanych działań wykonaj ćwiczenie 9-8.

Ćwiczenie 9-8

A. Napisz program QB-6 lub odczytaj go z pliku QB06.BAS.

Program QB-6

```
REM potegowanie, dzielenie i dzielenie calkowite - przyklady
CLS
PRINT "2 ^ 3 ="; 2 ^ 3, "3 ^ 2 ="; 3 ^ 2
PRINT "17 \ 5 ="; 17 \ 5, "17 MOD 5 ="; 17 MOD 5
PRINT "17 / 5 ="; 17 / 5
```

B. Oblicz wyniki, jakie powinieneś otrzymać (jeżeli dobrze rozumiesz przedstawiony zapis).

C. Uruchom program i sprawdź, czy otrzymane wyniki są zgodne z Twoimi.

9.5.2. Wyrażenia arytmetyczne

Posługując się arkuszem kalkulacyjnym zapisywałeś wyrażenia arytmetyczne, takie jak spotykane w zadaniach z matematyki lub fizyki. Podobnie możesz czynić w systemie QBasic. Czyniłeś to zresztą w programie QB-3, tyle że może bez dostatecznej pewności. Możesz używać wszystkich działań, a także nawiasów (okrągłych). Kolejność wykonywania obliczeń jest taka jak w matematyce, tzn. najpierw potęgowanie, później mnożenie i dzielenie, a na końcu dodawanie i odejmowanie; działania równorzędne są wykonywane w kolejności od lewej do prawej; ponadto działania w nawiasach są wykonywane przed innymi.

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

Program QB-7

```
REM Program QB-7 - obliczanie drogi w ruchu jednostajnie zmiennym
INPUT "Podaj wartosc predkosci poczatkowej v0 ", v0
INPUT "Podaj wartosc przyspieszenia a ", a
INPUT "Podaj wartosc zmiennej t (liczba ujemna konczy) "; t
LET s = v0 * t + a * t ^ 2 / 2
PRINT s
```

Uwaga. 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.

9.5.3. Warunki logiczne

Warunki formułujesz podobnie jak w arkuszu kalkulacyjnym i programie zarządzania bazami danych, patrz tab. 9.3.

Tabela 9.3. 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	>= =>	$3 \geq 3$ "Ewa" $\geq$ "Ela"	$3 \geq 3.001$ "A" $\geq$ "a"
Nie większe	<= =<	"Jan" $\leq$ "Jan" "Tak" $\leq$ "tak"	"Janek" $\leq$ "Jan" "nie" $\leq$ "Tak"
Nierówne	<>	"Tak" $\neq$ "tak"	$3 \neq 3$

9.5.4. Operacje logiczne

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

$x < 5$, $y = 3.2$, $z >= 7$, $x\$$="Ewa".

Każde z podanych wyrażeń dla konkretnej wartości zmiennych x, y i z jest albo prawdziwe, albo fałszywe.

Z wyrażeń takich za pomocą operatorów logicznych (tab. 9.4) 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 9.4. 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 $x = 20$.

9.6. 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**.

9.6.1. Instrukcje warunkowe

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

Instrukcja blokowa IF

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

Jeżeli warunek jest spełniony, to jest wykonywana „lista instrukcji 1" (*if then* — jeżeli to), natomiast w przeciwnym razie (*else*) jest wykonywana „lista instrukcji 2". Instrukcja nazywana jest blokową, ponieważ odnosi się nie do pojedynczych instrukcji wykonywanych warunkowo, lecz do całych ich bloków, zwanych tu listami. Instrukcje napisane w następnych liniach po słowach END IF, kończących tę instrukcję warunkową (*end* — koniec), są wykonywane niezależnie od tego, czy warunek jest spełniony czy nie.

Użycie słowa ELSE i drugiej listy instrukcji nie jest konieczne. Instrukcja taka powoduje wykonanie instrukcji wymienionych na liście, kiedy warunek jest spełniony, natomiast kiedy nie jest spełniony, to żadne z instrukcji wymienionych przed słowami END IF nie są wykonywane.

```
IF warunek THEN
    lista instrukcji
END IF
```

Instrukcja blokowa IF ma też odmianę o bardziej rozbudowanej strukturze, której nie będziemy tu wykorzystywać.

Uwaga. Przyjęte jest pisanie list instrukcji z wcięciem o szerokości kilku spacji w stosunku do słów: IF, ELSE oraz END IF. Taki sposób zapisu ułatwia analizowanie treści programu i wyszukiwanie ewentualnych błędów. Dotyczy to nie tylko instrukcji warunkowej, lecz wszelkich instrukcji, których zakresem działania jest lista instrukcji.

Instrukcja prosta IF

Jest też uproszczona odmiana tej instrukcji, zapisywana w jednej linii programu, i nie wymagająca w związku z tym użycia słów END IF (ponieważ system wie, gdzie kończy się zakres instrukcji). Ma ona jedną z dwóch postaci:

```
IF warunek THEN instrukcje wykonywane warunkowo
IF warunek THEN instrukcje 1 ELSE instrukcje 2
```

„Instrukcje wykonywane warunkowo" oraz „instrukcje 1" są wykonywane tylko wtedy, gdy „warunek" jest spełniony. Jeżeli warunek nie jest spełniony, to są wykonywane „instrukcje 2". Instrukcja musi się mieścić w jednej linii programu (w ten sposób program tłumaczący „wie", gdzie instrukcja IF się kończy. Następna linia programu jest wykonywana niezależnie od spełnienia warunku.

Uwaga. Użycie określenia „instrukcje", a nie „instrukcja" za słowem THEN jest celowe. Wprawdzie zapisywałeś dotychczas tylko jedną instrukcję w każdej linii programu, ale możliwe jest także napisanie kilku instrukcji oddzielonych od siebie znakiem dwukropka, np.

```
CLS: PRINT "Dwie instrukcje w jednej linii": PRINT 2
```

W książce unikamy wykorzystywania tych możliwości ze względu na to, że programy są mniej czytelne.

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

```
REM odczytanie zmiennej x
IF x < 0 THEN
  PRINT x; "< 0"
  END
END IF
REM Dalsza czesc programu ...
```

Gdy wartość zmiennej x będzie ujemna, to zostaną wykonane instrukcje pomiędzy IF i END IF, wobec czego zostanie wykonana instrukcja PRINT, sygnalizująca, że zmienna x przyjęła niedozwoloną wartość, i wykonywanie programu zostanie zakończone (instrukcja END).

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 QB-8

```
REM Program QB-8  Sprawdzanie podzielnosci liczb
CLS
INPUT "Wprowadz dzielna calkowita "; dzielna%
```

```
INPUT "Wprowadz dzielnik calkowity rozny od zera "; dzielnik%
IF dzielna% MOD dzielnik% = 0 THEN
  PRINT "Liczba"; dzielna%; "dzieli sie przez"; dzielnik%
ELSE
  PRINT "Liczba"; dzielna%; "nie dzieli sie przez"; dzielnik%
END IF
```

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 QB-9

```
REM Program QB-9  rozwiazywanie rownania liniowego (algorytm 2)
CLS
INPUT "Podaj liczby a,b,c (oddzielając je przecinkami) ", a,b,c
PRINT "Rownanie"; a; "x +"; b; "="; c
IF a <> 0 THEN PRINT "ma rozwiazanie x ="; (c - b) / a
IF a = 0 AND b = c THEN PRINT "jest zawsze prawdziwe"
IF a = 0 AND b <> c THEN PRINT "jest sprzeczne"
END
```

Można także posłużyć się instrukcjami warunkowymi umieszczonymi wewnątrz innych instrukcji warunkowych.

9.6.2. Instrukcje warunkowe zagnieżdżone

Wśród instrukcji wykonywanych warunkowo mogą być także następne instrukcje warunkowe. W ten sposób jedna instrukcja warunkowa może być „zagnieżdżona" wewnątrz drugiej.

Program realizujący algorytm 2 może mieć postać:

Program QB-10

```
REM Program QB-10  rozwiazywanie rownania liniowego (algorytm 2)
CLS
INPUT "Podaj liczby a,b,c (oddzielajac je przecinkami)", a,b,c
PRINT "Rownanie"; a; "x +"; b; "="; c
IF a <> 0 THEN
   PRINT "ma rozwiazanie x ="; (c - b) / a
ELSE
```

```
    IF b <> c THEN
        PRINT "nie ma rozwiazan (jest sprzeczne)"
    ELSE
        PRINT "jest zawsze spelnione"
    END IF
END IF
```

Ćwiczenie 9-9

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.

9.6.3. Instrukcja wyboru SELECT CASE

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 instrukcję SELECT CASE. Ma ona postać:

```
SELECT CASE wyrażenie
  CASE lista wartości 1
    lista instrukcji 1
  CASE lista wartości 2
    lista instrukcji 2
  ...
  CASE lista wartości n
    lista instrukcji n
  CASE ELSE
    lista instrukcji n+1
END SELECT
```

Bieg programu jest tu rozdzielony przez wartości wyrażenia zapisanego w pierwszej linii instrukcji (za słowami SELECT 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 „lista instrukcji 1", gdy przyjmuje wartość wymienioną w „lista wartości 2", wówczas jest wykonywana „lista instrukcji 2", itd.;

natomiast w przeciwnym razie — tzn. gdy wartość „wyrażenia" nie jest objęta żadną z list wartości, wówczas jest wykonywana „lista instrukcji n+1". Użycie linii ze słowami END SELECT jest konieczne, żeby program tłumaczący wiedział, gdzie kończy się ostatnia lista. Użycie słów CASE ELSE i listy poleceń $n+1$ nie jest konieczne.

Elementy listy wartości można oddzielać przecinkami, a ich zakresy podawać ze słowem TO, np. 3 TO 7 na oznaczenie dowolnej wartości całkowitej ze zbioru: 3, 4, 5, 6, 7. Można też podawać relacje, w których zamiast wartości wyrażenia występuje słowo IS, np. `IS > 7`.

A oto fragment programu z przykładem użycia instrukcji SELECT CASE do rozróżnienia wyboru przedmiotu, którego miałby dotyczyć test. Pliki z pytaniami i odpowiedziami, dla różnych przedmiotów powinny być podobne do pliku DANE5.DAT. 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).

```
REM poprzednia część programu
SELECT CASE haslo$
  CASE "M", "m"
    instrukcje odczytania pliku z pytaniami testowymi
                                          z matematyki
  CASE "F", "f"
    instrukcje odczytania pliku z pytaniami testowymi
    ...                                   z fizyki
  CASE ELSE
    PRINT "Dla tego przedmiotu nie ma przygotowanych
                                            pytan"
    END
END SELECT
REM dalszy ciąg programu
```

9.6.4. Obliczenia wielokrotne. Pętle

W niektórych algorytmach występują pętle służące do wielokrotnego powtarzania obliczeń. Jest kilka odmian instrukcji służących do ich realizacji, spośród których użytkownik może wybierać te, które w danej sytuacji są najwygodniejsze.

Instrukcje różnią się tym, czy warunek jest badany na początku pętli czy po jej zakończeniu, a także sposobem formułowania warunku zakończenia powtarzania obliczeń w pętli.

Instrukcja DO – LOOP

Słowa DO i LOOP wyznaczają początek i koniec pętli służącej do wielokrotnego wykonywania listy instrukcji umieszczonej pomiędzy nimi. Liczba powtórzeń pętli nie jest znana z góry, lecz zależy od spełnienia warunku.

Pętla DO – LOOP z warunkiem kontynuacji WHILE

Warunek kontynuowania obliczeń określa się słowem WHILE (oznaczającym: podczas gdy), po którym podaje się odpowiednią relację, na przykład $i\% < 10$, lub bardziej złożone wyrażenie mające wartość logiczną (to znaczy prawdziwe lub fałszywe, zależnie od wartości występujących w nim zmiennych).

```
DO WHILE i% < 10
    lista instrukcji (m.in. zmiana wartości zmiennej i%)
LOOP
```

Warunek jest badany każdorazowo na początku pętli. Jeżeli jest spełniony, to instrukcje zawarte na liście są wykonywane, w przeciwnym razie system uznaje wykonanie instrukcji za zakończone i przechodzi do wykonywania instrukcji napisanej w następnej linii za słowem LOOP. Zauważ, że kiedy warunek jest badany na początku pętli, to może się zdarzyć, że instrukcje objęte pętlą nigdy nie będą wykonane.

Spróbuj napisać program realizujący algorytm 4 do zadania 8-2 (lub odczytaj go z pliku QB10.BAS).

Program QB-11

```
REM program QB-11 mnozenie podanej liczby przez 2 ... 10
                                                (algorytm 4)
CLS
INPUT "Podaj liczbe calkowita "; liczba%
mnoznik% = 2
DO WHILE mnoznik% <= 10
    PRINT "Iloczyn"; liczba%; "przez"; mnoznik%; " = ";
                                        liczba% * mnoznik%
    mnoznik% = mnoznik% + 1
LOOP
```

Ćwiczenie 9-10

A. Przeanalizuj działanie programu QB-11. Sprawdź, czy działa zgodnie z Twoimi oczekiwaniami. W jakim zakresie danych? (Spróbuj wprowadzić liczby ujemne lub liczby duże.)

B. Zmień położenie warunku WHILE z początku pętli na jej koniec (obok słowa LOOP). Zastanów się, czy nie powinieneś zmodyfikować warunku zapisanego obok słowa WHILE, żeby działanie programu nie uległo zmianie. Sprawdź to.

Pętla DO – LOOP z warunkiem zakończenia UNTIL

Warunek zakończenia obliczeń w pętli jest formułowany za pomocą słowa UNTIL (oznaczającym: aż do spełnienia), za którym podaje się relację, na przykład *decyzja$ = "n"*.

```
DO
   lista instrukcji (m.in. nadawanie wartości zmiennej decyzja$)
LOOP UNTIL decyzja$ = "n"
```

Jeżeli warunek za słowem UNTIL jest spełniony, to obliczenia w pętli zostają zakończone, i jako kolejna jest wykonywana instrukcja napisana w następnej linii za słowem LOOP (także wtedy, gdyby warunek UNTIL był napisany na początku pętli, obok słowa DO), natomiast gdy warunek nie jest spełniony, to są kontynuowane. Zatem słowo UNTIL ma tu przeciwne znaczenie do słowa WHILE.

Zauważ, że kiedy warunek jest badany na końcu pętli, wtedy instrukcje z objętej nią listy będą wykonane przynajmniej jeden raz.

Spróbuj teraz napisać program realizujący algorytm 5 do zadania 8-3 (lub odczytaj go z pliku QB12.BAS).

Program QB-12

```
REM program QB-12  suma liczb i liczba najwieksza  (algorytm 5)
CLS
suma = 0
licz% = 0
DO
    PRINT "Podaj liczbe nieujemna"
    INPUT "(liczba ujemna konczy dzialanie programu) ";
                                                liczbawprow
    IF liczbawprow >= 0 THEN
        suma = suma + liczbawprow
        licz% = licz% + 1
        IF licz% = 1 THEN najwieksza = liczbawprow
```

```
        IF liczbawprow > najwieksza THEN najwieksza = liczbawprow
        PRINT "Suma "; licz%; "liczb = "; suma
        PRINT "najwieksza liczba = "; najwieksza        PRINT
    END IF
LOOP UNTIL liczbawprow < 0
```

Uwaga. Pętlę DO można przerwać przed spełnieniem warunku zakończenia (zapisanego za słowem UNTIL) lub niespełnieniem warunku kontynuacji (zapisanego za słowem WHILE). Służy do tego instrukcja EXIT DO. Można w ten sposób zabezpieczać się np. przed utratą wyników po wprowadzeniu niewłaściwych danych.

Instrukcja FOR

Obliczenia w pętli bywają prowadzone określoną z góry ilość razy, na przykład w algorytmie 4 realizowanym za pomocą programu QB-11. W takiej sytuacji wygodniejsza w realizowaniu pętli od instrukcji DO (DO LOOP) jest instrukcja FOR. Podobnie jak w przypadku instrukcji DO pierwsze słowo kluczowe (FOR) umieszczane jest na początku pętli, drugie (NEXT) na końcu; pomiędzy nimi znajduje się lista instrukcji objętych pętlą.

W pierwszej części instrukcji, za słowem FOR, określasz, ile razy pętla ma być wykonana, podając nazwę zmiennej, która ma służyć jako licznik, jej wartość początkową (za znakiem równości), wartość końcową (za słowem TO) oraz ewentualnie wartość przyrostu, o jaki jest zwiększany w każdym kroku licznik (za słowem STEP — krok). Tak więc początek instrukcji ma postać

```
FOR licznik = wartość początkowa TO wartość końcowa
```

lub

```
FOR licznik = wartość początkowa TO wartość końcowa STEP krok
```

Na początku zmienna służąca jako licznik pętli przybiera „wartość początkową". Następnie badany jest warunek, czy wartość zmiennej „licznik" nie przekroczyła „wartości końcowej" (tzn. nie jest większa, a w przypadku ujemnej wartości „kroku" — mniejsza). Jeżeli nie, to lista instrukcji umieszczonych przed słowem NEXT zostaje wykonana. Wartość licznika jest zwiększana następnie o 1, a jeżeli w poleceniu FOR występuje słowo STEP, to jest zwiększana o wartość określoną jako „krok" (podaną za słowem STEP). Wtedy program powraca do badania warunku, czy zmienna „licznik" nie przekroczyła „wartości końcowej".

Wykonywanie listy instrukcji objętej pętlą zostaje przerwane, gdy zmienna „licznik" przekracza „wartość końcową". Jako kolejna wykonywana jest wówczas instrukcja następna po słowie NEXT.

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

Program QB-13

```
REM program QB-13 mnożenie podanej liczby przez 2 ... 10
                                                (algorytm 4)
CLS
INPUT "Podaj liczbe calkowita "; liczba%
FOR mnoznik% = 2 TO 10
    PRINT "Iloczyn"; liczba%; "przez"; mnoznik%; " = ";
                                        liczba% * mnoznik%
NEXT
```

Pętlą FOR jest objęta jedynie instrukcja PRINT. Pętla ta jest wykonywana 10 razy. Rolę licznika odgrywa zmienna całkowita *mnoznik*%. Na początku przyjmuje ona wartość 2 i po każdym obiegu pętli jest zwiększana o 1; gdy przekroczy 10, to obliczenia w pętli przestają być wykonywane. Ostatnia wartość zmiennej *i*%, dla której obliczenia w pętli są wykonywane, wynosi 10.

Uwaga. Zmiennej użytej w roli licznika obiegów pętli, wymienionej za słowem FOR, nie należy nadawać wartości wewnątrz pętli, gdyż grozi to nieprawidłowym przebiegiem obliczeń, a w szczególności powstaniem pętli, z której program bez interwencji użytkownika nie wyjdzie. Interwencja może polegać na naciśnięciu klawiszy *Ctrl-Break*.

Za słowem NEXT można umieścić nazwę zmiennej służącej jako licznik (nie jest to niezbędne do poprawnego działania programu, ale należy do dobrych zwyczajów, ułatwiających unikania błędów podczas pisania programu oraz ich znajdowania, gdy zaistniały). W programie QB-13 linia ta miałaby postać:

```
NEXT mnoznik%
```

Zauważ, że w pętlach typu DO LOOP przy omyłkowo sformułowanym warunku lub nieprawidłowo modyfikowanej zmiennej pełniącej funkcję licznika (na przykład zwiększanej zamiast zmniejszania) obliczenia mogą być wykonywane bez końca, natomiast w podstawowej instrukcji FOR, w której licznik jest z definicji zwiększany za każdym razem o 1, przypadek taki nie może mieć miejsca (jeżeli przestrzegamy zalecenia, żeby instrukcjami znajdującymi się wewnątrz pętli nie zmieniać wartości licznika). Pętla FOR dobrze się nadaje do nadawania wartości zmiennym w tablicach lub ich odczytywania.

Pętlę FOR, podobnie jak pętlę DO, można przerwać przed osiągnięciem przez licznik właściwej wartości końcowej. Służy do tego instrukcja EXIT FOR.

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, np. o 0,5, o 0,1. Masz do wyboru dwa sposoby użycia pętli FOR:

- użyć jako licznika zmiennej całkowitej (zwiększanej o 1), i kolejne wartości zmiennej niezależnej x obliczać jako wyrażenia zależne od wartości licznika, np. $x = 2 + 0.1 * licznik\%$;
- użyć jako licznika zmiennej rzeczywistej x, zwiększanej o tyle, ile wynosi wartość kroku (STEP *krok*).

W programie QB-14 zastosowano drugie z wymienionych rozwiązań do obliczania wartości funkcji $f(x) = -2x^2 + 8x$ na przedziale od 2 do 3 z krokiem 0,01.

Program QB-14

```
REM Program QB-14  -  petla FOR z krokiem niecalkowitym
CLS
FOR x = 2 TO 3 STEP .01
    PRINT x, -2 * x * x + 8 * x
NEXT x
PRINT
PRINT "Po zakonczeniu petli x = "; x
END
```

Zwróć uwagę na wartości przyjmowane przez zmienną niezależną x. Nie są one dokładnie równe tym liczbom, jakim powinny. Po stukrotnym wykonaniu dodawania 0,01 do 2 wynik nie jest równy dokładnie 3 (konkretne wartości mogą zależeć od komputera). Ostatnia wartość zmiennej niezależnej jest mniejsza od wartości końcowej pętli, a następna większa. Przy dodawaniu stu liczb błąd nie jest duży, ale przy dodawaniu większej ich liczby może urosnąć. Poza tym musisz się liczyć z tym, że wartość funkcji nie zostanie obliczona dla wartości końcowej, gdy zmienna x zamiast być jej równa będzie od niej (nieznacznie) większa.

Ćwiczenie 9-11

Zorientuj się, jak szybko i jak dokładnie Twój komputer dodaje liczby rzeczywiste. W tym celu zmodyfikuj program QB-14 przez usunięcie

instrukcji PRINT objętej pętlą i zmniejszenie kroku do 0,001 lub nawet do 0,0001.

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 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 QB-15 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 QB-15

```
REM Program QB-15 - tabliczka mnozenia (zagniezdzone petle FOR)
CLS
FOR i% = 2 TO 5
  FOR j% = 2 TO 7
    iloczyn% = i% * j%
    PRINT i%; " x "; j%; " = "; iloczyn%
  NEXT j%
NEXT i%
END
```

Zewnętrzna pętla FOR zaczyna się w trzeciej i kończy w ósmej linii programu (takiego jak jest przedstawiony w książce); jako licznik służy w niej zmienna całkowita i%. Wewnętrzna pętla FOR zaczyna się w czwartej i kończy w siódmej linii programu; jako licznik służy w niej zmienna j%.

Ważne jest, że wewnętrzna pętla kończy się w całości przed zakończeniem zewnętrznej. Przy słowach NEXT kończących pętle umieszczono nazwy zmiennych pełniących funkcję licznika w tych pętlach, którym odpowiada dana instrukcja NEXT.

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 9-12

A. Zmodyfikuj program QB-15 przez zmianę wartości początkowej zmiennej $j\%$ z 2 na $i\%$ (możesz odczytać program z pliku QB15I.BAS). 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.

9.6.5. Instrukcja skoku. Etykiety

Do zmiany kolejności wykonywania poleceń 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*). W systemie QBasic etykietę stanowi słowo zakończone dwukropkiem umieszczone na początku linii programu lub liczba całkowita (bez dwukropka). Każda etykieta może wystąpić w programie tylko w jednym miejscu.

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

Uwaga. Pisząc swoje programy stosuj instrukcje strukturalne w rodzaju pętli DO i FOR oraz instrukcji warunkowych (a nie stosuj instrukcji skoku).

9.6.6. 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 QB-8). Spróbuj napisać program realizujący

ten algorytm. Sprawdź jego działanie. W razie trudności posłuż się programem QB-16.

Program QB-16

```
REM Program QB-16 - rozkladanie liczby na czynniki pierwsze
CLS
INPUT "Wprowadz liczbe calkowita dodatnia"; n%
IF n% < 1 THEN END
czynnik% = 2
iloraz% = n%
DO UNTIL iloraz% = 1 OR czynnik > n%
    IF iloraz% MOD czynnik% = 0 THEN
        iloraz% = iloraz% \ czynnik%
        PRINT "Kolejny czynnik liczby"; n%; " = "; czynnik%
    ELSE
        czynnik% = czynnik% + 1
    END IF
LOOP
```

Sprawdź działanie programu na różnych danych. Posłuż się pracą krokową lub innymi instrukcjami debugera, żeby „podejrzeć" jego działanie.

Zmodyfikuj podany program tak, by realizował algorytm 8 do zadania 8-6. Możesz też odczytać go z pliku i przeanalizować jego działanie (posługując się pracą krokową).

Program QB-17

```
REM Program QB-17 - wyznaczanie NWD i NWW dwoch liczb naturalnych
CLS
INPUT "Wprowadz liczbe calkowita dodatnia"; m%
INPUT "Wprowadz liczbe calkowita dodatnia"; n%
IF m% < 1 OR n% <1 THEN END
czynnik% = 2
ilorazm% = m%
ilorazn% = n%
nwd% = 1
DO UNTIL ilorazm% = 1 OR ilorazn% = 1 OR czynnik% > m%
                                      OR czynnik% > n%
    IF ilorazm% MOD czynnik% = 0 AND ilorazn% MOD czynnik%
                                              = 0 THEN
        ilorazm% = ilorazm% \ czynnik%
        ilorazn% = ilorazn% \ czynnik%
        nwd% = nwd% * czynnik%
        PRINT "Kolejny czynnik liczb"; m%; "i"; n%; "=";
                                                  czynnik%
```

```
    ELSE
        czynnik% = czynnik% + 1
    END IF
LOOP
nww% = (m% / nwd%) * n%
PRINT "NWD liczb"; m%; " i "; n%; " = "; nwd%
PRINT "NWW liczb"; m%; " i "; n%; " = "; nww%
```

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

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

Rzeczywiście byłoby to bardziej naturalne, zmiana została jednak podyktowana zakresem liczb całkowitych. Uruchom program dla danych 480 i 720. Czy otrzymałeś NWD = 240 i NWW = 1440? Zmień teraz w programie instrukcję obliczania NWW na podaną wyżej i uruchom program dla tych samych danych. Czy otrzymałeś wynik? Powinieneś otrzymać komunikat o powstaniu nadmiaru w instrukcji obliczania NWW. 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).

Jeżeli jednak zechcesz posługiwać się programem QB-17 dla większych danych, to i tak może się zdarzyć, że wystąpi nadmiar. W takiej sytuacji zmień typ liczb całkowitych na *long-integer*. Możesz zmienić tylko typ zmiennej *nww*%, ale możesz zmienić również typ pozostałych zmiennych całkowitych (w programie QB-17l zamieszczonym w pliku QB17L.BAS został zmieniony typ wszystkich zmiennych).

Ćwiczenie 9-13

Napisz program realizujący algorytm 9 (Euklidesa) i porównaj otrzymywane wyniki z otrzymanymi za pomocą programu QB-17. Jeżeli różnica w czasie obliczeń jest zauważalna, to sprawdź, który jest szybszy.

Uwaga. Łatwiej zauważysz różnicę szybkości dla dużych danych, zatem dobierz odpowiedni typ zmiennych w obu programach.

9.7. Tablice

9.7.1. Deklarowanie tablic

Oprócz wymienionych zmiennych prostych możesz używać tablic składających się ze zmiennych wymienionych typów, musisz jednak je zadeklarować. Po słowie kluczowym DIM (*dimension* — rozmiar) musisz podać nazwę