Elementarz programowania
1. QBASIC

Cel ogólny: Podstawowe zasady programowania w QBASIC

Część 2

Wprowadzanie danych do programu, wyprowadzanie wyników. Tablice. Instrukcje warunkowe: IF, CASE.
Iteracje: powtarzaj, dopóki, dla.
Funkcje operujące na znakach.
Podprogram a funkcja. Składnia podprogramu.

Przekazywanie zmiennych między podprogramami.
Deklaracja Common Shared. Funkcja.

Spis treści

Wprowadzanie danych do programu

- Najbardziej elementarnym sposobem wprowadzania danych jest wpisywanie ich za pomocą klawiatury.
  QBasic posiada instrukcje **INPUT**, przewidziana do tego celu.
  By zapoznać sie ze sposobem posługiwania sie ta instrukcja wpisujemy do edytora programu QBasic **INPUT** i naciskamy klawisz **F1.**
  Naciskamy dwukrotnie klawisz **F6** i za pomocą klawiszy kierunkowych możemy obejrzeć cały tekst pomocniczy dla słowa **INPUT**.
- **INPUT** reads input from the keyboard or a file.
  LINE INPUT reads a line of up to 255 characters from the keyboard or a file.

INPUT

- Wiersz
  INPUT [;] [„zgłoszenie"{; | ,}] listazmiennych
  należy rozumieć tak:
- **INPUT** jest elementem, który musi wystąpić w instrukcji,
  po słowie INPUT możemy, ale nie musimy wpisać średnik (;), następnie możemy ale nie musimy wpisać dowolny tekst zamknięty cudzysłowami i zakończony średnikiem (;) albo przecinkiem (,),
  na końcu musi sie znajdować **nazwa zmiennej** lub **kilka nazw** rozdzielonych przecinkami.

Instrukcja **INPUT** ma postać:

INPUT "*tekst żądania danych*"; *zmienna*

- np. INPUT "podaj wartość siły:"; sila
- lub INPUT "siła="; sila
- Tekst zawarty w cudzysłowach będzie wyświetlony aby użytkownik wiedział, że ma wpisać odpowiednią wartość.
  Wpisana wartość zostanie wstawiona do zmiennej, której nazwę podano w tej instrukcji.

Przykłady **INPUT**

Najprostsza instrukcja czytania danych z klawiatury ma postać: **INPUT a12**

Wykonanie tej instrukcji spowoduje wyświetlenie na ekranie znaku zapytania (?) i program będzie czekał na wpisanie z klawiatury stałej, która ma być przypisana do zmiennej o nazwie a12.

Wpisanie stałej kończymy wciśnięciem klawisza Enter.

Jeśli wpisana z klawiatury **stała** nie jest tego typu co zmienna, do której ma być przypisana, to możliwe są 2 przypadki:

a) nastąpi konwersja typu stałej do typu zmiennej i przypisanie tej stałej do zmiennej a12

b) wykonanie programu zostanie zatrzymane z powodu niemożności dokonania konwersji typu.

Przykłady INPUT c.d.

- Instrukcja
  INPUT liczba1, liczba2, liczba3
  zostanie wykonana po wpisaniu z klawiatury 3 liczb rozdzielonych przecinkami i wciśnięciu Enter. W instrukcji INPUT powinno sie zawsze umieszczać informacje dla operatora, jakie wielkości ma wprowadzić.
- Przykładem może być:
  INPUT "Podaj 3 liczby z przedziału 0 do 100"; n1, n2, n3

LINE INPUT

- Instrukcja **LINE INPUT** służy do wprowadzenia **jednej stałej znakowej, która może zawierać w sobie przecinki.**
  W instrukcji INPUT przecinek jest znakiem oddzielającym stale i nie może być wczytany do zmiennej.
  W instrukcji LINE INPUT dopiero [Enter] jest znakiem końca stałej i równocześnie sygnałem do jej przypisania do zmiennej umieszczonej w tej instrukcji.
- Np. instrukcja
  LINE INPUT "Podaj imiona rodzeństwa "; imiona$
  umożliwia przypisanie do zmiennej znakowej imiona$ kilku grup znaków (np. imion) rozdzielonych przecinkami.
  QBasic jest tak napisany, ze wpisanie z klawiatury stałej, która nie może być przypisana do zmiennej, umieszczonej w instrukcji INPUT lub LINE INPUT, nie zatrzymuje wykonywania programu, tylko powrót do początku wykonania instrukcji czytania danych.

Kontrola wprowadzania w INPUT

- Przykład:
  INPUT "Podaj liczbe uczniow w klasie ";
  liczbauczniow%
  i wpisujemy (zamiast liczby) słowo trzydzieści.
  Na ekranie pojawia sie komunikat
  Redo from start
  co znaczy - wykonaj od poczatku i program czeka na ponowne wprowadzenie danej lub danych.

Input do operacji dyskowych

- Instrukcje
  INPUT #filenumber, variablelist
  LINE INPUT #filenumber%, variable$
  służą do wczytania danych znajdujących się w zbiorze zapisanym na dysku. Wymagają one uprzedniego polaczenia programu z danym zbiorem danych.

Przykład

```
OPEN "DANE.TXT" FOR INPUT AS #1 'Otwarcie pliku do odczytu z numerem 1
PRINT "Tekst pliku:"
DO WHILE NOT EOF(1)        'Pętla działa aż do napotkania znacznika końca pliku 1
    LINE INPUT #1, LINIA$  'Wczytuje linię tekstu z pliku
    PRINT LINIA$           'Wyświetla wczytaną linię tekstu na ekranie
LOOP                                    'Koniec pętli
CLOSE #1                                'Zamknięcie pliku
```

DATA i READ - umieszczanie danych w programie

- Jedna z metod wprowadzenia danych do programu jest umieszczenie ich w tym programie na stale.
  Dane umieszcza sie w wierszach rozpoczynających sie słowem kluczowym **DATA.**
  Dane te przypisywane są do zmiennych instrukcja **READ**.

Przykładem takiego rozwiązania jest fragment programu

```
REM  DATA...READ
CLS
DATA 12, 23, 45, 74, 86, 345, 11
READ a1, a2, a3, a4, a5, a6, a7
PRINT A1, a2, a3, a4, a5, a6, a7
```

którego wykonanie spowoduje przypisanie liczb 12, 23,..11 do zmiennych a1, a2, ...a7.

- Dane z instrukcji DATA mogą być czytane tylko raz.
  Dane te można jednak odtworzyć instrukcja **RESTORE**.
  Instrukcja DATA może znajdować sie w dowolnym miejscu programu, na przykład na końcu.

Wyprowadzanie wyników: PRINT, LPRINT

- Dwie podstawowe metody wyprowadzania wyników działania programu to wyświetlenie ich na **ekranie** monitora **PRINT**
  lub **wydrukowanie** na drukarce: **LPRINT**
- Trzeci sposób to zapisanie wyników do **zbioru** na dysku, celem późniejszego zapoznania sie z nimi - wyświetlenia na ekranie i ewentualnego wydrukowania.
  Zapisywanie wyników na dysku jest w wielu przypadkach etapem pośrednim realizacji dużego zadania za pomocą programów komputerowych.
  Jeden program wykonuje jakiś etap obliczeń, zapisuje wyniki do zbioru na dysku, a inny program wczytuje zawartość tego zbioru, która stanowi dla niego dane wejściowe.

Wydruk wyników na ekran: PRINT, PRINT USING

- **PRINT**

  Np.

  PRINT „Obliczenie azymutu”

  PRINT x, y

- **PRINT USING** <wyrażenie wyświetla łańcuchy lub liczby z użyciem określonego formatu

  Np. **PRINT USING „####.##”; x**

  - wydrukuje wartość zmiennej x, 4 miejsca na część całkowitą, 2 miejsca po przecinku

Instrukcje wyprowadzania wyników

- **PRINT [lista wyrażeń]** – wyświetlanie
- **LPRINT [lista wyrażeń]** – wydruk na drukarkę
- np.
  PRINT wyn1, wyn2, wyn3 wydruk na ekran
  w której wyn1, wyn2, wyn3 to nazwy zmiennych, w których zapamiętane zostały wyniki działania programu.
  Wydrukowanie wymaga instrukcji
  LPRINT wyn1, wyn2, wyn3 – na drukarkę
  wymaga drukarki podłączonej do komputera, lub włączonej do sieci i odpowiednio przygotowanej
- Zamiast **PRINT** można pisać znak **pytajnika ?**

PRINT USING

- **PRINT USING <wyrażenie** wyświetla łańcuchy lub liczby z użyciem określonego formatu
- Jest to taki wariant instrukcji **PRINT** , który pozwala wyświetlać wyniki w równych słupkach o określonej ilości cyfr przed i po kropce dziesiętnej. Można przetłumaczyć jej znaczenie jako: **"drukuj używając podanego szablonu formatu"**.
- W najprostszym przypadku ten szablon formatu to zawarty w cudzysłowach ciąg tylu znaków # ile cyfr chcemy otrzymać odpowiednio przed i po kropce dziesiętnej.

Formaty łańcuchowe:

- **"!"** - tylko pierwszy znak łańcucha będzie wyświetlany
- print using "\ n spacji\" wyświetlonych będzie n+2 znaków łańcucha. Jeśli n+2 będzie większe od długości łańcucha to z prawej strony dodawane będą spacje
- **"&"** wyświetlony będzie cały łańcuch
- Formaty liczb:

  # reprezentuje każda wyświetlaną cyfrę w tekście

Przykłady PRINT USING

- Przykład 1

```
a = 123.456
PRINT USING "###.##"; a
a$ = "ABCDEFG"
PRINT USING "!"; a$
PRINT USING "\ \"; a$
```

- Przykład 2

```
CLS
PRINT "Obliczanie kwadratów liczb"
PRINT "liczba", "kwadrat"
FOR y = 0 TO 100 STEP 9.05
PRINT USING "######.####"; y; y^2
NEXT y
```

Program QBPR1.BAS

```
REM  QBPR1.BAS  - Nazwa programu
REM Wczytywanie liczb, znaków, sumy, wydruk – komentarz ‘
‘odnośnie funkcji programu
CLS   ' Kasowanie ekranu, REM lub apostrof oznacza komentarz
DEFDBL A-L ' Deklaracja zmiennych A-L jako Long
DEFSTR P-Z ' Deklaracja zmiennych A-L jako String
INPUT "Wpisz dowolna liczbe "; lb1 ' Wprowadzenie liczby
INPUT "wpisz inna liczbe "; lb2
INPUT "Wpisz znaki "; znak1 ' Wprowadzenie znaku
INPUT "wpisz znaki "; znak2
lb3 = lb1 + lb2
znak3 = znak1 + znak2
PRINT lb1, lb2, lb3 ' Wydruk na ekranie
PRINT znak1, znak2, znak3
PRINT lb1; lb2; lb3
PRINT znak1; znak2; znak3
PRINT "lb1="; lb1, "lb2="; lb2, "lb3=lb1+lb2="; lb3
END ' Koniec programu
```

Program QBRPR2.BAS

```
REM QBRPR2.BAS
REM Wydruk liczb z uzyciem formatowania
CLS
DEFDBL A-L
a = 100
b = 3
c = a / b
PRINT c, c, c
PRINT
PRINT c; c; c
PRINT
PRINT USING "###.###"; c; c; c
PRINT
PRINT USING "+###.####"; c; c; c
PRINT USING "+########.####"; c; c; c
PRINT USING "+#######.####"; c; c; c
PRINT USING "+##########.####"; c; c;
```

Zapis do pliku

Przykład programu zapisującego wyniki do pliku WYNIKI.TXT:

```
CLS
PRINT "Drukowanie do pliku WYNIKI.TXT"
OPEN „WYNIKI.TXT" FOR OUTPUT AS #1
    PRINT #1, "Obliczanie kwadratow liczb"
    PRINT #1, "liczba", "kwadrat"
    FOR y = 0 TO 100 STEP 9.05
        PRINT #1, USING "#######.####"; y; y ^ 2
    NEXT y
CLOSE #1
```

- Sprawdź jego działanie - otwórz plik wyników w NOTATNIKU i przejrzyj.

Tablice

- **DIM <lista>** definiuje maksymalne wartości indeksów tablic i rezerwuje na nie odpowiednia ilość pamięci
  - Np. **DIM A(20)** lub **DIM cena(20,30)**
    DIM cena(1 TO 1000) - liczby rzeczywiste zwykłej precyzji
  - lub **DIM** TXY**#(2, 20)** - tablica 2 wymiarowa, podwójna precyzja bo **#** na końcu nazwy
  - **DIM** Nazwiska**$(50) –** rezerwacja tablicy na 50 nazwisk
  - **DIM** polozenie**%(1 TO 10, 1 TO 20, 1 TO 30)**
    – tablica na liczby całkowite, 3 wymiarowa

Instrukcje warunkowe: IF ... ENDIF oraz SELECT CASE

W BASICU mamy możliwość skorzystania z dwóch typów instrukcji warunkowych.

Można spotkać następujące typy instrukcji **IF**:

Jednolinijkowa:

IF warunek THEN polecenie

Wielolinijkowa:

IF warunek THEN Polecenie 1: Polecenie 2: ... Polecenie n END IF

Wielolinijkowa z alternatywą:

```
IF warunek THEN  Polecenie1  (gdy wartość logiczna = "prawda")
ELSE Polecenie2  (gdy wartość logiczna ="fałsz")
END IF
```

Zagnieżdżona: gdy wewnątrz jednej struktury **IF ... ENDIF** znajduje się inna struktura

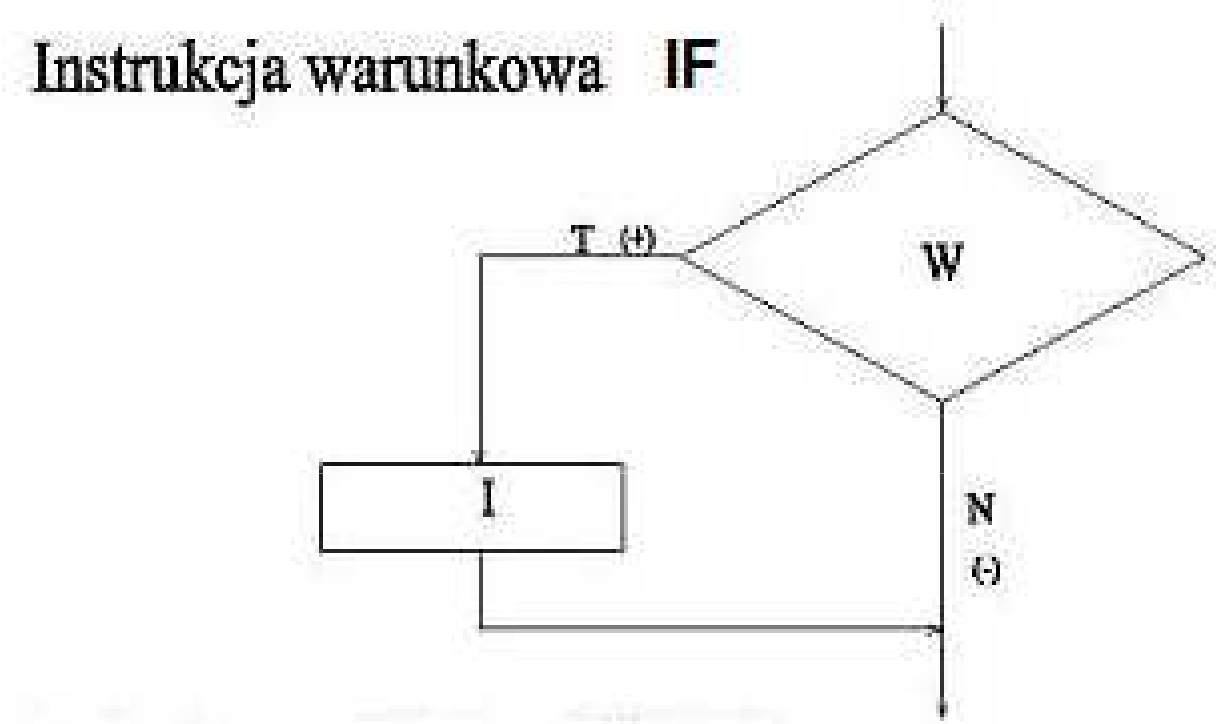
Instrukcja SELECT CASE wylicza dla testowanej zmiennej jej możliwe wartości, i dla każdej przypisuje grupę instrukcji:

```
SELECT CASE TestowanaZmienna
CASE PierwszaWartość :  Pierwsza grupa instrukcji
CASE DrugaWartość     :  Druga grupa instrukcji
...
END SELECT
```

Instrukcje decyzyjne (z wyborem)

Instrukcja Warunkowa:

IF W then I lub **IF W THEN blok_instrukcji END IF**



Konwencja notacyjne - pseudokod jeśli W to I;

Basic	if W then I IF W THEN blok_instrukcji END IF

IF warunek THEN instrukcja:

Najprostsza forma instrukcji warunkowej jest:

IF warunek THEN instrukcja

w której warunek jest wyrażeniem logicznym

- Przykład

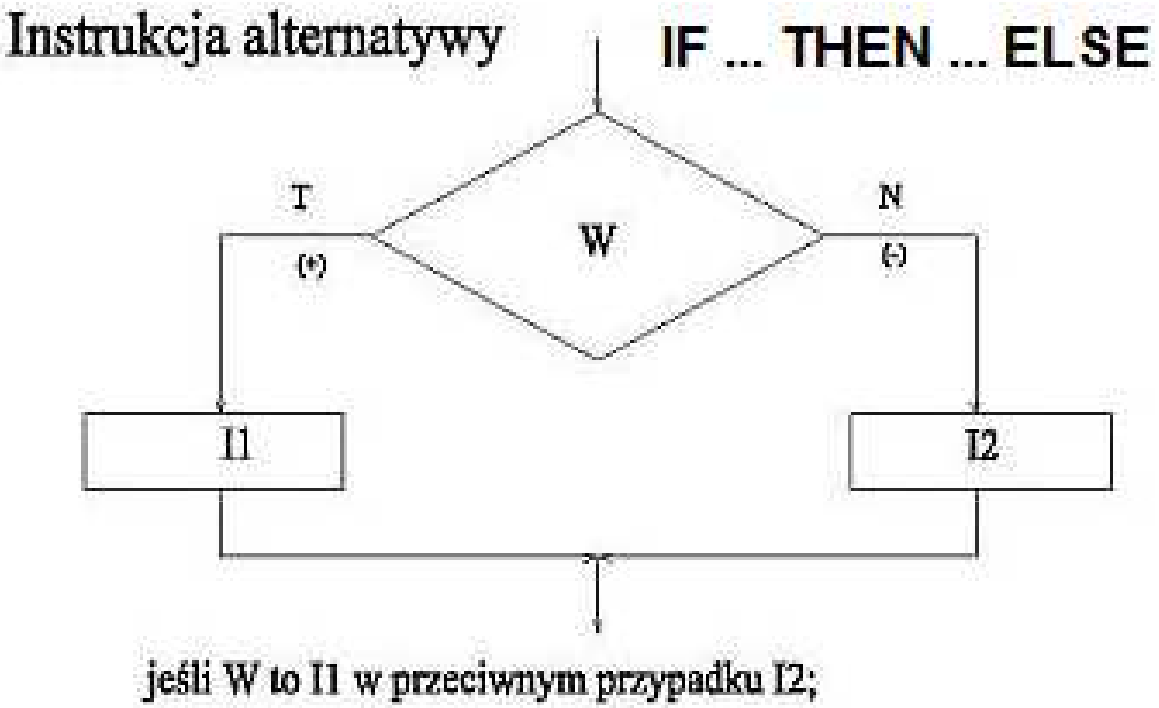
```
INPUT "Podaj liczby a,b "; a, b
IF a<b THEN PRINT a;"<";b
IF a>b THEN PRINT a;">";b
```

IF warunek THEN blok END IF

- Ten wariant umożliwia wykonanie całego **bloku instrukcji** w przypadku gdy warunek jest prawdziwy.
- Przykład

```
INPUT "Podaj liczby a,b "; a, b
IF a > b THEN
  wynik = a - b
  PRINT "a-b="; wynik
END IF
```

Instrukcja Alternatywy:
IF W THEN I1 ELSE I2 END IF



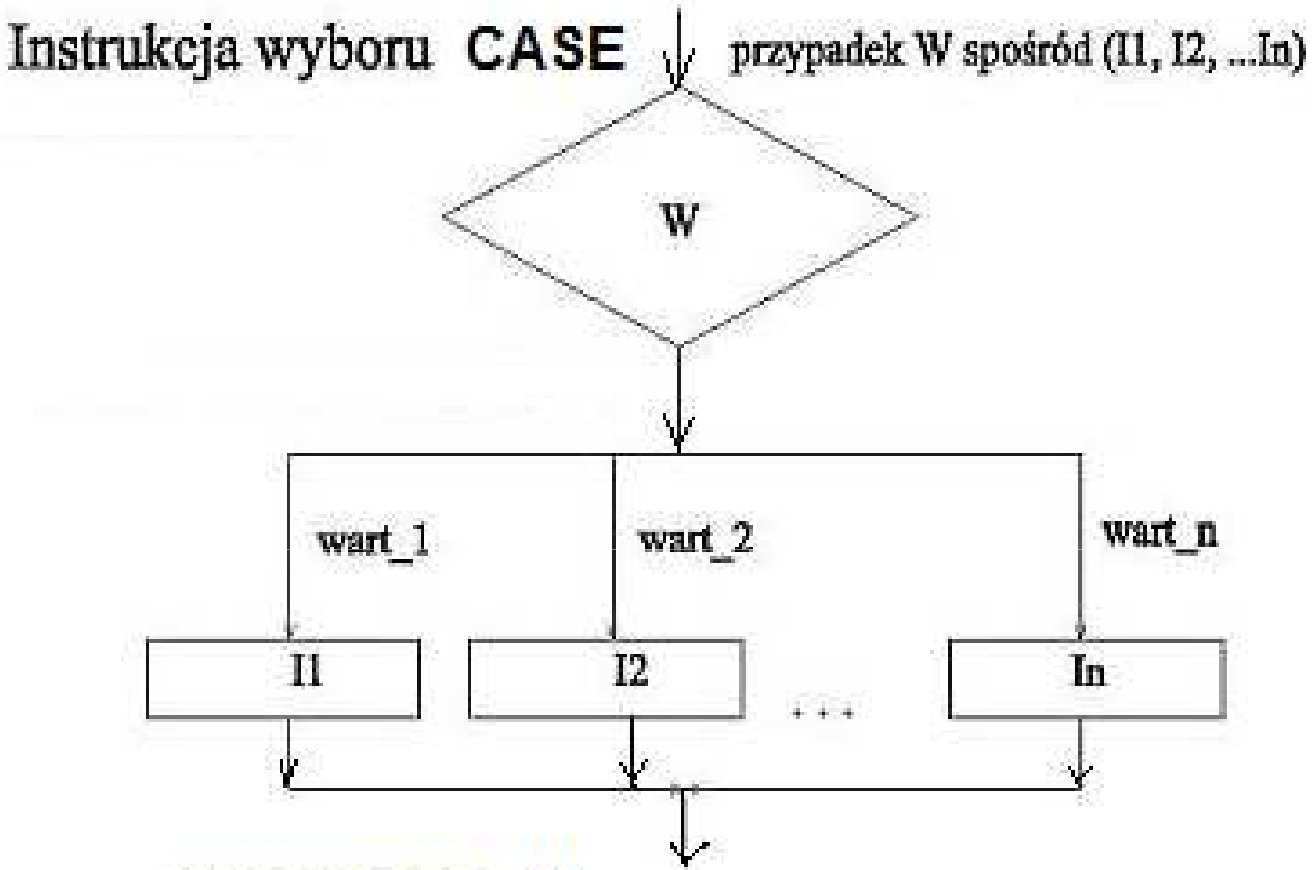
IF W THEN I1 ELSE I2 END IF
IF W THEN blok-instr1 ELSE blok_instrI2 END IF

W - warunek, I - instrukcja,
blok - blok instrukcji - kilka instrukcji

Przykład programu z instrukcją IF … THEN … ELSE … END IF

```
CLS
 INPUT "Podaj liczbe z przedzialu 1 do 20 "; liczba
 IF liczba >= 1 AND liczba <= 20 THEN
   PRINT "Prawidlowo"
 ELSE
   PRINT "Liczba spoza przedzialu"
 END IF
 PRINT "Koniec programu"
```

Instrukcja Wyboru CASE – przypadek z kilku




```
SELECT CASE  W
 CASE wart_1  I1
 CASE wart_2  I2
 CASE wart_n  In
 CASE ELSE  Instrukcja awaryjna
END SELECT
```

Przykłady z CASE: 1) - kolory

```
var$="blue”
Input „Podaj kolor”; var$
select case var$
   case "red”, „czerwony”
      print "red - czerwony"
   case "green","yellow” „black”
      print "green or yellow or black ”
case „blue”, „niebieski”
      print „blue - niebieski"
   case else
       print "color unknown - nieznany"
end select
```

Przykład z CASE 2) - liczby

```
DIM num AS INTEGER
INPUT "Wprowadz liczbe calkowita  "; num
SELECT CASE num
CASE 1
 PRINT "one"
 CASE 2
 PRINT "two"
 CASE 3
 PRINT "three"
 CASE ELSE
 PRINT "other number"
END SELECT
```

Pułapki - ON ERROR, instrukcja CASE

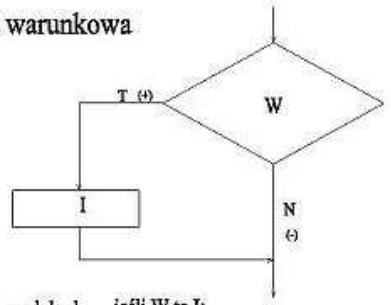
```
REM ABCQ12.BAS Sprawdzanie rodzaju bledu: brak gotowości drukarki, dzielenie przez zero, inny
CLS
ON ERROR GOTO label1 ‘ skok do etykiety label1
DEFINT A-D
INPUT "Podaj promień "; r
 obw = 2 * 3.14 * r:   pole = 3.14 * r ^ 2:  ‘ Obliczenia
 LPRINT "Promień="; r, " obwod = "; obw, "; pole = "; pole;  ‘ Wydruk na drukarkę
 w = pole / (r - r)
END

label1:  ‘ etykieta
PRINT "Blad nr "; ERR
 SELECT CASE ERR
  CASE 25
   PRINT "Drukarka nie jest gotowa.  Włącz ja i Enter"
   INPUT d$
   RESUME
 CASE 11
  PRINT "Dzielenie przez zero - Enter by zakończyć"
  INPUT d$
 CASE ELSE
  PRINT "Nieznany błąd - Enter by zakończyć"
  INPUT d$
 END SELECT
```

Obliczenie azymutu - ćwiartki

- Program az2.bas Obliczenie azymutu ze wspolrzednych - metoda czwartaków

```
 pi = 4.0*atn(1.0) ' Obliczenie Pi
 rg=200/pi        ' Ro gradowe
 rs=180/pi        ' Ro stopniowe
Input "x1 " ;x1  : input "y1 " ; y1: input "x2 "; x2:  input "y2 "; y2
dx=x2-x1      : dy=y2-y1
print "dx=";dx:  print "dy=";dy
if dx=0 then       ' Warunek gdy DX=0
  if dy> 0 then
    az = pi/2
   else
   az=1.5*pi
  end if
else             ' Gdy DX <> 0
czw= atn(dy/dx)    ' Obl. czwartaka w radianach
czw=abs(czw): czwg=czw*rg       ' Zamiana na grady
czws=czw*rs       ' Zamiana na stopnie
print "pi=";pi     ' Wydruk Pi
print "czwart=";czwg; "[grad] ="; czws; "[st]„
if dx>0 and dy>0 then cw=1: if dx<0 and dy>0 then cw=2: if dx<0 and dy<0 then cw=3: if dx>0 and dy<0 then cw=4
select case cw       ' Wariant ćwiartki
 case 1
  az=czw             ' ćwiartka I
  print "case="; cw
 case 2              ' ćwiartka II
   az=pi-czw
   print "case=";cw
 case 3
   az=pi+czw
    print "case=";cw
 case 4
   az=2*pi-czw       ' ćwiartka IV
    print "case=";cw
 end select
end if
azg=az*rg       ' Zamina na grady
azs=az*rs       ' Zamiana azymutu na stopnie dziesietne
print "dx = ";dx: print "dy="; dy: print "azg[grad]=";azg: print "azs[st]  =";azs
```

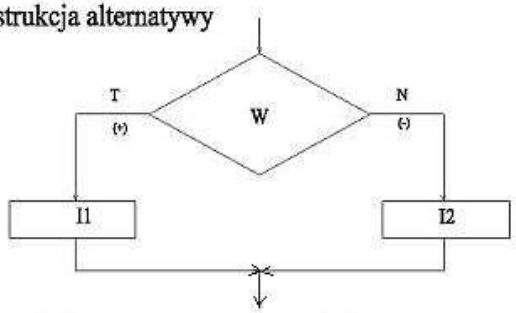
Instrukcja warunkowa



Konwencja notacyjne - pseudokod jeśli W to I;

Pascal	if W then I;
C i C++I	if (W) I;
Basic	if W then I IF W THEN blok_instrukcji END IF

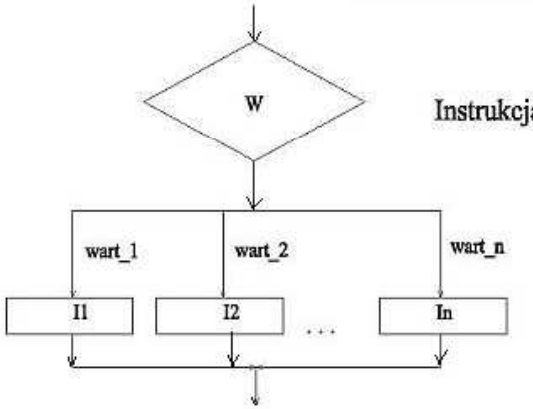
Instrukcja alternatywy



jeśli W to I1 w przeciwnym przypadku I2;

```
if W then I1 else I2;
if (W) I; else I2;
IF W THEN I1  ELSE I2  END IF
IF W THEN blok-instr1  ELSE blok_instrI2  END IF
```

Instrukcja wyboru



przypadek W spośród (I1, I2, ...In)

Pascal

```
case  W of
wart_1: I1;
wart_2: I2;
wart_n: In;
else Instrukcja_awaryjna
end;
```

C i C++I

```
switch (W) {
 case wart_1: I1;
 case wart_2: I2;
 case wart_n: In;
default Instrukcja_awaryjna
}
```

```
switch (W) {
 case wart_1: I1;  break
 case wart_2: I2;  break
 case wart_n: In;  break
default Instrukcja_awaryjna; break
}
```

Basic

```
select case  W
 case wart_1
        I1
 case wart_2
        I2
 case wart_n
        In
 case else
   Instrukcja_awaryjna
end select
```

Instrukcje z wyborem

Iteracje - Pętle: powtarzaj (do until), dopóki (while), dla (for)

- **Pętla** (**loop**) to fragment programu, który jest wykonywany wielokrotnie
- Basic posiada możliwość użycia szeregu rodzajów pętli dzielących się na
  - **pętle warunkowe**:
    - **Powtarzaj** Instrukcje aż do Warunku (*jeśli warunek spełniony to wyjdź) –> **DO UNTIL***
    - **Dopóki** Warunek wykonuj Instrukcje *(jeśli warunek nie spełniony to wyjdź) –> **DO WHILE***
  - **pętle z licznikiem**: **dla –> *FOR … NEXT***
    dla licznika i od i1 do i2 wykonuj Instrukcje

INSTRUKCJE ITERACYJNE

- **Iteracja** - jest to metoda matematyczna polegająca na wielokrotnym kolejnym zastosowaniu tego samego algorytmu postępowania, przy czym wynik i-tej operacji stanowi dane wejściowe dla kolejnej, (i+1)-szej operacji.
- Służą do ograniczenia cykli programowych, tj. wielokrotnego wykonywania pewnych sekwencji instrukcji.
- W języku Basic występują trzy instrukcje iteracyjne.
  - instrukcja [dla] **FOR**,
  - instrukcja [dopóki] **DO** ... **WHILE**,
  - instrukcja [powtarzaj] **DO UNTIL**

Pętle warunkowe

- Pętle warunkowe są wykonywane dopóki jest prawdziwy zadany warunek logiczny
  DO...WHILE (dopóki)
- bądź tak długo dopóki warunek logiczny nie zajdzie **DO...UNTIL** (powtarzaj)
- Za każdym przebiegiem pętli sprawdzany jest warunek logiczny i w zależności od tego czy wynikiem będzie "prawda", czy też "fałsz" podejmowana jest decyzja o kontynuowaniu bądź przerwaniu pętli.

Pętla **DO...WHILE**

Pętla **DO...WHILE** może mieć postacie:

DO WHILE Warunek Instrukcje **LOOP** ‘ warunek sprawdzany na początku

WHILE Warunek instrukcje **WEND** ‘warunek sprawdzany na początku

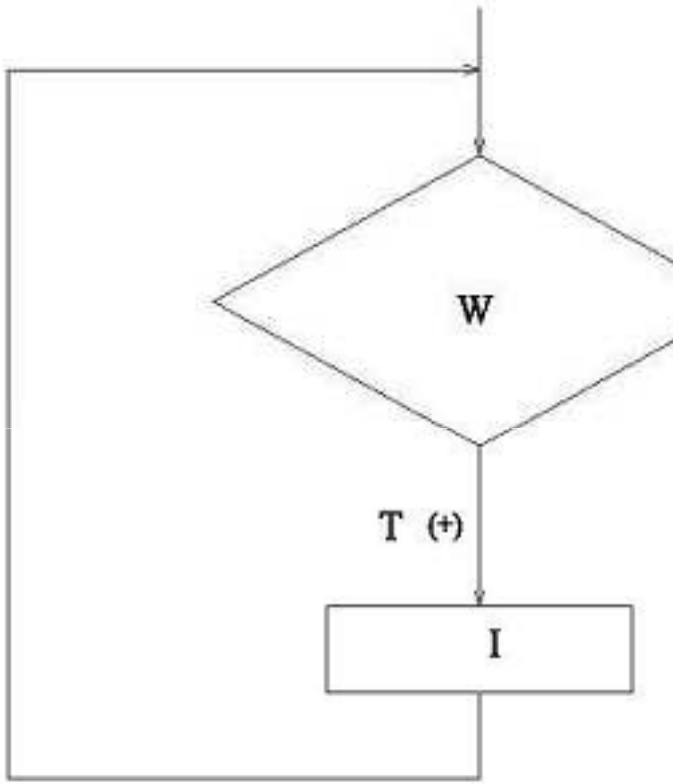
lub

DO Instrukcje **LOOP WHILE** Warunek ‘ warunek sprawdzany na końcu

- Różnica wynika z miejsca, gdzie jest sprawdzany warunek.
  Wynikiem jest to, że instrukcje pętli z warunkiem na początku mogą nie wykonać się ani razu,
  podczas gdy w przypadku pętli z warunkiem w środku lub na końcu
  zawsze nastąpi przynajmniej jednokrotne wykonanie instrukcji wnętrza pętli.

Schemat instrukcji DOPÓKI – DO WHILE

dopóki W wykonuj I;



DO WHILE warunek
instrukcja_1
.....
instrukcja _n
LOOP

Np.

```
DO WHILE k<=20
  k = k + 1
  PRINT k
LOOP
```

Pętla iteracji "Dopóki W wykonuj I"

Warunek sprawdzany na początku – może się ani raz nie wykonać

DO WHILE W
instrukcje
LOOP

Lub

WHILE W
Instrukcje
WEND

Lub

DO
instrukcje
LOOP WHILE W

Pętla "**Powtarzaj Instrukcje aż do Warunku**"
DO … UNTIL

- Pętla **DO..UNTIL** ma dwie postacie:

  DO UNTIL Warunek *‘ pierwsza postać – warunek sprawdzany na początku*

  Instrukcje

  LOOP

- lub

  DO *‘ druga postać – warunek sprawdzany na końcu*

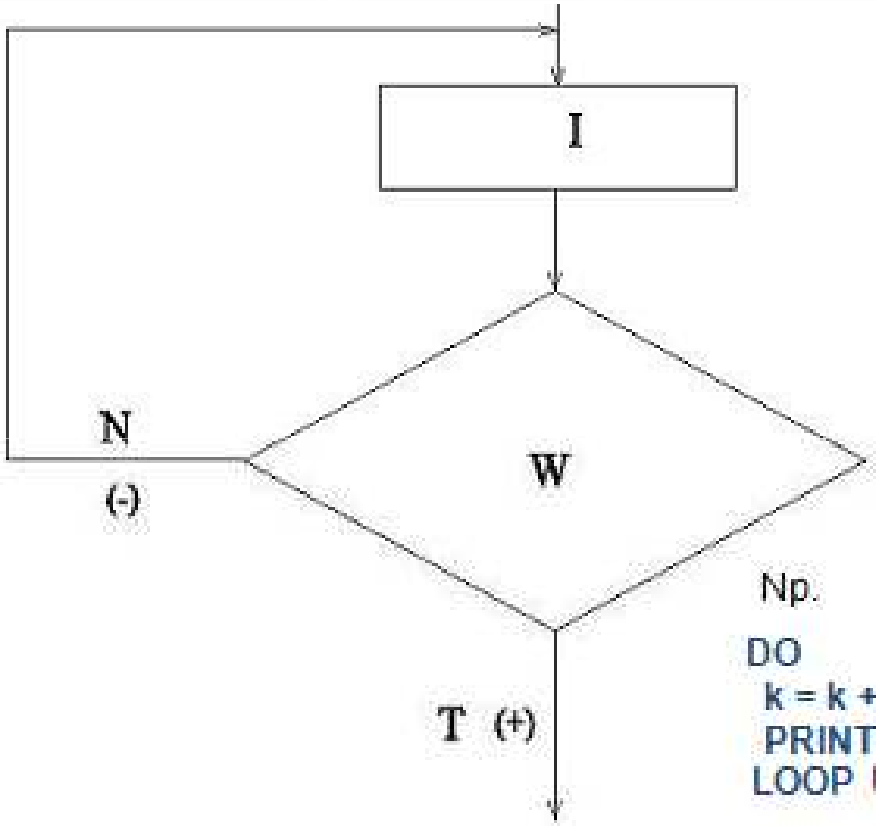
  Instrukcje

  LOOP UNTIL Warunek

- W pierwszej postaci warunek sprawdzany na początku
  a w drugiej postaci na końcu.
- Gdy warunek spełniony to następuje wyjście – przerwanie wykonywania instrukcji w pętli

Schemat instrukcji POWTARZAJ – DO… UNTIL

Pseudokod **powtarzaj I aż do W;**



Np.

```
DO
 k = k + 1
 PRINT k
LOOP UNTIL k = 20
```

Do
Instrukcje
LOOP Until Warunek

Lub

DO UNTIL Warunek
Instrukcje
LOOP

przynajmniej raz się wykona - warunek na koncu

Pętle z licznikiem DLA – FOR … NEXT

- Pętle z licznikiem są wykonywane zadaną z góry ilość razy zgodnie z wielkościami zmiennych sterujących.
  Realizowane są w Basicu przy pomocy struktury FOR...NEXT.
- **Składnia instrukcji FOR :**

FOR ZmiennaSterująca = WartośćPoczątk **TO** WartośćKońc **[STEP** krok**]**

    Instrukcje wnętrza pętli

NEXT [ZmiennaSterująca]

ZmiennaSterująca - licznik pętli, zmienia się po każdym jej przebiegu o wielkość kroku

WartośćPoczątk - wartość początkowa – startowa licznika

WartośćKońc – wartość końcowa licznika, krok – wielkość kroku

STEP krok - może być pominięte (wtedy domyślna wielkość kroku wynosi 1), w przeciwnym wypadku krok jest zmienną określającą o ile ma się zwiększać licznik pętli po każdym jej przebiegu.

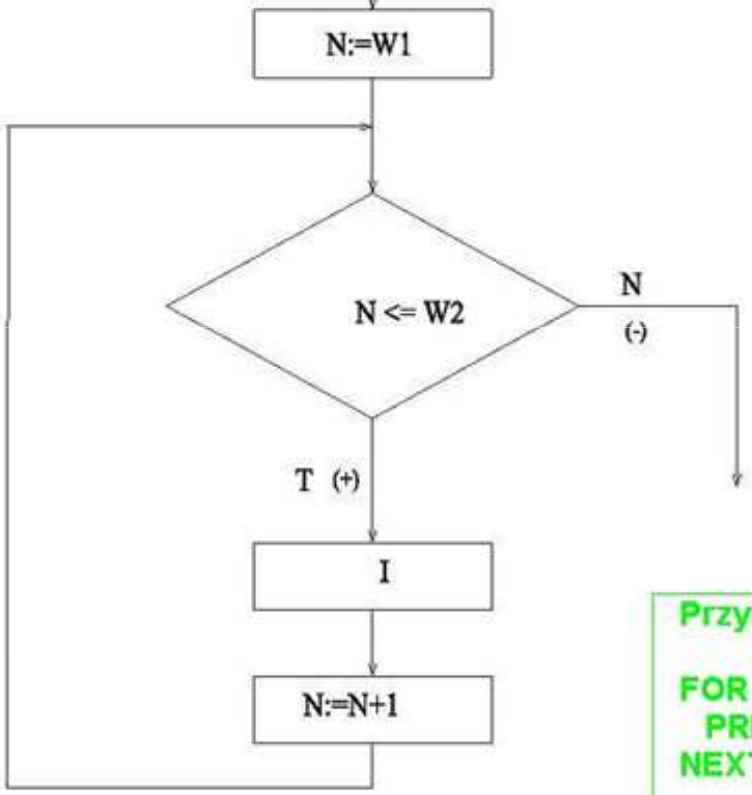
Schemat instrukcji DLA - FOR

Iteracje DLA

Basic

Pseudokod

dla N:=W1 do W2 wykonuj I;




```
Przyklad;

FOR i=1 TO 20 STEP 2
  PRINT i
NEXT i
```

```
for N =W1 to W2
  I
next N
```

lub

```
for N = W1 to W2 [step przyrost]
                I
next N
```

Przykład programu z FOR - znajdowanie największej i najmniejszej liczby

```
REM abcq11.bas
REM znajdowanie najwiekszej i najmniejszej liczby
 CLS :  DIM liczba(50):  INPUT "Ile liczb "; n
FOR i = 1 TO n
 PRINT "liczba nr "; i;  :  INPUT liczba(i) :  NEXT i
REM znajdowanie liczby największej
max = liczba(1) '1
FOR i = 1 TO n
 IF liczba(i) > max THEN max = liczba(i) '2
NEXT i
REM znajdowanie liczby najmniejszej '3
min = liczba(1)
FOR i = 1 TO n
IF liczba(i) < min THEN min = liczba(i) '4
NEXT i
REM wydruk liczby max i minim.
PRINT  "Liczba największa = "; max, "Liczba najmniejsza="; min
END
```

Uwagi dotyczące pętli

- Przy używaniu pętli warunkowych należy pamiętać o tym, by zmienna występująca w warunku mogła się zmieniać (inaczej możemy uzyskać pętlę nieskończoną).
- Z pętli z licznikiem jak i z pętli warunkowych możemy wyjść wcześniej niż to zakładają warunki funkcjonowania pętli.
  Do tego służą polecenia **EXIT FOR** oraz **EXIT DO**. Zazwyczaj instrukcje te będą szły w parze z instrukcjami **IF...THEN**.
- W przypadku gdy stworzymy pętle nieskończoną, a program jest uruchomiony, należy wcisnąć CTRL+BREAK. Gdy program działa jako samodzielna aplikacja pod środowiskiem Windows, można przerwać jego wykonywanie przy pomocy Managera zadań.
- Pętle można zagnieżdżać jedną wewnątrz drugiej, pamiętając jednak o tym że wewnętrzna pętla musi się zawierać całkowicie wewnątrz zewnętrznej (tj. pętle nie mogą się krzyżować).

IF warunek THEN EXIT - opuszczenie pętli

- **' Przyklad ABCQ8A.BAS**

```
REM ABCQ8a.BAS
REM Petla DO..LOOP
CLS
k = 0
INPUT "Ile przebiegow petli "; n
DO
  k = k + 1
  IF k = n THEN EXIT DO
LOOP
PRINT "Wykonano "; k, "przebiegow petli LOOP"
```

Funkcje operujące na znakach - wybrane

- **CHR$ i ASC**

Są to funkcje pozwalające na zamianę kodów ASII
na odpowiadające im znaki – CHR$
oraz na znajdowanie kodów ASCII podanych znaków - ASC.

- **CHR**$(Numer_kodu)
  - Zwraca literę (łańcuch jednoznakowy) odpowiadającą danemu numerowi kodu ASCII.
  - Argumentem funkcji może być dowolny kod ASCII, liczba od 0 do 255. Wynikiem jest znak odpowiadający kodowi (liczbie).
  - Np**. print chr$(77)** da w wyniku **M**
- **ASC**(wyrazenie_lancuchowe)
  - Zwraca nr kodu ASCII pierwszej litery w wyrażeniu łańcuchowym

Przykłady programów z ASC i CHR$

Przykład 1:

```
start:    ' lub [start]
iNPUT "znak"; zn$  ‘ Wprowadzamy jakiś znak z klawiatury
PRINT ASC(zn4)                ‘ wydruk kodu znaku (liczby)
GOTO start
```

Przykład 2

```
' Wydruk wszystkich znaków ASCII
CLS
PRINT "Kody ASCII i znaki "
FOR i = 0 TO 255
 PRINT "kod: ", i, "  znak ", CHR$(i)   ‘ Wyświetla znak (np. literę) dla kolejnych
     cyfr od 0 do 255
NEXT i
```

Inne funkcje operujące na znakach:

- **LEFT**$(wyrazenie_lancuchowe, n)
  - Zwraca n pierwszych znaków wyrażenia łańcuchowego
  RIFHT$(wyrazenie_lancuchowe, n) - Zwraca n ostatnich znaków wyrażenia łańcuchowego

  np. **a$="dzien dobry" : print right$(as$, 5)**
  ‘ Na ekranie będzie dobry
- **MID$(**wyrazenie_lancuchowe, start, [dlugosc])
  - Funkcja zwraca łańcuch wycięty z wnętrza łańcucha

  np. **print mid$("Ala ma kota", 1, 3)** - Wynik: Ala
- **LEN**(wyrazenie_lancuchowe)
  ‘Zwraca ilość znaków w wyrażeniu łańcuchowym,

Podprogram (procedura) a **funkcja**

- **Podprogram SUB** (inaczej **procedura**) - może wykonać wiele operacji i nie zwraca konkretnej jednej wartości,
- **Funkcja FUNCTION** zwraca jedną konkretną wartość, choć może też wykonać wiele operacji

Składnia **podprogramu** (procedury)

SUB nazwa_ogolna (lista_parametrow_formalnych)

...

END SUB

- Np.

```
SUB drukuj (x)
   PRINT "Jestem w podprogramie DRUKUJ ";
   PRINT "x="; x, "a="; a, "b="; b
END SUB
```

Deklaracja procedury na początku programu

DECLARE SUB nazwa_ogolna (lista_parametrów_formalnych)

Np.

DECLARE SUB drukuj (x!)

' ! oznacza single precision – real,

‘ x! – parametr formalny

- **Wywołanie podprogramu**

CALL nazwa_podprogramu (parametry_aktualne)

Np. **CALL drukuj(b)**

‘ Wywołanie podprogramu z parametrem aktualnym b

Przekazywanie wartości zmiennych między podprogramami

- Podprogram musi być wywołany z odpowiednimi argumentami.

  np. instrukcja wywołania **CALL podpr1(a, b)**

  powoduje przejście do wykonywania podprogramu **SUB podpr1(x,y)**

  Wartości zmiennych a, b (argumenty) w chwili przejścia do instrukcji wywołania, są przekazane do zmiennych x i y w podprogramie.

  Zmiana wartości x, y, w wyniku obliczeń wykonywanych w podprogramie, nie powoduje jednoczesnych zmian wartości zmiennych a i b w programie wywołującym.

  Dopiero powrót do programu wywołującego powoduje, ze zmienne a, b przyjmują wartości, jakie w momencie opuszczania podprogramu miały w nim zmienne x i y.

  Zmienne o tych samych nazwach występujące w programie i podprogramach, jeśli nie są umieszczone jako argumenty w instrukcji wywołania, są zupełnie od siebie niezależne,

  chyba, że umieścimy w podprogramie specjalne deklaracje **SHARED**.

- Reguły te można sprawdzić z przebiegu wykonania poniższego programu

Przykład z podprogramem drukuj

```
REM abcq13.bas  - podprogram drukuj – 2 wywołania
DECLARE SUB drukuj (x!)   ' ! oznacza single precision - real
CLS
a = 7
b = 15
CALL drukuj(a)
PRINT "Pierwszy powrót do Main"
CALL drukuj(b)
PRINT "Drugi powrót do Main"
PRINT "w programie MAIN a="; a, "b="; b, "x="; x
END

SUB drukuj (x)
PRINT "Jestem w podprogramie DRUKUJ ";
PRINT "x="; x, "a="; a, "b="; b
END SUB
```

Wyniki uruchomienia programu

- Jestem w podprogramie DRUKUJ x=7 a=0 b=0
- Pierwszy powrót do Main
- Jestem w podprogramie DRUKUJ x=15 a=0 b=0
- Drugi powrót do Main
- w programie MAIN a=7 b=15 x=0

Deklaracja COMMON SHARED, SHARED

Składnia;

COMMON [**SHARED**][**/blok/**] **zmienna** [({**rozmiar**})] [**AS typ**] [, ...]

Opis: Definiuje zmienne globalne wspólne dla różnych modułów tego samego programu lub różnych programów powiązanych ze sobą.

Parametry:

SHARED - określa, że zmienne tego modułu będą dostępne w całym programie

blok - nazwa identyfikująca grupę zmiennych; maks. 40 znaków

zmienna - nazwy zmiennych

rozmiar - dowolna stała podająca liczbę wymiarów dla zmiennych tablicowych

AS typ - deklaruje zmienne jako typu: **INTERGER, LONG, SINGLE, DOUBLE, STRING** lub wg własnej definicji.

Przykład programu ze zmiennymi SHARED

```
REM ABCQ14.BAS
REM 2 podprogramy: druk1 i druk2
DECLARE SUB druk1 ()
DECLARE SUB druk2 ()
COMMON SHARED ws
CLS
ws = 23
PRINT "W programie glownym ws="; ws
 CALL druk1    ' wywołanie podprogramu druk1 bez podawania argumentów
 CALL druk2    ' wywołanie podprogramu druk2 bez podawania argumentów
END
SUB druk1
PRINT "w SUBroutine druk1 ws="; ws
END SUB
SUB druk2
PRINT "Tu takze (w subroutine druk2)  ws="; ws
END SUB
```

Funkcja

- **Funkcja** jest oddzielnym modułem programu, w którym mogą być wykonywane różne operacje, a ich wynik przekazywany do programu głównego poprzez nazwę funkcji.
- Deklaracja funkcji:
  DECLARE FUNCTION nazwafunkcji (parametry)
- Wywołanie funkcji:
  Nazwafunkcji (parametry_aktualne)

Przykład programu z **funkcją** sumprz(a, b, c)

- Przykład: obliczenie sumy długości przekątnych prostopadłościanu o danych krawędziach, z wykorzystaniem funkcji.

```
REM Zastosowanie funkcji – FUNCTION nazwa (parametry)
DECLARE FUNCTION sumprz (a, b, c)
CLS
INPUT "Podaj 3 liczby "; a, b, c
PRINT "Suma dlug. przekątnych  prostopadloscianu  a,b,c= "; sumprz(a, b, c)
END  ‘ Koniec programu głównego

FUNCTION sumprz (a, b, c)
 p1 = SQR(a ^ 2 + b ^ 2)
 p2 = SQR(b ^ 2 + c ^ 2)
 p3 = SQR(a ^ 2 + c ^ 2)
 sumprz = 2 * (p1 + p2 + p3)
END FUNCTION
```