Struktura programu

```
PROGRAM nazwa;
   opis nazw zmiennych i ich typu
BEGIN
   ciąg instrukcji;
   wyprowadzenie rezultatu
END.
```

Analogie do przepisu kulinarnego:

- ✓ nazwa potrawy $\longrightarrow$ nazwa programu,
- ✓ składniki $\longrightarrow$ deklaracje zmiennych,
- ✓ przepis $\longrightarrow$ ciąg instrukcji.

```
PROGRAM Suma2;
VAR a, b, c: Integer;
BEGIN
   Read(a, b);
   c := a + b;
   WriteLn(a, b, c)
END.
```

Efekt wykonania programu:

```
13   11   ↵
13   11   24
_
```

Możliwe wariacje:

- Poprzez `Read` uzyskuje się taki sam efekt jak `a := 13`, `b := 11`, ale nie trzeba zmieniać programu.
- Można napisać

  ```
  Read(a);
  Read(b);
  ```

  ale `Read(a, b)` jest bardziej zwarte.

- `WriteLn(a, b, c)` $\equiv$

  ```
  Write(a);
  Write(b);
  WriteLn(c)
  ```

Komunikacja z użytkownikiem

```
PROGRAM Suma2;
VAR a, b, c: Integer;
BEGIN
   Write('Podaj dwie liczby: ');
   Read(a, b);
   c := a + b;
   WriteLn('1-szy składnik = ', a);
   WriteLn('2-gi składnik = ', b);
   WriteLn('Suma składników = ', c)
END.
```

Efekt wykonania programu:

```
Podaj dwie liczby: 13   11   ↵
1-szy składnik = 13
2-gi składnik = 11
Suma składników = 24
_
```

Komentarze

Wszystko, co stoi między symbolami `{` i `}` (lub `(*` i `*)`) jest ignorowane przy tłumaczeniu programu źródłowego na kod maszynowy.

```
PROGRAM Komentarze;
(* Mały program z komentarzami *)
VAR a: Integer;
BEGIN { Początek }
   a := 1;
   WriteLn('To na ekranie: a = ', a);
   (* Ten komentarz jest zbyteczny *)
END.
```

Efekt wykonania programu:

```
To na ekranie: a = 1
_
```

Styl („charakter pisma”)

```
PROGRAM
        Suma2
;
VAR   a,
      b,   c   :
      Integer;
BEGIN READ(a, b);
c:=
        a+
   b
; WriteLn(a, b,c) END.
```

Instrukcja przypisania

```
PROGRAM Przypisanie;
VAR a: Integer;
BEGIN
   a := 4;     WriteLn(a);
   a := a + 1; WriteLn(a);
   a := 8;     WriteLn(a)
END.
```

Operacje na liczbach całkowitych

```
PROGRAM calkowite;
VAR a, b, c: Integer;
BEGIN
   a := 17;
   b := 3;
   c := a * b;
   Writeln('17 * 3 = ', c);
   c := a DIV b;
   WriteLn('17 DIV 3 = ', c);
   c := a MOD b;
   WriteLn('17 MOD 3 = ', c);
   c := a + b;
   Writeln('17 + 3 = ', c);
   c := a - b;
   Writeln('17 - 3 = ', c);
END.
```

Efekt wykonania programu:

```
17 * 3 = 51
17 DIV 3 = 5
17 MOD 3 = 2
17 + 3 = 20
17 - 3 = 14
_
```

Etymologia:

- DIV od *divide*, czyli „dzielić”,
- MOD od *modulo*, czyli określenie reszty.

Funkcje standardowe

```
PROGRAM funkcje_calkowite;
(* Demonstracja
      funkcji standardowych *)
VAR a, b, c: Integer;
BEGIN
   a := -2;
   b := Abs(a);
   WriteLn('Abs(-2) = ', b);
   c := Sqr(b);
   WriteLn('Sqr(b) = ', c);
```

```
    c := Sqr(b + b);
    WriteLn('Sqr(b + b) = ', c)
END.
```

Efekt wykonania programu:

```
Abs(-2) = 2
Sqr(b) = 4
Sqr(b + b) = 16
_
```

Etymologia: absolute value = wartość bezwzględna, square = kwadrat

Co to jest *Real*?

```
PROGRAM Suma3;
VAR a, b, c: Real;
BEGIN
    a := 3.5;
    b := 7.6;
    c := a + b;
    Writeln('Suma = ', c)
END.
```

Efekt wykonania programu:

```
Suma = 1.1100000000E+01
_
```

Zapis liczb

Reguła: Jeśli w liczbie pojawia się kropka dziesiętna, to w każdym przypadku przed, jak również po niej, powinna stać cyfra!

.7	0.	$\longleftarrow$ zabronione!
0.7	0.0	$\longleftarrow$ poprawne!

Postać z wykładnikiem:

$$\text{liczba} = \text{mantysa} \times 10^{\text{potęga}}$$

Przykłady

$$\texttt{2.7E3} = 2,7 \times 10^{3} = 2,7 \times 1000 = 2700$$

$$\texttt{-1.51E-5} = -1,51 \times 10^{-5} = -0,0000151$$

$$\texttt{2.753E12} = 2753000000000$$

$$\texttt{2.753E-12} = 0.000000000002753$$

Operacje na liczbach rzeczywistych

```
PROGRAM Rzeczywiste;
VAR a, b, c: Real;
BEGIN
   a := 17.3;
   b := 3.4;
   c := a * b;
   Writeln('a * b = ', c);
   c := a / b;
   WriteLn('a / b = ', c);
   c := a + b;
   Writeln('a + b = ', c);
   c := a - b;
   Writeln('a - b = ', c);
END.
```

```
a * b = 5.8820000000E+01
a / b = 5.0882352941E+00
a + b = 2.0700000000E+01
a - b = 1.3900000000E+01
_
```

Integer i *Real* razem

```
PROGRAM Rozne_typy;
VAR n, k: Integer;
    a, b: Real;
BEGIN
   a := 3.6; n := 4;
   b := n; WriteLn('b = ', b);
   n := Trunc(a);
   WriteLn('Trunc(3.6) = ', n);
   k := Round(a);
   WriteLn('Round(3.6) = ', k)
END.
```

```
b = 4.0000000000E+00
Trunc(3.6) = 3
Round(3.6) = 4
_
```

Czy dozwolone są poniższe instrukcje przypisania?

```
b := n + 4.6;
b := 3 * 7.2 + n;
n := 2.5 * 4;
```

Wykorzystanie nawiasów i priorytety

Jak interpretować poniższą instrukcję przypisania?

```
a := 4 - 3 * 8;
```

Mamy dwie możliwości:

① $4 - (3 \times 8)$

② $(4 - 3) \times 8$

Mamy następujące reguły priorytetów :

- Mnożenie i dzielenie wykonuje się wcześniej niż dodawanie i odejmowanie (lub: mnożenie i dzielenie mają wyższy priorytet niż dodawanie i odejmowanie).
- Operacje o identycznym priorytecie wykonują się od lewej na prawo. Mnożenie i dzielenie mają jednakowy priorytet (podobnie dodawanie i odejmowanie).
- Operacje na zmiennych zawartych w nawiasach mają pierwszeństwo.

Przykład wykorzystania nawiasów:

```
PROGRAM Transformacja;
VAR far, cel: Real;
BEGIN
   WriteLn('Podaj temperature ',
           'Fahrenheita');
   Read(far);
   cel := ((far - 32) / 9) * 5;
   WriteLn('Temperatura Celsjusza:',
                                cel)
END.
```

Funkcje standardowe na typie *Real*

```
PROGRAM Wszystkie_fukcje;
VAR a, b: Real;
BEGIN a := 2.0;
   b := Sqr(a);
   WriteLn('Sqr(2) = ', b);
   b := Sqrt(a);
   WriteLn('Sqrt(2) = ', b);
   b := Sin(a);
   WriteLn('Sin(2) = ', b);
```

```
    b := Cos(a);
    WriteLn('Cos(2) = ', b);
    b := ArcTan(a);
    WriteLn('ArcTan(2) = ', b);
    b := Ln(a);
    WriteLn('Ln(2) = ', b);
    b := Exp(a);
    WriteLn('Exp(2) = ', b)
END.
```

```
Sqr(2) = 4.0000000000E+00
Sqrt(2) = 1.4142135624E+00
Sin(2) = 9.0929742682E-01
Cos(2) = -4.1614683655E-01
ArcTan(2) = 1.1071487178E+00
Ln(2) = 6.9314718056E-01
Exp(2) = 7.3890560989E+00
```

Funkcje te można składać:

```
a := 4.0;
b := Sqrt(25.6 + Sqr(a) + a * 6.5);
```

Stałe

```
PROGRAM Okrag1;
VAR l, r, pi: Real;
BEGIN
   pi := 3.14159;
   Read(r);
   l := 2 * pi * r;
   WriteLn('l=', l)
END.
```

Jak zabezpieczyć się przed przypadkową zmianą wartości `pi`? Lepiej napisać

```
PROGRAM Okrag2;
CONST pi = 3.14159;
VAR l, r: Real;
BEGIN
   Read(r);
   l := 2 * pi * r;
   WriteLn('l=', l)
END.
```

Uwaga: Stała `Pi` jest predefiniowana!

Inne przykłady:

```
CONST E = 2.71828;
      tuzin = 12;
```

Pytanie: Jak określa się typ stałej?

```
PROGRAM Predkosc_upadku;
CONST g = 9.81;
VAR v, h: Real;
BEGIN
   Read(h);
   v := Sqrt(2 * g * h);
   WriteLn(v)
END.
```

Instrukcja `Write`

```
PROGRAM Formatowanie;
VAR a1, a2, b1, b2: Real;
BEGIN
   a1 := 1.111; a2 := a1 * a1;
   b1 := 1.222; b2 := b1 * b1;
   WriteLn('Wielkosc': 10,
                    'Kwadrat': 10);
```

```
   WriteLn(a1: 10: 2, a2: 10: 2);
   WriteLn(b1: 10: 2, b2: 10: 2)
END.
```

```
  Wielkosc   Kwadrat
       1.11       1.23
       1.22       1.49
_
```

Typ logiczny

```
PROGRAM Porownywanie;
VAR x: Integer;
    b: Boolean;
BEGIN
   x := 4;
   b := x > 3;
   WriteLn(b);
   b := x < 3;
   WriteLn(b)
END.
```

```
TRUE
FALSE
_
```

Etymologia: George Boole (1815-1864)

W wielu programach występuje sytuacja, kiedy na podstawie określonego warunku będzie lub nie będzie wykonany szereg instrukcji. Taki warunek możemy rozumieć jako stwierdzenie *prawdziwe* lub *fałszywe* (warunek spełniony lub niespełniony).

symbol	relacja	przykład
<	$<$	`x < 3`
<=	$\leq$	`x <= 3`
>	$>$	`x > 3`
>=	$\geq$	`x >= 3`
=	$=$	`x = 3`
<>	$\neq$	`x <> 3`

Warunki mogą zawierać wyrażenia arytmetyczne:

```
x + 6.5 < y * 5
```

Można też rozważać bardziej złożone warunki, np.

```
x < 7  i  x > 3
```

```
x > 100  lub  x < 10
```

dla których odpowiednio mamy zapis

```
(x < 7) AND (x > 3)
```

```
(x > 100) OR (x < 10)
```

a	b	a AND b	a OR b
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Oprócz tego istnieje operator NOT, który neguje wartość wyrażenia:

```
NOT (a < b)   ≡   a >= b
```

Pytanie: Jak zinterpretować poniższy warunek?

```
(3<x) AND (x<10) OR (13<x) AND (x<20)
```

Odpowiedź: Zob. reguły priorytetów.

❶ Najwyższy: NOT,

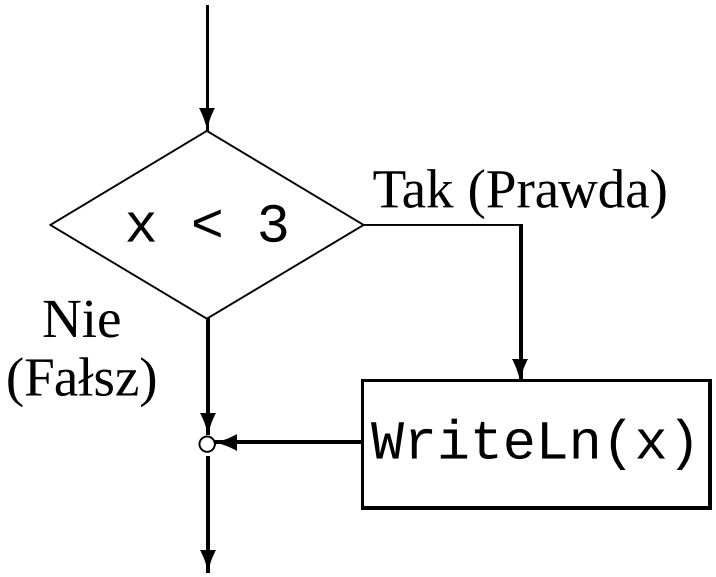
❷ Niższy: *, /, DIV, MOD, AND,

❸ Najniższy: : +, -, OR, operacje porównywania.

Instrukcja warunkowa IF

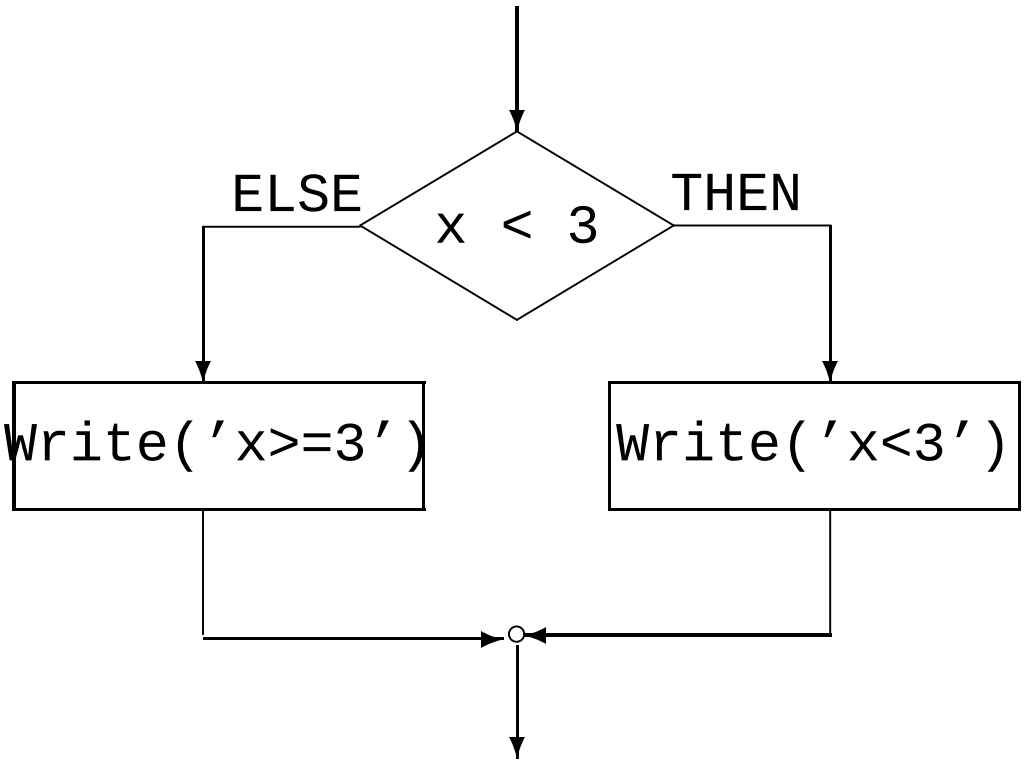
Instrukcje warunkowe służą do wpływania na kolejność wykonania instrukcji programu.

```
IF x < 3 THEN WriteLn(x);
```




```
WriteLn('Podaj swoj wiek');
Read(wiek);
IF wiek >= 18 THEN
   WriteLn('Jestes pelnoletni');
```

```
IF x < 3 THEN Write('x<3')
         ELSE Write('x>=3');
```




```
WriteLn('Podaj swoj wiek');
Read(wiek);
IF wiek >= 18 THEN
   WriteLn('Jestes pelnoletni')
ELSE
   WriteLn('Jestes maloletni');
```

Uwaga! Przed ELSE nigdy nie piszemy średnika!

```
PROGRAM Instrukcja_grupujaca;
VAR x: Integer;
BEGIN
   Read(x);
   IF x < 2 THEN WriteLn('x<2');
   WriteLn('Dzien dobry!');
   IF x < 2 THEN WriteLn('x<2')
            ELSE WriteLn('x>=2');
   WriteLn('Dalej');
   IF x < 2 THEN
      BEGIN
         WriteLn('x<2');
         WriteLn('mala liczba')
      END;
   WriteLn('Koniec programu')
END.
```

Uwaga: Po `THEN` (lub `ELSE`) można znowu umieścić operator `IF`.

Pytanie: Jaki będzie rezultat poniższych instrukcji?

```
WriteLn(1/3 = Sqr(Sqrt(1/3)));
WriteLn(1.0 + 1.0e-20 = 1.0);
```

```
WriteLn(Abs(1/3 - Sqr(Sqrt(1/3)))
                              < 1.0e-5);
```

Type znakowy

`Char` – uporządkowany zbiór wszystkich znaków graficznych oraz sterujących reprezentowanych na danym komputerze (zależy od typu komputera).

Znaki ASCII:

- Litery: `'A'`, `'B'`, `'C'`, `'D'`, ..., `'Z'`, `'a'`, `'b'`, `'c'`, `'d'`, ..., `'z'`;
- Cyfry: `'0'`, `'1'`, `'2'`, `'3'`, ..., `'9'`;
- Znaki specjalne: `'!'`, `'@'`, `'#'`, `'%'`, ...;
- Spacja: `' '`;
- Znaki sterujące, np. `^M`, czyli CR (*ang.* carriage return) – powrót kursora na początek wiersza, `^J` czyli LF (*ang.* line feed) – przejście kursora do następnego wiersza.

Pytanie: Jak uzyskać apostrof? ➜ `''''`

Znaki są wewnętrznie reprezentowane przez ciąg zer i jedynek. Jeżeli te ciągi zinterpretuje się jako liczby całkowite, zdefiniuje się w ten sposób *uporządkowanie* zbioru znaków, np.:

LR$\mapsto$10	CR$\mapsto$13	`' '`$\mapsto$32
`'0'`$\mapsto$48	`'1'`$\mapsto$49	`'2'`$\mapsto$50
`'A'`$\mapsto$65	`'B'`$\mapsto$66	`'C'`$\mapsto$67
`'a'`$\mapsto$97	`'b'`$\mapsto$98	`'c'`$\mapsto$99

Dostępne funkcje:

- `Ord('L')` $\rightarrow$ 76
- `Chr(22)` $\rightarrow$ `'!'`
- `Pred('*')` $\rightarrow$ `')'`
- `Succ('b')` $\rightarrow$ `'c'`

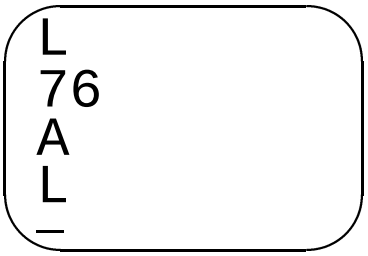
Własności:

```
Ord(Chr(i)) = i
Chr(Ord(c)) = c
   Pred(c) = Chr(Ord(c) - 1)
   Succ(c) = Chr(Ord(c) + 1)
```

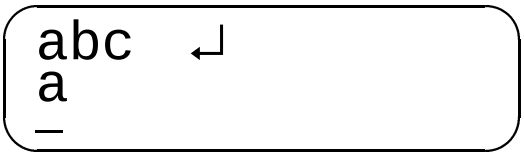
```
PROGRAM Litery;
VAR c1, c2, c3: Char;
BEGIN
   c1 := 'L'; WriteLn(c1);
   c2 := Pred(c1);
   WriteLn('Pred = ', c2);
   c3 := Succ(c1);
   WriteLn('Succ = ', c3)
END.
```

```
L
Pred = K
Succ = M
_
```

```
PROGRAM Litery_i_liczby;
VAR n: Integer; c: Char;
BEGIN
   c := 'L'; WriteLn(c);
   n := Ord(c); WriteLn(n);
   c := 'A'; WriteLn(c);
   c := Chr(n); WriteLn(c)
END.
```




```
PROGRAM znak;
VAR c: Char;
BEGIN
   Read(c); Write(c)
END.
```




```
PROGRAM na_duze;
VAR c: Char;
BEGIN
   Read(c);
   IF ('a' <= c) AND (c <= 'z') THEN
      c := Chr(Ord(c)
               - Ord('a') + Ord('A'));
   Write(c)
END.
```

Uzupełnienia:

- Bufor tekstowy a wciśnięcie klawisza `<Enter>` (`<Enter>` $\equiv$ CR + LF!),
- `STRING` – `'Pascal'`, `'McDonald''s'`.

Instrukcja WHILE . . . DO

Zadanie. Obliczyć sumę ciągu liczb, w którym pierwszy składnik jest zadany, a każdy następny jest o jeden mniejszy od poprzedniego. Ciąg ma się kończyć na ostatniej liczbie większej od 5.5.

```
VAR x, sum: Real;
BEGIN
   sum := 0.0; x := 10.0;
   WHILE x > 5.5 DO
      BEGIN
         sum := sum + x;
         x := x - 1
      END;
   WriteLn('sum =', sum: 6: 2);
   WriteLn('  x =',   x: 6: 2)
END.
```

```
sum = 40.00
  x =  5.00
_
```

- Wiele instrukcji wewnątrz pętli ➡ zastosować instrukcję grupującą `BEGIN ... END`.
- Wartość wyrażenia logicznego musi być modyfikowana wewnątrz pętli.
- Liczba wykonań nie jest z góry wiadoma.
- Kiedy instrukcje wewnątrz pętli nie wykonanją się?

Instrukcja REPEAT ... UNTIL

```
VAR x, sum: Real;
BEGIN
   sum := 0.0; x := 10.0;
   REPEAT
      sum := sum + x;
      x := x - 1
   UNTIL x < 5.5;
   WriteLn('sum =', sum: 6: 2);
   WriteLn('  x =',   x: 6: 2)
END.
```

Instrukcja FOR

```
FOR k := 3 TO 23 DO
    WriteLn(k * k);
FOR k := 99 DOWNTO 78 DO
    WriteLn(k * k);
```

Uwaga: Zmienna sterująca `k` nie może być typu `Real`, a krok zawsze jest równy jedności.

```
PROGRAM Szereg_liczb;
VAR sum: Real;
    x: Integer;
BEGIN
   sum := 0.0;
   FOR x := 10 DOWNTO 6 DO
      sum := sum + x;
   WriteLn('sum =', sum: 6: 2);
   WriteLn('  x =',   x: 6: 2)
END.
```

Zadanie. Wyznaczyć wartość sumy

$$\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\frac{1}{5^2}$$

```
VAR sum, a, a2: Real;
    k: Integer;
BEGIN
   sum := 0.0;
   FOR k := 1 TO 5 DO
      BEGIN
         a := 1 / k;
         a2 := a * a;
         sum := sum + a2;
         WriteLn('a2 =', a2: 10: 7)
      END;
   WriteLn('sum =', sum: 10: 7)
END.
```

(Ważne) uwagi:

- Już na początku wiadomo ile razy pętla się wykona.
- Po zakończeniu pętli zmienna sterująca nie jest określona!

- Wartość początkową i końcową zmiennej sterującej wyznacza się tylko raz – zanim rozpocznie się wykonywanie pętli.
- Wewnątrz pętli nie można zmieniać wartości zmiennej sterującej!
- Zmienna sterująca niekoniecznie musi być typu `Integer`:

```
FOR litera := 'P' DOWNTO 'J' DO
   Write(litera);
```

- Jak wprowadzić krok inny od jedności?

```
PROGRAM SumaParzystych;
VAR sum, nrParz, lParz: Integer;
BEGIN
   sum := 0;
   FOR nrParz := 1 TO 500 DO
      BEGIN
         lParz := nrParz * 2;
         sum := sum + lParz
      END;
   WriteLn(sum)
END.
```

Typ wyliczeniowy

Problem: Jak reprezentować pojęcia *miesiąc*, *dzień tygodnia*, *kolor tęczy*, *owoce*, *pora roku*, *samochód*? Kodowanie z zastosowaniem zmiennych całkowitych prowadzi do utraty czytelności:

```
IF b = 9 THEN ...
```

O wiele wygodniejsza byłaby konstrukcja

```
IF b = wrzesien THEN ...
```

Przykłady typów wyliczeniowych:

```
TYPE
  miesiac = (styczen, luty, marzec,
             kwiecien, maj, czerwiec,
           lipiec, sierpien, wrzesien,
                pazdziernik, listopad,
                             grudzien);
  figura = (pionek, skoczek,
      goniec, wieza, hetman, krol);
```

```
  dzien_tygodnia = (pn, wt, sr, cz,
                    pt, so, ni);
```

Wartości typu wyliczeniowego są uporządkowane:

$$\underbrace{\texttt{pn}}_{0} < \underbrace{\texttt{wt}}_{1} < \underbrace{\texttt{sr}}_{2} < \cdots < \underbrace{\texttt{ni}}_{6}$$

Dostępne funkcje:

- `Ord(pn)` ➜ `0`
- `Succ(pn)` ➜ `wt`
- `Pred(ni)` ➜ `so`

Pytanie: Dlaczego niepoprawna jest poniższa deklaracja?

```
TYPE
  dzien_tygodnia = (pn, wt, sr, cz,
                    pt, so, ni);
  weekend = (so, ni);
```

Na podobnej zasadzie, niedozwolone są deklaracje

```
TYPE  maleLiczbyCalkowite
          = (0, 1, 2, 3, 4, 5);
```

```
   samogloski
        = (’a’, ’e’, ’i’, ’o’, ’u’);
   wartosciLogiczne = (True, False);
```

Deklaracje zmiennych:

```
VAR
   dzien: dzien_tygodnia;
   a, b, c: (krzeslo, kanapa, stol,
                     szafa, taboret);
```

Możliwe są wówczas przypisania:

```
   dzien := niedziela;
   a := krzeslo;
```

Konstrukcje zaawansowane:

```
   IF dzien > pn THEN
      dzien := Pred(dzien);
   FOR dzien := pn TO so DO
      WriteLn(Ord(dzien));
```

Problem: Standardowe operacje WE/WY nie mogą być użyte do wprowadzania i wyprowadzania wartości typów wyliczeniowych. Np. błędne jest

```
   dzien := wtorek; Write(dzien);
```

Instrukcja wyboru

Selektor typu całkowitego:

```
Randomize;
k := Random(9) + 1;
CASE k OF
   1, 2: Write('Prawdopodob. 0.2');
   3..5: Write('Prawdopodob. 0.3')
ELSE
   Write('Prawdopodob. 0.5')
END;
```

Selektor typu znakowego:

```
CASE znak OF
  'a'..'z',
  'A'..'Z': Write('Litera');
  '0'..'9': Write('Cyfra');
  ' ': Write('Odstep')
ELSE
  Write('Znak specj. lub sterujacy')
END;
```

Selektor typu wyliczeniowego:

```
CASE dzien OF
   pn..pt: WriteLn('Dzien roboczy');
   so, ni: WriteLn('Dzien wolny')
END;
```

Wypisywanie wartości zmiennych typu wyliczeniowego:

```
CASE dzien OF
   pn: Write('poniedzialek');
   wt: Write('wtorek');
   sr: Write('sroda');
   cz: Write('czwartek');
   pt: Write('piatek');
   so: Write('sobota');
   ni: Write('niedziela')
END;
```

Uwaga! Jako selektora nie można używać wartości typu `Real`!

```
PROGRAM Brygada;
TYPE
  nazwisko = (Jankowski, Kowalski,
        Majewski, Nowak, Sikorski,
                          Strzelecki);
  tydzien = (pn, wt, sr, cz, pt);
VAR
  najwydajniejszy,
         robotnik: nazwisko;
  dzien: tydzien;
  k, m, max: Integer;
BEGIN
  max := -1;
  FOR robotnik := Jankowski
              TO Strzelecki DO
    BEGIN
      m := 0;
      FOR dzien := pn TO pt DO
        BEGIN
          Read(k); m := m + k
        END;
```

```
      IF m > max THEN
        BEGIN
          najwydajniejszy
                    := robotnik;
          max := m
        END
    END;
  Write('Najwydajniejszy byl ');
  CASE najwydajniejszy OF
    Jankowski:  Write('Jankowski');
    Kowalski:   Write('Kowalski');
    Majewski:   Write('Majewski');
    Nowak:      Write('Nowak');
    Sikorski:   Write('Sikorski');
    Strzelecki: Write('Strzelecki)
  END;
  WriteLn(', ktory wykonal ', max,
                          ' detali')
END.
```

Pytanie: Kiedy stosować `IF`, a kiedy `CASE`?

```
IF a DIV b < 4 THEN ...;
```

Wykonanie powyższej instrukcji może spowodować błąd. Alternatywą mogłoby być

```
IF (b <> 0) AND (a DIV b < 4) THEN
   ...;
```

Wykonanie zależy jednak od implementacji.
Pozostaje więc

```
IF b <> 0 THEN
   IF a DIV b < 4 THEN
      ...;
```

Typ okrojony

```
TYPE
   maleCalkowite = 1..20;
   maleLitery = 'a'..'z';
   dzienRoboczy = pn..pt;
```

Powyższą informację o zakresach przekazuje się translatorowi, aby kontrolował poprawność przypisywania wartości zmiennym.

```
TYPE
   indeks = -10..20;
VAR
   i, j: Integer; l: indeks;
BEGIN
   ...
   i := 200;
   j := 5;
   l := i * j - 2;
```

Wykonanie ostatniej instrukcji spowoduje wyświetlenie komunikatu o błędzie (*Range Check Error*).

Typ tablicowy

Zadanie. Pewna 100-osobowa grupa studencka zdawała egzamin z informatyki. Napisać program pytający użytkownika o numery ewidencyjne i liczby zdobytych punktów kolejnych studentów, a następnie komunikujący którzy studenci uzyskali wynik powyżej średniej.

Nasuwające się rozwiązania:

❶ Wczytać dane, obliczyć średnią, po czym ponownie wczytać dane.

❷
```
TYPE punkty = 0..100;
VAR ocena1, ocena2, ...,
          ..., ocena100: punkty;
      numSt1, numSt2, ...,
          ..., numSt100: Integer;
```

Co jednak zrobić, gdy taką statystykę należy wykonać dla setek lub nawet tysięcy wartości?

Trudność: Jedna zmienna poznanych typów może przechowywać tylko jedną wartość.

Remedium: Zastosować typ tablicowy; zmienna tego typu może przechowywać cały zbiór wartości.

Tablica = uporządkowany zbiór elementów *tego samego typu*, do których dostęp uzyskuje się poprzez podanie ich pozycji.

Wymiary tablicy – liczba wartości koniecznych do określenia położenia elementu w tablicy

Tablica jednowymiarowa (analogia: wektor):

`v` – nazwa całej tablicy

`v[1], ..., v[8]` – poszczególne elementy

23	2	12	34	0	12	4	87

Tablica dwuwymiarowa (analogia: macierz):

`a` – nazwa całej tablicy

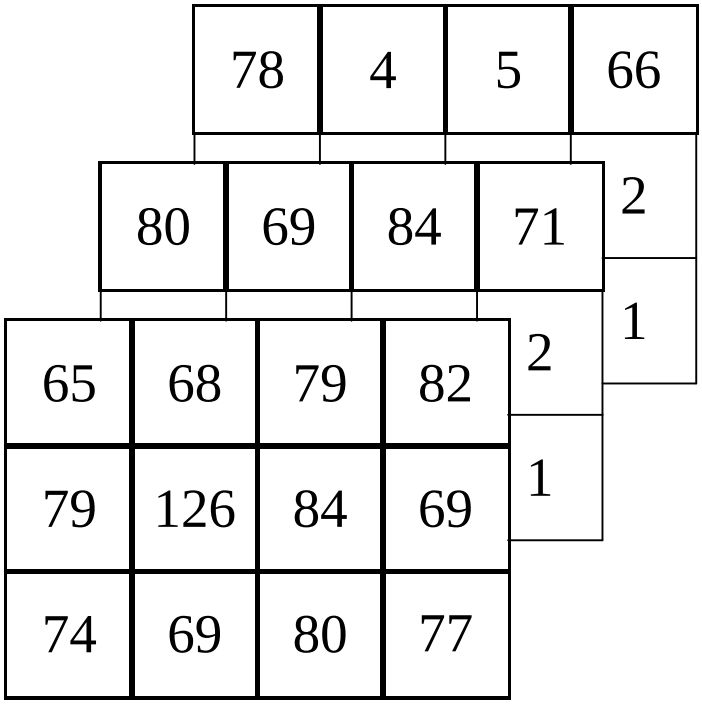
`a[1, 1], ..., a[3, 4]` – poszczególne elementy

2	4	12	5
14	45	0	13
67	35	78	1

Tablica trójwymiarowa:

`c` – nazwa całej tablicy

`c[1, 1, 1], ..., c[3, 4, 3]` – poszczególne elementy



Defincja typu tablicowego

```
ARRAY[<typIndeksu>] OF <typSkladowych>
```

Typ indeksu – najczęściej wykorzystuje się ograniczony typ całkowity:

```
ARRAY[1..100] OF Real
```

Nie zawsze numeracja musi zaczynać się od jedności:

```
ARRAY[1901..2001] OF LongInt
ARRAY[-754..-1] OF Integer
```

Każdemu typowi można nadać nazwę:

```
TYPE
 granice = 1..100;
 wektor = ARRAY[granice] OF Real;
 lZielGory = ARRAY[1901..2001]
                          OF LongInt;
 lRzymu = ARRAY[-754..-1] OF Integer;
VAR
 a, b: wektor;
 c, d: lZielGory;
```

Może być rownież i tak:

```
VAR
 r, t: ARRAY[granice] OF Real;
```

```
 s, q: ARRAY[1901..2001] OF LongInt;
 q, h:  ARRAY[-754..-1] OF Integer;
 k, m: ARRAY[1..50]
        OF (kula, szescian, stozek);
```

Inny przykład: średnie temperatury miesięczne

```
TYPE miesiac = (styczen, luty, ...,
                        ..., grudzien);
VAR t, r: ARRAY[miesiac] OF Real;
    m: miesiąc;
```

W tej sytuacji możliwe jest odwołanie do `t[m]`.

Inne przykłady:

```
VAR
   k: ARRAY[Boolean] OF Integer;
   s: ARRAY[Char] OF Integer;
   b, d: Boolean; sym: Char;

 k[False], k[True], s['d'], s['h']
 k[b OR d], s[sym], s[Succ(sym)]
```

Dopuszczalna jest operacja przypisania

```
b := a;
```

o ile `a` i `b` są tego samego typu tablicowego. Poza tym nie istnieją inne operacje na całych tablicach – ani arytmetyczne, ani porównywania!

```
PROGRAM PowyzejSredniej;
CONST
  liczbaStudentow = 100;
TYPE
  studentInfo
    = ARRAY[1..liczbaStudentow]
                          OF Integer;
  ocenaStudenta = 0..100;
  tablicaOcen
    = ARRAY[1..liczbaStudentow]
                    OF ocenaStudenta;
VAR
  nrStud: studentInfo;
  ocena: tablicaOcen;
  sumaOcen, i: Integer;
  srednia: Real;
```

```
BEGIN
  sumaOcen := 0;
  FOR i := 1 to liczbaStudentow DO
     BEGIN
       WriteLn('Podaj nr studenta',
                   ' i jego ocene');
       ReadLn(nrStud[i], ocena[i]);
       sumaOcen :=
                 sumaOcen + ocena[i]
     END;
  srednia :=
       sumaOcen / liczbaStudentow;
  WriteLn('Numery studentow ',
     'z ocenami powyzej sredniej:');
  FOR i := 1 TO liczbaStudentow DO
    IF ocena[i] > srednia THEN
      WriteLn(nrStud[i])
END.
```

Zadanie. Napisać program, który wyznaczy najczęstszy dzień tygodnia w 2001 roku (1.01.2001 = poniedziałek).

```
PROGRAM MaxDzien;
TYPE
  dzien_tygodnia
    = (pn, wt, sr, cz, pt, so, ni);
  dni = ARRAY [dzien_tygodnia]
                          OF Integer;
VAR
  maxDzien, dzien: dzien_tygodnia;
  liczbaDni: dni;
  k: Integer;
BEGIN
  FOR dzien := pn TO ni DO
    liczbaDni[dzien] := 0;
  FOR k := 1 TO 365 DO
    CASE k MOD 7 OF
      0: liczbaDni[ni]
              := liczbaDni[ni] + 1;
      1: liczbaDni[pn]
              := liczbaDni[pn] + 1;
```

```
      2: liczbaDni[wt]
            := liczbaDni[wt] + 1;
      3: liczbaDni[sr]
            := liczbaDni[sr] + 1;
      4: liczbaDni[cz]
            := liczbaDni[cz] + 1;
      5: liczbaDni[pt]
            := liczbaDni[pt] + 1;
      6: liczbaDni[so]
            := liczbaDni[so] + 1
    END;
k := liczbaDni[pn];
maxDzien := pn;
FOR dzien := wt TO ni DO
    IF k < liczbaDni[dzien] THEN
       BEGIN
         k := liczbaDni[dzien];
         maxDzien := dzien
       END;
Write('Najczestszy dzien to ');
CASE maxDzien OF
    pn: WriteLn('poniedzialek');
    wt: WriteLn('wtorek');
```

```
    st: WriteLn('sroda');
    cz: WriteLn('czwartek');
    pt: WriteLn('piatek');
    so: WriteLn('sobota');
    ni: WriteLn('niedziela')
  END
END.
```

Zadanie. Napisać fragment programu sortujący n-elementową tablicę liczb rzeczywistych `x`.

```
VAR
  v: Real; i, j, k: Integer;
  ...
FOR i := 1 TO n DO
  BEGIN  k := i;
    FOR j := i + 1 TO n DO
      IF x[j] < x[k] THEN k := j;
    v := x[i];
    x[i] := x[k];
    x[k] := v
  END;
```

Tablice wielowymiarowe

```
VAR
  a: ARRAY[1..10]
       OF ARRAY[1..20] OF Real;
  c: ARRAY[pn..so]
       OF ARRAY[Boolean] OF -20..20;
```

lub

```
TYPE
  macierz = ARRAY[1..10]
              OF ARRAY[1..20] OF Real;
  nowyTyp = ARRAY[pn..so]
              OF ARRAY[Boolean]
                OF -20..20;
VAR
  a: macierz;
  c: nowyTyp;
```

Odwołanie:

```
a[5][7]
```

Dopuszczalna jest zwięzła notacja:

```
TYPE
  macierz = ARRAY[1..10, 1..20]
                          OF Real;
  ...
            a[5, 7]
```

Uwaga! Indeksy nie muszą być takiego samego typu!

Zadanie. Dane jest słowo złożone z małych liter, zawierające nie mniej niż 2 litery (kropka kończy słowo). Określić liczbę różnych par kolejnych liter.

Przykład: w słowie **babacabacd** mamy **ba**, **ab**, **ac**, **ca**, **cd**.

```
PROGRAM ParyLiter;
TYPE
  macierz
    = ARRAY['a'..'z', 'a'..'z']
                          OF Boolean;
VAR
  tablica: macierz;
```

```
  c, c1, c2: Char;
  liczbaPar: Integer;
BEGIN
  FOR c1 := 'a' TO 'z' DO
    FOR c2 := 'a' TO 'z' DO
      tablica[c1, c2] := False;
  ReadLn(c1); ReadLn(c2);
  REPEAT
    tablica[c1, c2] := True;
    c1 := c2; ReadLn(c2)
  UNTIL c2 = '.';
  liczbaPar := 0;
  FOR c1 := 'a' TO 'z' DO
    FOR c2 := 'a' TO 'z' DO
      IF tablica[c1, c2] THEN
        liczbaPar := liczbaPar + 1;
  WriteLn('Liczba par = ', liczbaPar)
END.
```

Procedury

Zadanie. Na podstawie danych x i y należy wyznaczyć $u = \max(x + y, xy)$, $v = \max(0.5, u)$.

```
PROGRAM MaxA;
VAR x, y, u, v: Real;
BEGIN
  Read(x, y);
  IF x + y > x * y THEN
     u := x + y
  ELSE
     u := x * y;
  IF 0.5 > u THEN
     v := 0.5
  ELSE
     v := u;
  WriteLn(u, v)
END.
```

Procedury bez parametrów

```
PROGRAM MaxB;
VAR x, y, u, v: Real;
    a, b, s: Real;
BEGIN
  Read(x, y);
  a := x + y; b := x * y;
  IF a > b THEN s := a
           ELSE s := b;
  u := s;
  a := 0.5; b := u;
  IF a > b THEN s := a
           ELSE s := b;
  v := s;
  WriteLn(u, v)
END.
```

Sposób na powtarzający się fragment:

```
PROCEDURE max2A;
  BEGIN IF a > b THEN s := a
                 ELSE s := b
  END;
```

```
PROGRAM MaxC;
VAR x, y, u, v: Real;
    a, b, s: Real;
PROCEDURE max2A;
  BEGIN IF a > b THEN s := a
                 ELSE s := b
  END;
BEGIN
  Read(x, y);
  a := x + y; b := x * y;
  max2A; u := s;
  a := 0.5; b := u;
  max2A; v := s;
  WriteLn(u, v)
END.
```

Wada: Danymi mogą być tylko wartości zmiennych `a` i `b`.

Rozwiązanie:

```
PROCEDURE max2B(r1, r2: Real);
  BEGIN IF r1 > r2 THEN s := r1
                   ELSE s := r2
  END;
```

Przekazywanie parametrów przez wartość

```
PROGRAM MaxD;
VAR x, y, u, v: Real;
    s: Real;
PROCEDURE max2B(r1, r2: Real);
  BEGIN IF r1 > r2 THEN s := r1
                   ELSE s := r2
  END;
BEGIN
  Read(x, y);
  max2B(x + y, x * y); u := s;
  max2B(0.5, u); v := s;
  WriteLn(u, v)
END.
```

Wykonanie polecenia `max2B(x + y, x * y)` jest równoważne wykonaniu bloku

```
VAR r1, r2: Real;
BEGIN r1 := x + y; r2 := x * y;
BEGIN IF r1 > r2 THEN s := r1
                 ELSE s := r2 END
END;
```

Pytanie 1: Jaki wpływ na wartości `x` i `y` miałoby wywołanie `max2C(x + y, x * y)`?

```
PROCEDURE max2C(r1, r2: Real);
  BEGIN IF r1 > r2 THEN s := r1
                   ELSE s := r2;
        r1 := 0; r2 := 0
  END;
```

Wykonanie polecenia `max2C(0.5, u)` jest równoważne wykonaniu bloku

```
VAR r1, r2: Real;
BEGIN r1 := 0.5; r2 := u;
BEGIN IF r1 > r2 THEN s := r1
                 ELSE s := r2 END;
      r1 := 0; r2 := 0
END;
```

Pytanie 2: Czy do wartości `r1` i `r2` mamy dostęp po zakończeniu procedury `max2C`?

Przekazywanie parametrów przez zmienną

```
PROGRAM MaxE;
VAR x, y, u, v: Real;
PROCEDURE max2D(r1, r2: Real;
                    VAR rez: Real);
  BEGIN IF r1 > r2 THEN rez := r1
                    ELSE rez := r2
  END;
BEGIN
  Read(x, y);
  max2D(x + y, x * y, u);
  max2D(0.5, u, v); WriteLn(u, v)
END.
```

Wykonanie polecenia `max2B(x + y, x * y, u)` jest równoważne wykonaniu bloku

```
VAR r1, r2: Real;
BEGIN r1 := x + y; r2 := x * y;
BEGIN IF r1 > r2 THEN u := r1
                 ELSE u := r2 END
END;
```

Zadanie. (*Programowanie techniką zstępującą*) Napisać program wczytujący trzy dodatnie liczby całkowite reprezentujące długości boków trójkąta, a następnie określający, czy trójkąt jest równoboczny, równoramienny lub też różnoboczny.

```
PROGRAM Trojkaty;
...
PROCEDURE czytajBoki(VAR bok1,
                     bok2, bok3: Integer);
   ...
PROCEDURE klasyfikuj(bok1, bok2,
                          bok3: Integer;
                    VAR kl: typTrojkata);
   ...
PROCEDURE poinformujUzytkownika(
                       kl: typTrojkata);
   ...
BEGIN
  czytajBoki(bok1, bok2, bok3);
  klasyfikuj(bok1, bok2, bok3, kl);
  poinformujUzytkownika(kl)
END.
```

Idąc dalej:

```
PROCEDURE czytajBoki(VAR bok1,
                bok2, bok3: Integer);
  PROCEDURE zapytaj(
                   VAR bok: Integer);
    ...
BEGIN
  zapytaj(bok1);
  zapytaj(bok2);
  zapytaj(bok3);
END;
```

Szczegółowa implementacja:

```
PROGRAM Trojkaty;
TYPE typTrojkata = (rownoboczny,
       rownoramienny, roznoboczny);
VAR b1, b2, b3: Integer;
    wynik: typTrojkata;

PROCEDURE czytajBoki(VAR bok1,
                bok2, bok3: Integer);
```

```
  PROCEDURE zapytaj(
                    VAR bok: Integer);
  BEGIN
    WriteLn('Podaj dlugosc boku:');
    ReadLn(bok)
  END; {zapytaj}
BEGIN
  zapytaj(bok1);
  zapytaj(bok2);
  zapytaj(bok3)
END; {czytajBoki}

PROCEDURE klasyfikuj(bok1, bok2,
                     bok3: Integer;
                VAR kl: typTrojkata);
BEGIN
  IF (bok1 = bok2)
        AND (bok1 = bok3) THEN
    kl := rownoboczny
  ELSE IF (bok1 = bok2) OR
          (bok1 = bok3) OR
          (bok2 = bok3) THEN
    kl := rownoramienny
```

```
  ELSE
    kl := roznoboczny
END; {klasyfikuj}

PROCEDURE poinformujUzytkownika(
                    kl: typTrojkata);
BEGIN
  CASE kl OF
    rownoboczny:
      WriteLn('Tr. rownoboczny');
    rownoramienny:
      WriteLn('Tr. rownoramienny');
    roznoboczny:
      WriteLn('Tr. roznoboczny')
  END
END; {poinformujUzytkownika}

BEGIN
  czytajBoki(b1, b2, b3);
  klasyfikuj(b1, b2, b3, wynik);
  poinformujUzytkownika(wynik)
END.
```

Parametry mogą być również typów złożonych . . .

Zadanie. Napisać procedurę wyznaczającą wartości

$$s = a_1^2 + a_2^2 + \cdots + a_n^2$$

$$t = \max(|a_1|, |a_2|, \ldots, |a_n|)$$

```
CONST n = 10;
TYPE tablica = ARRAY[1..n] OF Real;
...
PROCEDURE r(a: tablica;
                VAR s, t: Real);
VAR i: Integer; u: Real;
BEGIN
  s := Sqr(a[1]); t := Abs(a[1]);
  FOR i := 2 TO n DO
    BEGIN s := s + Sqr(a[i]);
      u := Abs(a[i]);
      IF u > t THEN t := u
    END
END;
```

Uwaga! Nie można napisać wprost:

```
PROCEDURE r(a: ARRAY[1..n] OF Real;
            VAR s, t: Real);
```

Poza tym, gdy `n` jest duże, lepiej zapisać

```
PROCEDURE r(VAR a: tablica;
                VAR s, t: Real);
```

Jak zinterpretować wywołanie `r(x, p, q)`?

```
BEGIN a := x;
  p := Sqr(a[1]); q := Abs(a[1]);
  FOR i := 2 TO n DO
    BEGIN p := p + Sqr(a[i]);
      u := Abs(a[i]);
      IF u > q THEN q := u
    END
END;
```

Funkcje

Przykłady standardowych: `Sqrt`, `Sin`, itd.

Załóżmy, że chcemy obliczyć wartość

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Interpretacja: Dla zbioru $\{A, B, C\}$ mamy $\binom{3}{2} = \frac{3!}{2!(3-1)!} = 3$ podzbiory dwuelementowe: $\{A, B\}, \{A, C\}, \{B, C\}$.

Program byłby dużo prostszy, gdyby w Pascalu istniała odpowiednia funkcja standardowa, np. `silnia`:

```
PROGRAM Kombinacje;
VAR n, k, komb: Integer;
BEGIN
  Read(n, k);
  komb := silnia(n) DIV
        (silnia(k) * silnia(n - k));
  WriteLn('Liczba kombinacji:', komb)
END.
```

Co prawda takiej funkcji nie ma, ale można ją napisać:

```
FUNCTION silnia(x: Integer): Integer;
VAR f, i: Integer;
BEGIN
  f := 1;
  FOR i := 1 TO x DO f := f * i;
  silnia := f
END;
```

Inny przykład: Obliczanie pola trójkąta na podstawie długości jego boków a, b i c wg wzoru Herona:

$$S = \sqrt{p(p-a)(p-b)(p-c)}, \quad p = \frac{a+b+c}{2}$$

```
FUNCTION trojkat(a, b, c: Real): Real;
VAR p: Real;
BEGIN p := (a + b + c) / 2;
  trojkat := Sqrt(p * (p - a)
                  * (p - b) * (p - c))
END;
```