Instrukcje podsumowanie

Proste:

- przypisania
- wejścia-wyjścia (read, readln, write, writeln)
- pusta - po prostu ; (średnik)

Strukturalne:

- **grupująca**

  begin <*zestaw instrukcji*> **end**

- **warunkowa**

  if <*wyrażenie*> **then** <*instrukcja1*>;

  lub

  if <*wyrażenie*> **then** <*instrukcja1*>
  else <*instrukcja2*>;

- **pętli**

  repeat
  <*instrukcje*>
  until <*warunek_końca*>;

  while <*warunek kontynuacji*> **do**
  <*instrukcja*>;

```
suma := 0;
writeln('Liczby zakończ 0');
readln(a)
while a <> 0 do
    begin
    suma := suma + a;
    readln(a);
    end;
writeln('Suma=', suma:8:2);
```

```
suma := 0;
writeln('Liczby zakończ 0');
repeat
    readln(a);
    suma := suma + a;
until a = 0;
writeln('Suma=', suma:8:2);
```

```
for <zmienna> := <wyr_pocz> to <wyr_kon> do <instrukcja>;

for <zmienna> := <wyr_pocz> downto <wyr_kon> do <instrukcja>;
```

Przykłady

```
for i := 1 to n do
   writeln('i=', i, ' i*i=', i*i);
```

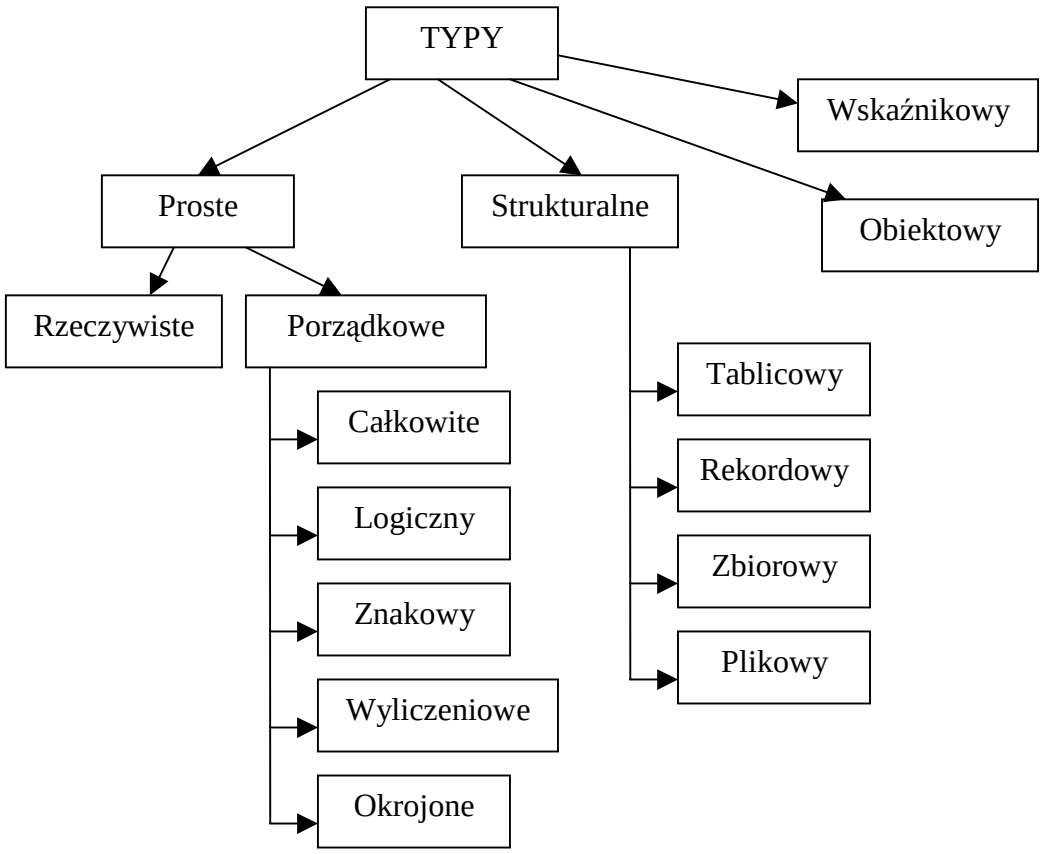
Ile dodatnich z 2, 3, 4 liczb

```
readln(a, b);
ile := 0;
if a > 0 then ile := ile+1;
if b > 0 then ile := ile+1;
writeln('Ilosc liczb dodatnich: ', ile);
```

Ile dodatnich z N liczb:

```
readln(n);
ile := 0;
for i := 1 to n do
     begin
     readln(a);
     if a > 0 then ile := ile+1;
     end;
writeln('Ilosc liczb dodatnich: ', ile);
```

Klasyfikacja typów danych w języku Pascal:



Definiowane typy proste w pascalu.

Typ okrojony:

[*<wartosc_dolna>* **..** *<wartosc_gorna>* **]**

Typ taki obejmuje wartości określone zakresem. Oba ograniczenia muszą być jednakowego typu prostego nie rzeczywistego.

Typ wyliczeniowy

(*<stała1>* **,** *<stała2>* **, ...)**

Stałe muszą być identyfikatorami. Typ zawiera tyle wartości ile jest wymienione w deklaracji. Deklaracja ta służy jednocześnie do zdefiniowania stałych.

Problem:

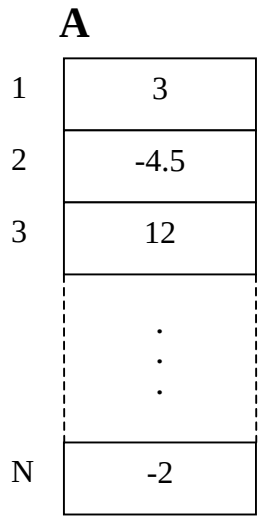
Dane jest N liczb. Obliczyć ile z nich jest większe od średniej wszystkich liczb.

```
readln(n);
suma := 0;
for i := 1 to n do
     begin
     readln(a);
     suma := suma+a;
     end;
srednia:=suma/n;
ile := 0;
for i := 1 to n do
     begin
     readln(a);
     if a > srednia then ile := ile+1;
     end;
writeln('Ilosc liczb wiekszych od sredniej: ', ile);
```

Zamieszczony wyżej przykład wymaga dwukrotnego podawania liczb przez użytkownika co jest niedopuszczalne. Komputer powinien raz wczytać liczby, zapamiętać je, by następnie można było wielokrotnie z nich korzystać.

Tablice

Tablica to pojemnik na ciąg wartości (liczb). Pojemnik ma jedną nazwę, a składa się z wielu jednakowych komórek Każda komórka przeznaczona jest do przechowywania jednego elementu tablicy.
Komórki są ponumerowane (indeksowane). Indeksami komórek są zwykle kolejne liczby całkowite.



Element tablicy identyfikujemy podając jej nazwę i indeks elementu. Obowiązuje składnia:

```
<nazwa tablicy> [ <indeks elementu> ]
```

Na przykład

A[1] element 1-szy
A[2] element 2-gi

A[i+1] element i+1 –szy

Deklaracja tablic

Obowiązująca składnia opisu typu tablicowego:

```
array [ <typ_porządkowy>, ... ] of <typ_elementu>;
```

Tablica może mieć wiele wymiarów.

Deklaracja tablicy jednowymiarowej:

```
array [ <typ porządkowy> ] of <typ elementu>
```

<typ porządkowy> określa zakres indeksów tablicy (a więc także liczbę elementów tablicy)

<typ porządkowy to najczęściej okrojenie.

Przykłady deklaracji tablic:

```
Type
     Tab1 = array [1 .. 10] of char;
     Tab2 = array[ 1.. 10, 1..10 ] of integer;
Var
     Nap: array [ 1..20 ] of Tab1;
     X, Y : array [ 1 .. 10 ] of real;
     Y1, Y2 : Tab2;
```

Można deklarować tablice o bardzo złożonej strukturze. Najczęściej używane są jednowymiarowe o indeksach wyrażanych liczbami całkowitymi od 1.

W obsłudze tablic przydatna jest instrukcja **for**.

Rozwiązanie problemu (liczba większych od średniej) z wykorzystaniem tablic:

```
var  i, n, ile: integer;
     suma, srednia: real;
     a: array [1..100] of real;
begin
writeln('Podaj liczbe elementow');
readln(n);
writeln('Podaj elementy');
for i := 1 to n do           { wczytywanie tablicy }
     readln(a[i]);
suma := 0;
for i := 1 to n do           { sumowanie elementow tablicy }
     suma := suma+a[i];
srednia:=suma/n;
ile := 0;
for i := 1 to n do       { zliczanie liczby el. > sredniej }
     if a[i] > srednia then ile := ile+1;
writeln('Ilosc liczb wiekszych od sredniej: ', ile);
end.
```

Przykład – wyznaczenie maksimum z użyciem tablic:

```
var  i, n, imax: integer;
     max: real;
     a: array [1..100] of real;
begin
writeln('Podaj liczbe elementow');
readln(n);
writeln('Podaj elementy');
for i := 1 to n do
     readln(a[i]);
max := a[1]; imax:=1;
for i := 1 to n do
     if a[i] > max then
          begin max := a[i]; imax := 1; end;
writeln('Najwiekszy element na pozycji: ', imax,
                          ' ma wartosc = ', max:8:2);
end.
```

Procedury i funkcje

Przykład – pole trójkąta na podstawie 3 boków wykorzystaniem funkcji

```
var a, b, c, p: real;

function pole (a, b, c: real) : real;
var p: real;
begin
p:=(a + b +c)/2;
pole:=sqrt((p-a)*(p-b)*(p-c)*p);
end;

begin
writeln('Podaj boki');
readln(a, b, c);
p:=pole(a, b, c);
writeln('Pole = ', p:8:2);
readln;
end.
```

Składnia funkcji

```
function <nazwa>(<lista_parametrów>) : <typ_wyniku>;
 <deklaracje_lokalne>
begin
<instrukcje>
<nazwa> := <wyrażenie>;        { lub return <wyrażenie>; }
end;
```

Przykład – rysowanie gwiazdek

```
var k: integer;
procedure gwiazdki (n: integer);
var i: integer;
begin
for i := 1 to n do write('*');
writeln;
end;
begin
writeln('Podaj liczbe');
readln(k);
gwiazdki(k);
readln;
end.
```

```
var i, n: integer;
     a: array [1..100] of integer;
procedure gwiazdki (n: integer);
var i: integer;
begin
for i := 1 to n do write('*');
writeln;
end;
begin
writeln('Podaj liczbe paskow');
readln(n);
writeln('Podaj ilosci gwiazdek');
for i := 1 to n do
     readln(a[i]);
for i := 1 to n do  gwiazdki(a[i]);
readln;
end.
```

Składnia procedury

```
procedure <nazwa>(<lista parametrów>);
 <deklaracje_lokalne>
begin
<instrukcje>
end;
```

Przykład – wyznaczenie maksimum z użyciem procedur:

```
type t1=array [1..100] of real;
var  i, n: integer;
     max: real;
     a: t1;
procedure emax(n: integer; x: t1; var max: real);
var i: integer;
begin
```

```
max := a[1];
for i := 1 to n do
     if a[i] > max then max := a[i];
end;
begin
write('Podaj liczbe elementow: ');
readln(n);
writeln('Podaj elementy');
for i := 1 to n do
     readln(a[i]);
writeln('Poczatek obliczen');
Emax(n, a, max);
writeln('Najwiekszy element =', max:8:2);
readln;
end.
```

Typ rekordowy

Rekordem nazywa się strukturę, której składowe, zwane polami, mogą być rożnych typów. Definicja typu rekordowego specyfikuje typ każdego pola oraz identyfikatory tych pol.

Składnia deklaracji rekordu

```
record
<lista_nazw_pol1> : <typ1>
<lista_nazw_pol2> : <typ2>;
 . . .
end;
```

Przykład deklaracji:

```
var  s1: record
         nazwisko, imie: string;
         wiek: integer;
         stypendium: real;
         end;
```

lub

```
type student=  record
               nazwisko, imie: string;
               wiek: integer;
               stypendium: real;
               end;
var s1, s2: student;
```

Poszczególne pola mogą być same obiektami złożonymi: tablicami, rekordami itp.
Liczba i struktura pól rekordu jest ustalona w deklaracji i nie może być potem zmieniana.

Bardziej złażone przykłady:

```
type data= record
           dzien: 1..31;
           miesiac: 1..12;
           rok: integer;
           end;
    punkt= record
           x, y: real
           end;
    okrag= record
           promien: real;
           srodek: punkt
           end;
  student= record
           nazwisko: string [20];
           imie: string [15];
           plec: boolean;
           ocena: real;
           dataur: data
           end;
```

Do poszczególnych pól odwołujemy się za pomocą deskryptorów pól zapisywanych z użyciem notacji kropkowej:

```
<nazwa rekordu> . <nazwa pola>
```

Deskryptor pola (desygnator pola) wskazuje pole o podanym identyfikatorze w danej zmiennej rekordowej. Zmienna ta może być zmienną całościową lub też może być elementem innej struktury złożonej (tablicy lub rekordu).

Zakładając definicje typów jak wyżej oraz deklaracje zmiennych:

```
   var  urodziny: data;
            P, Q: punkt
              OK: okrag;
         Kowalski: student;
            grupa: array [1..30] of student;
```

możemy, przykładowo, użyć następujących deskryptorów:

```
urodziny.dzien
P.y
ok.srodek.x
grupa[12].ocena
Kowalski.dataur
grupa[4].dataur.rok
```

Przykład w programie:

```
type trojkat =  record
          a, b, c: real;
          end;
function obwod(t: trojkat): real;
begin
obwod:=t.a+t.b+t.c;
end;
var tr: trojkat;

begin
readln(tr.a, tr.b, tr.c);
writeln('Obwod=', obwod(tr));
 . . . .
```

Obiekty

Żeby umożliwić rozdzielne rozwiązywanie zagadnień na które podzielono złożony problem w sposób doskonalszy, pozwalający na zupełnie dowolne dobieranie oznaczeń (także procedur i funkcji) należy zastosować programowanie obiektowe.

```
type trojkat = object
               a, b, c: real;
               function obwod:real;
               end;
function trojkat.obwod:real;
begin
obwod:=a+b+c;
end;
var t: trojkat;

begin
readln(t.a, t.b, t.c);
writeln('Obwod=', t.obwod);
 . . . .
```

Jak widać z powyższego przykładu obiekt jest rozwinięciem idei rekordu. Pozwala na zamknięcie w jednym pojemniku wszystkich nazw wykorzystywanych do opisu problemu: zarówno zmiennych-pól jak i nazw procedur.

Typy obiektowe w Pascalu definiowane są z wykorzystaniem słowa kluczowego **object** zgodnie ze składnią:

type <identyfikator typu obiektowego> = **object** [(nazwa przodka)]
[<lista deklaracji pól obiektu>]
[< lista zapowiedzi metod obiektu>]
end;

W prostym przypadku gdy w deklaracji nie umieszczono nazwy przodka oraz nie zdefiniowano żadnej metody obiekt ma budowę i własności rekordu. Na przykład:

```
type TArgumenty = object
                  a, b, c: Real;
                  end;
```

Definicja typu obiektowego umieszczona powyżej różni się od definicji rekordu użyciem słowa kluczowego **object**. Pola tak zdefiniowanego obiektu można wykorzystywać jedynie w sposób rekordowy.

W przykładzie typu Trojkat można zaobserwować charakterystyczny sposób posługiwania się obiektami i składowymi obiektów. Podstawowe zasady można sformułować następująco:

- W deklaracji typu obiektowego w pierwszej kolejności wymieniane są pola obiektu, a w drugiej metody.

- Każda metoda deklarowana jest w typie obiektowym w postaci zapowiedzi zawierającej jedynie nagłówek metody.

- Kompletna deklaracja metody podawana jest odrębnie poza deklaracją typu obiektowego. Nazwa metody jest uzupełniana prefiksem – nazwą typu obiektowego. Lista parametrów w nagłówku nie może być zmieniona w stosunku do poprzedzającej zapowiedzi. Może być jedynie powtórzona bez zmian lub opuszczona w całości.

- Typy obiektowe mogą być deklarowane tylko globalnie, w programie lub w module lecz nie wewnątrz procedury.

- Zmienne obiektowe muszą być deklarowane przez wykorzystanie identyfikatora typu (zdefiniowanego zawsze globalnie), a nie opisu typu.

- Metoda należy do obiektu i w związku z tym posiada dostęp do pól i innych metod obiektu bezpośrednio. W treści procedur można korzystać z nazw pól bez konieczności dopisywania prefiksów lub umieszczania ich w parametrach metod - obydwa rozwiązania są nawet zabronione. Np. niedopuszczalny jest zapis:

  ```
  procedure. Trojkat obwod;
  begin
  Trojkat obwod := Trojkat.a + Trojkat.b + Trojkat.c;          {błędne odwołania do pól}
  end;
  ```

- Poza definicją typu obiektowego i definicjami związanych z nim metod przy odnoszeniu się do pól lub metod obiektu należy używać nazw dwuczęściowych:

  <nazwa zmiennej obiektowej> **.** < nazwa pola lub metody>

  Podobnie jak w przypadku rekordów możliwe jest uproszczenie dostępu do składowych obiektu przy wykorzystaniu instrukcji **with**.

- Zmienne lokalne i parametry metody nie mogą mieć tych samych nazw co pola obiektu.

- W metodach można wywoływać inne metody obiektu nawet te zdefiniowane szczegółowo później (bez prefiksu, wystarczy nazwa metody).

```
type trojkat =  object
                a, b, c: real;
                function obwod: real;
                function pole: real;
                end;

function trojkat.obwod:real;
begin
obwod:=a+b+c;
end;

function trojkat.pole:real;
var p: real;
begin
p:=obwod/2;
pole:=sqrt(p*(p-a)*(p-b)*(p-c));
end;
var t: trojkat;

begin
readln(t.a, t.b, t.c);
writeln('Obwod=', t.obwod, ' Pole=', t.pole);
. . . .
```

Wszystkie metody posiadają niejawny parametr Self reprezentujący obiekt przekazywany przez zmienną. Oznacza to, że do składowych obiektu można odwoływać się także z użyciem prefiksu Self:

Self **.** <nazwa składowej>