

7 Typy danych c.d.

7.1 Typy wprowadzone wcześniej

- Typy całkowite
Byte, Short, Integer, Word, Longint
- Typy rzeczywiste
Real, single, double, extended
- Typ Logiczny Boolean
- Typ Znakowy Char
- Typ Napisowy String
Typ String jest równoważny String[255] co oznacza tablicę 255 znaków.

7.2 Typ wyliczeniowy

Typ definiowany przez użytkownika.

```
Type Dni=(Poniedzialek, Wtorek, Sroda, Czwartek,  
         Piątek, Sobota, Niedziela);  
var d:dni;
```

```
begin  
  for d:= Poniedzialek to piątek do  
    writeln ('Dziś do pracy ');  
end.
```

7.3 Typ okrojony

Typ okrojony może być utworzony z dowolnego typu prostego (całkowite, logiczny, znakowy, wyliczeniowy) poprzez zawężenie jego zakresu do jawnie podanego.

```
Type LiteraMała = 'a'..'z';  
   Cyfra        = 0..9;  
   Dzienroboczy = Poniedzialek..Piątek;
```

Wielkość zajętej pamięci przez typ okrojony jest taka sama jak przez typ bazowy.

7.4 Typ zbiorowy

Type Nazwa = Set of Typ_ Elmenetowy

Typ zbiorowy umożliwia na operowanie podzbiorach typu Typ_Elementowy. Typ_Elementowy musi być typem porządkowym o co najwyżej **256** różnych wartościach. **Może** to być typ okrojony bądź wyliczeniowy.

Nie może to być typ Integer – bo zawiera za dużo możliwych wartości.

```
Type
Dni=(Poniedzialek,Wtorek,Sroda,Czwartek,Piatek,
      Sobota,Niedziela);
Robocze =set of Dni;

var d : dni;
    r : Robocze;

begin
    r:=[];
    r :=r+ [poniedzialek..Piatek];
    randomize;
    d:=dni(random(7)+1);
    if d in r then
        writeln (' Dzień roboczy',byte(d))
    else
        writeln('Dzień wolny', byte(d));

end.
```

7.5 Typ tablicowy

Jest to typ złożony.

```
Array[typ_indeksowy]of typ_danych;
```

Przykład:

```
Type Liczby= Array[1..10] of byte;  
   Nazwiska= Array[100..1000] of String;  
   Dane=Array[Boolean]of Real;  
   Zadania=Array[Poniedzialek..Piatek]of String;
```

Typ danych może być dowolnym wcześniej określonym typem.

```
Program T1;                               Program T1a;  
Var                                         Var  
   a:array[1..5] of                       a:array[1..5] of  
Integer;                                   Integer;  
                                           i:byte;  
Begin                                       Begin  
   ReadLn(a[1]);                           for i :=1 to 5 do  
   ReadLn(a[2]);                           ReadLn(a[i]);  
   ReadLn(a[3]);                           for i :=5 downto 1 do  
   ReadLn(a[4]);                           WriteLn(a[i]);  
   ReadLn(a[5]);                           End.  
   WriteLn(a[5]);  
   WriteLn(a[4]);  
   WriteLn(a[3]);  
   WriteLn(a[2]);  
   WriteLn(a[1]);  
End.
```

Zmienna typu tablicowego w pamięci zajmie
Wielkość jednego elementu * Ilość Elementów

W powyższym przypadku $\text{SizeOf}(\text{Integer}) * 5$ czyli $2*5 = 10$.

7.5.1 String

Typ `String` jest równoważny `String[255]` co oznacza tablicę 255 znaków. Jest to maksymalna długość pojedynczego łańcuch. Można ograniczyć go poprzez konstrukcję:

```
String[B]
```

gdzie B jest typu `Byte` 0..255

```
var   Imie           : String[15];
      Nazwisko       : String[25];
```

Typ `String` jest specyficznym typem tablicowym.

Dopuszczalne są oracje na zmiennej

Operatory:

```
+   =   <>   <   >   <=   >=
```

```
var S: String;
```

```
Read(s);
```

```
Write(s);
```

```
S:='Ala ma kota'
```

```
S:='Coś innego' + ' może jeszcze coś';
```

```
If s => 'tekst' then Delete(s,2,10);
```

Funkcje i procedury operujące na elementach typu `String`:

```
f.Length(S: String): Integer;
```

```
f.Copy(S: String; Index: Integer; Count:
      Integer): String;
```

```
f.Concat(s1 [, s2, ..., sn]: String): String;
```

```
f.Pos(Substr: String; S: String): Byte;
```

```
p.Insert(Source: String; var S: String;
      Index: Integer);
```

```
p.Delete(var S: String; Index: Integer;
      Count:Integer);
```

```
Pos(Substr: String; S: String): Byte;
```

```
var
```

```
  s: string;
```

```
begin
```

```
  s := 'Ala ma kota';
```

```
  Delete(s,5,3); Writeln(s); { 'Ala kota' }
```

```
  Insert('znalazła',s,5);
```

```
  Writeln(s); { 'Ala Znalazła kota' }
```

```
end.
```

7.5.2 Tablice wielowymiarowe

W konstrukcji

```
Array[typ_indeksowy] of typ_danych;
```

nie powiedziano o typ_danych. Może to być dowolny określony wcześniej, lub w tym momencie, typ danych. W szczególności może to być typ tablicowy.

7.5.2.1 Tablice dwu-wymiarowe

```
Array[zakres1] of array [zakres2] of typ_danych
```

```
Array[zakres1, zakres2] of typ_danych
```

```
T2a=Array [1..10, 1..20] of Real;
```

```
T2b=Array [ char, 1..3] of Char;
```

```
T2c=Array [32..255, Boolean] of Byte;
```

Ilość elementów tablicy równa się iloczynowi mnogości typów indeksujących. Np. T2a posiada 10*20 czyli 200 elementów * 4B(typ Real), czyli łącznie 800B.

Cała tablica musi zmieścić się w jednym segmencie danych 64KB.

Przykład:

```
Var T: array['a'..'z', 1..3] of byte;
    C:char; I:byte;
Begin
    Randomize;
    For c:= 'a' to 'z' do
        For I:=1 to 3 do
            T[c, I]:=Random(2);
        For I:=1 to 3 do
begin
    For c:= 'a' to 'p' do
        If T[c, I]>0 then Write('*')
        Else write(' ');
    Writeln;
    End;
END.
```

Co oznaczają symbole T, T[c] , T[c, i].

Przykład:

Program Przekatne;

```
const N=6; M=7;
```

```
Type Tablica2= Array[1..N,1..M] of char;
```

```
var
```

```
  T : Tablica2;
```

```
  i,j,k,p:Integer;
```

```
BEGIN
```

```
  Randomize; k:=1;
```

```
  For i:=1 to N do
```

```
    for j:=1 to M do
```

```
      T[i,j]:= chr(random(255-32)+32);
```

```
  for i:=1 to N do begin
```

```
    for j:=1 to M do
```

```
      write(ord(t[i,j]):4);
```

```
    writeln
```

```
  end;
```

```
  if N<M then k:=N else k:=M;
```

```
  for i:=1 to K do
```

```
    writeln(ord(t[i,i]):i*4);
```

```
  for i:=1 to K do
```

```
    writeln(ord(t[i,M-i+1]):(M-i+1)*4);
```

```
  p:=K div 2;
```

```
  for i:=1 to p do
```

```
    writeln(ord(T[i,i]):i*4,
```

```
      ord(T[i,M-i+1]):(M-2*i+1)*4);
```

```
  if odd(m) then
```

```
    begin
```

```
      inc(p);  writeln(ord(T[p,p]):p*4)
```

```
    end;
```

```
  for i:=p+1 to K do
```

```
    writeln(ord(t[i,M-i+1]):(M-i+1)*4,
```

```
      ord(T[i,i]):(2*i-M-1)*4);
```

```
END.
```

7.5.2.2 Algorytmy dotyczące tablic

```
PROGRAM SortowanieBabel ;
CONST   N = 10 ;
VAR     A : ARRAY[1..N] OF Integer ;

      I,ZAM: Integer ;
      B : Integer ;
BEGIN
  Randomize;
  FOR I := 1 TO N DO
    A[I]:=Random(199)-100;
  Writeln('Wylosowany wektor:') ;
  Writeln ;
  FOR I := 1 TO N DO
    Write(A[I]:5) ;

  REPEAT
    ZAM := 0 ;

    FOR I := 1 TO N - 1
    DO BEGIN
      IF A[I]> A[I+1]
      THEN BEGIN
        B      := A[I];
        A[I]   := A[I+1];
        A[I+1] := B;
        ZAM    := ZAM + 1
      END
    END
  UNTIL ZAM = 0 ;

  Writeln('Wektor po sortowaniu:') ;
  Writeln ;
  FOR I := 1 TO N DO
    Write(A[I]:5) ;
END.
```

7.5.2.3 Tablice n-wymiarowe

Type

```
Pozycja =[-10..10,-10..10,-10..10] of real;
Tab3= Array [1..3,1..10,1,5] of Boolean;
Tab4= Array[Boolean,1..3,6..10,2..4]of Real;
Tab5= Array[1..2,1..10,1..5,1..4,0..9] of
Integer;
```

Type

```
Punkt      = Array[1..3]    of Real;
Odcinek    = Array[1..2]    of Punkt;
Figura     = Array[1..4]    of Punkt;
Bryla      = Array[1..6]    of Figura;
Ruch       = Array[1..100]  of Bryla;
```

Nie istnieje ograniczenie, poza zajętością pamięci, na ilość wymiarów tablicy. Jednakże niezmiernie rzadko stosuje się tablice o większej ilości niż 3. Najczęściej pojawiają się tablice jedno i dwu wymiarowe.

Większa ilość wymiarów utrudnia określenie adresu elementu, a także zależności pomiędzy sąsiadami.

Czasami lepszym rozwiązaniem jest skonstruowanie prostszego typu danych.

Założmy za `var B:Bryła;`

Co oznaczają zmienne i czy poprawne są odwołania:

```
B, B[1], B[1,2], B[1,2,3], B[1,2,3,4],
B[1,2,3,4,5]
```

7.5.2.4 Zgodność typów tablicowych

```
Type T1=Array[1..N] of integer;
```

```
T2=Array[1..N] of integer;
```

```
Var a1:T1; a2:T2;
```

```
a1:=a2;
```

```
a1:=T1(a2);
```