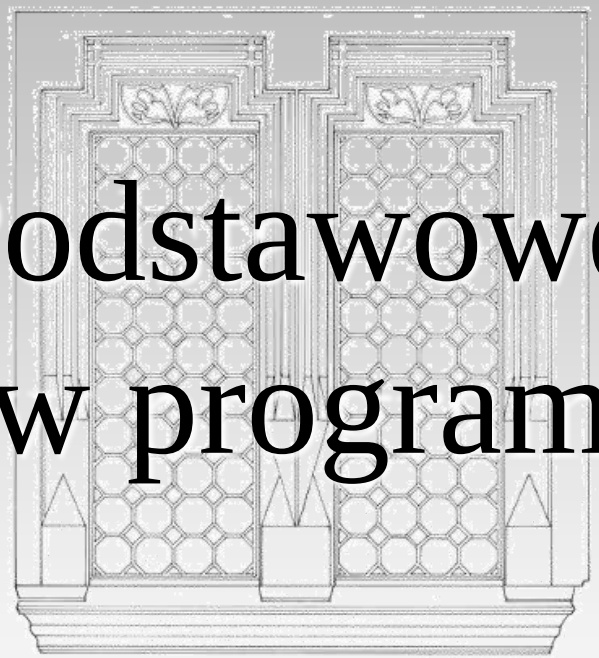

Podstawowe koncepcje w programowaniu (3)

Zmienne tablicowe

var x: **array** [1..10] **of** integer;

Jak się nazywa piąty element tablicy ?

Odpowiedź: x [5] - bo w języku naturalnym i w naszym języku numerujemy poczynając od 1, ale np. w języku C elementy tablicy indeksuje się od 0 i wtedy piąty element tablicy nazywa się x[4].

Problem I: Znajdowanie wartości największego elementu w tablicy

```
max := x[1];
for i := 2   to  10  do
    if  x[i] > max  then
                max := x[i];
```

../Przykłady/MaxValueInArray.pascal

Do tablicy x wstawiono następujące wartości:

indeksy tablicy:	1	2	3	4	5	6	7	8	9	10
wartości w tablicy:	3	1	2	4	7	6	1	8	7	8

Wykonanie algorytmu znajdowania największego elementu tablicy:

	wartość max
przed iteracją:	3
po iteracji dla i=2:	3
po iteracji dla i=3:	3
po iteracji dla i=4:	4
po iteracji dla i=5:	7
po iteracji dla i=6:	7
po iteracji dla i=7:	7
po iteracji dla i=8:	8
po iteracji dla i=9:	8
po iteracji dla i=10:	8

Problem II: Znajdź indeks (czyli położenie) największego elementu w tablicy. Daje więcej informacji niż rozwiązanie problemu I – bo mając indeks możemy również mieć natychmiast wartość.

```
p:= 1 //wartość p będzie indeksem
      //elementu największego
for i := 2  to  10 do
   if  x[i] > x[p]  then  p := i;
```

Uzasadnienie poprawności algorytmu

```
p := 1;
// wartość p będzie indeksem elementu największego
for i := 2  to  10  do
   // teraz x[p] największe spośród x[1],...,x[i-1]
   if  x[i] > x[p]  then p := i;
   // a teraz x[p] największe spośród x[1],...,x[i]
```

Na końcu ostatniej iteracji wartość i=10, zatem po zakończeniu pętli zachodzi: x[p] największe spośród x[1], . . . , x[10]

PYTANIE: Czy wynik będzie ten sam, jeśli zamiast ostrej nierówności użyjemy nierówności nieostrej ? Kiedy wynik będzie inny ?

Problem III praktyczny : Jak wczytywać tablicę ?
Odpowiedź: Wczytywać w pętli kolejne elementy.

Wariant 1 – najprostszy.
Deklarujemy tablice i wypełniamy ją całkowicie.

```
const       n=20;
var         a: array [1..n] of integer;
. . . . .
write ('Podaj kolejne elementy tablicy – 20 liczb');
for i:= 1  to  20  do     read (a[i]);
```

Wariant 2 – użytkownik poda jawnie wielkość tablicy

```
var  a: array [1..1000] of integer; // rozmiar tablicy – na zapas
       n: integer;
. . . . .
write ('Podaj wielkość tablicy, nie więcej niż 1000');
read (n);
for i:= 1  to  n  do       read (a[i]);
// wykorzystuje się tylko początkowy fragment n-elementowy
```

Wariant 3 – użytkownik podaje niejawnie wielkość tablicy

```
var   a: array [1..1000] of integer;     //  na zapas
       liczba, n: integer;
      . . . . .
write('Podawaj kolejne elementy, nie więcej niż 1000');
write('zero kończy i nie jest wczytywane');
n := 0;                          // liczba wczytanych elementów
read(liczba);
while   liczba <> 0 do
  begin
      a [n] := liczba;
      n := n + 1;
      read ( liczba );
  end; // liczba wczytanych elementów (bez zera) = n
```

PROBLEM IV: Wyszukiwanie w tablicy

```
const       n = 1000;                    // wielkość tablicy nie gra roli
var         a: array [1..n] of integer;
            x: integer;

. . . . .   // wczytano zawartość tablicy a i wartość zmiennej x
```

Czy w tablicy jest element o wartości równej wartości zmiennej x?

Inaczej: **Znajdź w tablicy a element o wartości x.**

../Przykłady/SearchValueInArray.pascal

Rozwiązanie najprostsze

Wynikiem algorytmu jest zmienna logiczna, o wartości *true* (prawda) jeśli taki element w tablicy istnieje, *false* (fałsz) w przeciwnym przypadku

```
var  w: boolean; // zmienna w jest typu logicznego
. . . . . .
w := false;                  // ustawienie początkowe
for   i:= 1  to  n do
      if  a[i] = x then  w := true;
```

Jeśli w tablicy nie występuje element o wartości x, to podstawienie "w:=*true*" nie będzie wykonane, i wartość zmiennej w pozostanie "*false*". W przeciwnym przypadku podstawienie będzie wykonane tyle razy ile jest takich elementów w tablicy (co najmniej raz) i wartość zmiennej w po zakończeniu pętli wyniesie „*true*".

Rozwiązanie 2

Wynikiem algorytmu jest zmienna liczbowa, o wartości 0 jeśli takiego elementu w tablicy nie ma, a w przeciwnym przypadku o wartości która jest indeksem szukanego elementu tablicy (czyli wskazuje położenie szukanego elementu).

```
p := 0;
for  i:= 1  to n do
      if  a[i] = x  then p := i
if  p <= 0  then  write('W tablicy nie ma szukanej wartości')
            else  write ('Szukany element znajduje się na pozycji',  p);
```

Rozwiązanie 3 (zalecane, „dojrzałe")

Sprawdzamy kolejne elementy tablicy, i kończymy pętle, gdy napotkamy pierwszy element o wartości x lub gdy osiągniemy koniec tablicy.

Konieczne jest użycie złożonego warunku logicznego, z wykorzystaniem operatorów logicznych.

```
var   a: array [1..n] of integer;
      z: boolean;  // true, jeśli znaleziono
. . . . .
 p := 1;
 z :=  false;    // jak na razie – nie znaleziono
 while   p < = n  and not z  do
     begin
 if  a[p] = x  then  z := true
                 else   p := p + 1;
     end;
```

Jeśli wartość x występuje w tablicy, to przy pierwszym takim wystąpieniu podczas wykonywania pętli następuje podstawienie z:=*true* i pętla się kończy (bo do kontynuacji potrzeba spełnienia OBU warunków). Zmienna p jest w chwili opuszczenia pętli indeksem elementu o wartości x.

Jeśli wartość x nie występuje w tablicy, to za każdym razem zwiększa się wartość p, aż dochodzi do p>n i pętla się kończy, z niezmienioną wartością z=*false*.

Rozwiązanie 3a (podobne, równie dojrzałe)

```
p := 1;
while   p <=n  and a[p] <> x  do  p := p + 1;
// Koniec. Jeśli teraz p>n to znaczy, że nie znaleziono x w a.
// Jeśli p<=n, to znaleziono, i zachodzi a[p] = x
```

Założyliśmy, że w trakcie wykonywania algorytmu, jeśli p>n to drugi warunek już nie jest sprawdzany (bo koniunkcja i tak fałszywa).