

Programowanie 1

Sławomir Pluciński (splucinski@pjawstk.edu.pl)

Tablice

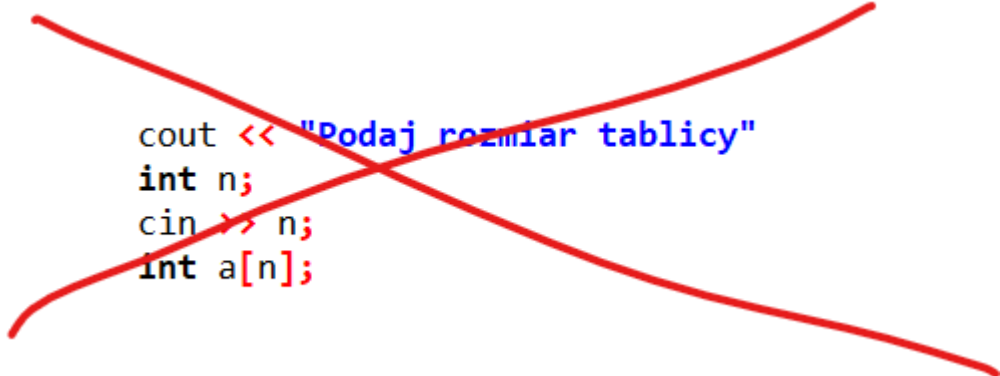
- Tablica jest to zbiór elementów tego samego typu, które zajmują ciągły obszar w pamięci.
- Tablice są typem pochodnym, tzn. buduje się je z elementów jakiegoś typu nazywanego typem składowym.

Przykład:

```
int a[50];  
float tab[20];  
long w[30];  
char tekst[80];
```

Tablice

- Rozmiar tablicy musi być stałą, znaną już w trakcie kompilacji;
- Kompilator musi wiedzieć ile miejsca ma zarezerwować na daną tablicę.
- Rozmiar ten nie może być ustalany dopiero w trakcie pracy programu.



```
cout << "Podaj rozmiar tablicy"  
int n;  
cin >> n;  
int a[n];
```

Typ tablic:

- typy prymitywne (z wyjątkiem void);
- typ wyliczeniowy (enum);
- wskaźniki;
- obiekty typu zdefiniowanego przez użytkownika (czyli klasy);
- tablicowy

Elementy tablicy:

```
int a[5];    //5 elementów typu int
```

```
a[0] a[1] a[2] a[3] a[4]
```

- Numeracja elementów tablicy zaczyna się od zera.
- Element A[5] nie istnieje.
- Próba wpisania jakiejś wartości do A[5] nie będzie sygnalizowana jako błąd.
- W języku C++ zakres tablic nie jest sprawdzany.
- Wpisanie wartości do nieistniejącego elementu A[5] spowoduje zniszczenie w obszarze pamięci wartości, wpisanej bezpośrednio za tablicą.

Inicjowanie tablicy:

Tablicę można zainicjować w momencie definicji.

```
int a[5] = {1, 2, 3, 4, 5};
```

Wynik zainicjowania tablicy:

```
a[0] = 1  
a[1] = 2  
a[2] = 3  
a[3] = 4  
a[4] = 5
```

Inicjowanie tablicy:

- Jeżeli w momencie inicjalizacji na liście jest więcej elementów, niż wynika z definicji to kompilator zasygnalizuje błąd.
- Podczas inicjalizacji kompilator sprawdza, czy nie jest przekroczony rozmiar tablicy.

Możliwe jest takie zainicjowanie tablicy:

```
int a[5] = {1, 2};
```

```
a[0] = 1
```

```
a[1] = 2
```

```
a[2] = 0
```

```
a[3] = 0
```

```
a[4] = 0
```

Inicjowanie tablicy:

Kolejny sposób inicjowania tablicy:

```
int a[] = {1,2,3,4,5};
```

- Kompilator w tym przypadku przelicza, ile liczb podano w klamrach.
- W efekcie rezerwowana jest pamięć na te elementy.

Przekazywanie tablicy do funkcji:

- Tablice w C++ nie są przesyłane do funkcji przez wartość.
- Przez wartość można przesyłać tylko pojedyncze elementy tablicy, ale nie całość.
- Tablice przesyła się podając do funkcji tylko adres początku tablicy.

```
float X[ ] = { 21, 4, 45, 38, 17 };  
void sort ( float X[] );
```

Funkcję Sort wywołujemy w sposób następujący:

```
sort(X);
```

- W języku C++ nazwa tablicy jest jednocześnie adresem elementu zerowego.

- Ponadto wyrażenie: `X + 3`

jest adresem tego miejsca w pamięci, gdzie znajduje się element o indeksie 3, czyli `X[3]`.

- W naszym przykładzie jest to element o wartości 4.

Adres takiego elementu to również: `&X [3]`

- Znak `&` jest jednoargumentowym operatorem oznaczającym uzyskiwanie adresu danego obiektu.
- Zatem poniższe dwa wyrażenia są równoważne:

`X+3 &X[3]`

Tablice znakowe:

Specjalnym rodzajem tablic są tablice do przechowywania znaków

```
char tekst[80]
```

- W tablicach znakowych po ciągu znaków następuje znak o kodzie 0 (znak NULL).
- Znak ten stosuje się do oznaczenia końca ciągu znaków innych niż NULL.
- Ciąg znaków zakończony znakiem NULL nazywamy łańcuchem.

Inicjowanie tablic znakowych:

Tablicę można zainicjalizować w trakcie definicji :

```
char tekst[80] = {"C++"};
```

0	1	2	3	4	77	78	79
C	+	+	NULL						

- nie wymienione elementy inicjuje się do końca pustymi elementami;
- znak NULL został automatycznie dopisany po ostatnim znaku + dzięki temu, że inicjowaliśmy tablicę ciągiem znaków ograniczonym cudzysłowem.

Inicjowanie tablic znakowych:

Jest też inny sposób inicjowania tablicy znaków:

```
char tekst[80] = {'C', '+', '+'};
```

```
tekst[0] = 'C';
```

```
tekst[1] = '+';
```

```
tekst[2] = '+';
```

- Ponadto, ponieważ nie było tu cudzysłowu, kompilator nie dokończył inicjowania znakiem NULL.
- Wszystkie elementy tablicy poczynając od tekst[3] do tekst[79] włącznie zostaną zainicjowane zerami.
- Ponieważ znak NULL ma kod 0 - zatem łańcuch w tablicy tekst zostanie poprawnie zakończony.

Inicjowanie tablic znakowych:

```
char tekst[] = {'C', '+', '+'};
```

- Jest to definicja tablicy znakowej o 3 elementach, w której znajdują się znaki 'C', '+' i '+'.
- Znak NULL tam nie będzie.
- Wniosek - tablica tekst nie przechowuje łańcucha znaków, lecz pojedyncze znaki.

```
char tekst[] = {"C++"};
```

- zostanie zarezerwowana pamięć dla 4 elementów tablicy znakowej tekst.
- kolejne elementy tablicy przechowują następujące znaki: 'C', '+', '+' i NULL.

```
#include <iostream>
#include <conio.h>
using namespace std;
int main ( )
{
    char napis1[ ] = { "Nocny lot" };
    char napis2[ ] = { 'N', 'o', 'c', 'n', 'y', ' ', 'l', 'o', 't' };
    cout << "rozmiar tablicy pierwszej: "
    << sizeof(napis1) << endl;
    cout << "rozmiar tablicy drugiej: "
    << sizeof(napis2) << endl;
    return 0;
}
```

```
rozmiar tablicy pierwszej: 10
rozmiar tablicy drugiej: 9
```

Wpisywanie łańcuchów do tablic:

```
#include <string>
#include <iostream>
```

```
using namespace std;
```

```
int main(){
    char charArray[80];
```

```
    string word = "Nocny lot";
```

```
    for (int i = 0; i < 10; i++){
        charArray[i] = word[i];
    }
```

```
    for (int i = 0; i < 10; i++){
        cout << charArray[i];
    }
```

```
}
```

```
tekst [80] = "Nocny lot"; // błęd
tekst = "Nocny lot";      // błęd
```


Tablice wielowymiarowe

Tablice można tworzyć z różnych typów obiektów, w tym również z innych tablic np. :

```
int X [4][3];
```

Tablica **X** składa się z **4** wierszy i **3** kolumn:

X [0] [0]	X [0] [1]	X [0] [2]
X [1] [0]	X [1] [1]	X [1] [2]
X [2] [0]	X [2] [1]	X [2] [2]
X [3] [0]	X [3] [1]	X [3] [2]

Tablice wielowymiarowe

- Elementy tablicy umieszczają się w pamięci komputera tak, że najszybciej zmienia się najbardziej skrajny prawy indeks tablicy.
- Zatem poniższa inicjalizacja zbiorcza:

```
int X [4] [3] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
```

spowoduje, że elementom tej tablicy zostaną przypisane wartości początkowe:

1	2	3
4	5	6
7	8	9
10	11	12

Tablice wielowymiarowe

W tablicy `int X[4][3]` element `X[1][2]` leży w stosunku do początku tablicy o tyle elementów dalej:

$$(1*3) + 2$$

Element `X[i][j]` z tablicy o liczbie kolumn 3 leży o:

$$(i*3) + j$$

elementów dalej niż początkowy.

Wniosek:

- do orientacji w tablicy kompilator musi znać liczbę jej kolumn;
- natomiast wcale nie musi używać liczby wierszy.