

VI. Tablice zmiennych

6.1. Po co wprowadzono tablice

Praktycznie zawsze w programach mamy potrzebę zorganizowania naszych danych w jakiś wygodny sposób do późniejszego ich przetwarzania. Tablice dobrze się nadają do takich zastosowań, przyjrzyjmy się im bliżej.

6.2. Tablice jednowymiarowe

Tablica jest to nic innego jak *zwykła* zmienna. Różnica polega na tym, iż przy tworzeniu tablicy zmiennych należy pamiętać o trzech elementach:

→ **typie wartości elementów, które będzie przechowywać tablica,**

→ **nazwę tablicy,**

→ **liczbie elementów(wielkości tablicy).**

```
//Zapis deklaracji tablicy
// TypTablicy NazwaTablicy[LiczbaElementów]
int moja_Tablica[10];
```

Ważnym aspektem jest to, iż **LiczbaElementów** musi być **wartością stałą całkowitą**, lub stałą **const**. Czyli:

```
const STALA = 10;
int Tablica[STALA];
int Tablica_P[10];
int Tablica_A[2.5]; //BŁĄD
```

Inną różnicą jest fakt, że tablica zajmuje więcej miejsca w pamięci.

Odczytywanie i zapisywanie danych do zmiennej z tablicy odbywa się tak samo jak dla zwykłych zmiennych. Jedyną istotną różnicą jest podanie **numeru indeksu**, do którego dane mają zostać zapisane. Tak więc zapis

```
nazwa_tablicy[10] = 33;
//będzie oznaczał przypisanie wartości 33 do jedenastego pola w tablicy.
```

Należy tu wyraźnie podkreślić, że numerowanie tablic w C++ zaczyna się **zawsze od zera !!!**

```
//Przykład użycia tablic zmiennych -----
```

```
#include <iostream>
#include <conio.h>
const short WIELKOSC_TAB = 5;
//-----
int main()
{
    using namespace std;
    //Tworzenie tablic
    int tab1[WIELKOSC_TAB]; //Tablica liczb
    char tab4[2]; //Tablica znaków
```

```
//Przypisywanie wartości
tab1[0] = 1; //pierwszy element tabeli
tab1[1] = 2; //drugi element tabeli
tab1[2] = 3; //trzeci element tabeli
tab1[3] = 4; //czwarty element tabeli
tab1[4] = 5; //piąty element tabeli
```

//NIE DOPUSZCZALNE JEST NASTĘPUJĄCY ZAPIS

tab1[5] = 6; // szósty element tabeli

/* Tabela "tab" zawiera tylko pięć elementów

kompilator tego nie wychwyty !!! */

//Inny sposób przypisywania wartości

char tab2[WIELKOSC_TAB] = {'N', 'u', 'm', 'e', 'r'};

/*Sposobem tym możemy się posłużyć wiedząc co ma się znajdować w tabeli od samego początku działania programu*/

int tab3[4] = {7, 8}; /*nie trzeba wypełniać całej tabeli w tym przypadku tylko pierwszy i drugi element jest wypełniony*/

//możliwości użycia tablic

cout << "Oto dane zawarte w tab1:\n"

<< "pierwszy element - " << tab1[0]

<< "\ndrugi element - " << tab1[1]

<< "\ntrzeci element - " << tab1[2]

<< "\nczwarty element - " << tab1[3]

<< "\npiaty element - " << tab1[4]

<< "\nA tak wygląda nieprawidłowy element szesty - "

<< tab1[5] << endl << endl;

cout << "Należy pamiętać, by nie przekroczyć "

<< tab2[0] << tab2[1] << tab2[2] << tab2[3]

<< tab2[4] << " indeksu tabeli." << endl;

//przypisywanie zmiennych między tablicami

tab4[0] = tab2[0];

tab4[1] = tab2[4];

cout << "Jeżeli tabela ma mieć 5 elementów "

"to jej ostatni element ma " << tab4[0]

<< tab4[1] << " indeksu równy " << tab1[3]

<< "!" << endl << endl;

//Tablica nie wypełniona całkowicie

cout << "tab3 zawiera takie dane:\n"

<< "1- " << tab3[0] << endl

<< "2- " << tab3[1] << endl

<< "3- " << tab3[2] << endl

<< "4- " << tab3[3] << endl;

getch();

return 0;

}

//-----

6.3 Tablice wielowymiarowe

Do tej pory operowaliśmy na tablicach jedno wymiarowych, przyjrzyjmy się jak tworzy i stosuje tablice wielowymiarowe. Założmy, że chcemy napisać program, która będzie przechowywać wyniki Eliminacje MŚ w piłce nożnej *grupy polskiej*.

```
//Przykład tablicy wielowymiarowych -----
#include <iostream>
#include <conio.h>
const short LICZBA_DRUZYN = 6;
const short MAX_ZNAKOW = 12;
//-----
int main()
{
    using namespace std;
    /*Tworzenie i deklaracja tablic dwuwymiarowej przechowującej nazwy państw w grupie */
    char tab1[LICZBA_DRUZYN][MAX_ZNAKOW] = {
        {'S', 'I', 'o', 'w', 'a', 'c', 'j', 'a'},
        {'I', 'r', 'I', 'a', 'n', 'd', 'i', 'a', ' ', 'P'},
        {'C', 'z', 'e', 'c', 'h', 'y'},
        {'P', 'o', 'I', 's', 'k', 'a'},
        {'S', 'I', 'o', 'w', 'e', 'n', 'i', 'a'},
        {'S', 'a', 'n', ' ', 'M', 'a', 'r', '.'},
    };

    /*tablica z pkt tab2 i tablica państw tab1, mają wspólne indeksy dla każdego państwa czyli indeks 4
    (tab1[4], tab2[4]) - oznacza Polskę, a indeks 0 Słowację */
    short tab2[LICZBA_DRUZYN] = {
        9, 7, 7, 7, 7, 0};
    //tabela zawierająca lp kompilator określi wielkość
    short tab3[] = {1, 2, 3, 4, 5, 6};

    //Wyświetlenia danych
    cout << "Tabela eliminacji mistrzostw swiata 2010"
    << endl << "Grupa Polska\n\n"
    << " Lp." << "|" << " Druzyna    " << "|"
    << " pkt. " << endl
    << "-----\n"
    //wyświetlenie I drużyny
    << " " << tab3[0] << " "
    << "|" << tab1[0] << " |" << tab2[0]
    //wyświetlenie II drużyny
    << "\n " << tab3[1] << " "
    << "|" << tab1[1] << " |" << tab2[1]
    //wyświetlenie III drużyny
    << "\n " << tab3[2] << " "
    << "|" << tab1[2] << " |" << tab2[2]
    //wyświetlenie IV drużyny
    << "\n " << tab3[3] << " "
    << "|" << tab1[3] << " |" << tab2[3]
    //wyświetlenie V drużyny
    << "\n " << tab3[4] << " "
    << "|" << tab1[4] << " |" << tab2[4]
    //wyświetlenie VI drużyny
```

```
<< "\n " << tab3[5] << " "
<< "| " << tab1[5] << " | " << tab2[5];
```

```
getch();
return 0;
}
//-----
```

W programie użyto tablic dwuwymiarowych, jednak C++ umożliwia również tworzenie tablic więcej wielowymiarowych. Ich zapis jest bardzo prosty i wygodny w użyciu np.

```
int mapa[10][20][30];
```

utworzy tablicę trójwymiarową, w której będziemy mogli przechowywać liczby typu int. Zapis danych do takiej tablicy jak nie trudno się domyślić będzie wyglądał tak:

```
mapa[9][3][1] = 13;
```

Przeanalizujmy kod programu, skupmy się na tabelach. Pierwszą tabelę **tab1** od razu deklarujemy:

```
/*Tworzenie i deklaracja tablic dwuwymiarowej
przechowującej nazwy państw w grupie */
char tab1[LICZBA_DRUZYN][MAX_ZNAKOW] = {
    {'S', 'I', 'o', 'w', 'a', 'c', 'j', 'a'},
    {'I', 'r', 'l', 'a', 'n', 'd', 'i', 'a', ' ', 'P'},
    {'C', 'z', 'e', 'c', 'h', 'y'},
    {'P', 'o', 'l', 's', 'k', 'a'},
    {'S', 'I', 'o', 'w', 'e', 'n', 'i', 'a'},
    {'S', 'a', 'n', ' ', 'M', 'a', 'r', ' '},
};
```

Po pierwsze tworzymy tablicę o wymiarach [6][12]. Następnie od razu ją wypełniamy o czym świadczy *klamra* ({ }). Jakbyśmy chcieli wypełnić tą tabelę na piechotę podczas działania programu należało by zrobić to tak:

```
char tab1[LICZBA_DRUZYN][MAX_ZNAKOW];
tab1[0][0] = 'S';
tab1[0][1] = 'I';
tab1[0][2] = 'o';
tab1[0][3] = 'w';
tab1[0][4] = 'a';
tab1[0][5] = 'c';
tab1[0][6] = 'j';
tab1[0][7] = 'a';
};
```

Więc podkreślę to jeszcze raz, jeśli znasz dane, które mają znajdować się w tabeli to wprowadź je od razu, zaoszczędzisz czas. Jak to się dzieje, iż kompilator wie w jakie indeks wprowadzić dane?

```
char tab1[LICZBA_DRUZYN][MAX_ZNAKOW] = {
    ----[0] [1] [2] [3] [4] [5] [6] [7]
    [0]{'S', 'I', 'o', 'w', 'a', 'c', 'j', 'a'},

    ----[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
    [1]{'I', 'r', 'l', 'a', 'n', 'd', 'i', 'a', ' ', 'P'},

    ----[0] [1] [2] [3] [4] [5]
```

```
[2]{'C', 'z', 'e', 'c', 'h', 'y'},
```

```
----[0] [1] [2] [3] [4] [5]
```

```
[3]{'P', 'o', 'l', 's', 'k', 'a'},
```

```
----[0] [1] [2] [3] [4] [5] [6] [7]
```

```
[4]{'S', 'l', 'o', 'w', 'e', 'n', 'i', 'a'},
```

```
----[0] [1] [2] [3] [4] [5] [6] [7]
```

```
[5]{'S', 'a', 'n', ' ', 'M', 'a', 'r', ' '},
```

```
};
```

Ponieważ sam numeruje sobie indeksy tak jak jest to pokazane wyżej. A co jeśli usielibyśmy pierwszy wpis(index tabeli[0]) *Słowacja*?

```
char tab1[LICZBA_DRUZYNY][MAX_ZNAKOW] = {
```

```
[0]{'I', 'r', 'l', 'a', 'n', 'd', 'i', 'a', ' ', 'P'},
```

```
[1]{'C', 'z', 'e', 'c', 'h', 'y'},
```

```
[2]{'P', 'o', 'l', 's', 'k', 'a'},
```

```
[3]{'S', 'l', 'o', 'w', 'e', 'n', 'i', 'a'},
```

```
[4]{'S', 'a', 'n', ' ', 'M', 'a', 'r', ' '},
```

```
};
```

Dla kompilatora nie będzie to miało znaczenia, ponieważ co również podkreślę **nie trzeba wypełniać całej tabeli danymi**, a indeksy się tylko przesuną. Jednak w tabeli, która nie jest całkowicie wypełniona mogą znajdować się przypadkowe dane(podobnie jak jest to ze zmiennym gdy ich od razu nie zadeklarujemy).

Przyjrzyjmy się teraz **tab3**, składnia:

```
short tab3[] = {1, 2, 3, 4, 5, 6};
```

informuje kompilator o tym by sam policzył ile jest elementów(sam wstawia indeksy) tabeli. Taki rodzaj tabeli od razu musi zawierać wszystkie dane, w przeciwnym wypadku kompilator nie wie jak duża ma być tabela.

Nie objaśnię zasady wyświetlania danych, ponieważ Sam już powinieneś ją znać i mieć opanowaną z wcześniejszych kursów.

6.4 Tablice i inne zmienne, a zużycie pamięci.

Tworząc tablice należy pamiętać o zasobach pamięci, jakie tablica nam pochłonie. Tablica jednowymiarowa zabierze nam: **sizeof(typ_zmiennej)*ilość_elementów**; bajtów pamięci, a tablica **int mapa[10][20][30]**; pochłonie 4*10*20*30 bajtów pamięci, co daje prawie 24KB zarezerwowanej pamięci.

```
//Przykład zużycie pamięci-----
```

```
#include <iostream>
```

```
#include <conio.h>
```

```
//-----
```

```
int main()
```

```
{
```

```
    using namespace std;
```

```
    int liczba_calkowita;
```

```
    double liczba_zmiennoprzecinkowa;
```

```
    char znak;
```

```
    int tab_jedenwymiar_liczba[10];
```

```

char tab_jedenwymiar_znak[5];

int tab_wielowymiar_liczba[10][20][30];
char tab_wielowymiar_znak[10][20][30];

/*przy wyświetlaniu rozmiaru tablicy połączymy się
operatorem sizeof, który zwraca wielkość
typów(i zmiennych) w bajtach*/
cout << "Pokaz ile zajmują użyte zmienne:\n"
<< " int ma " << sizeof (int) << endl
<< " Liczba całkowita ma " << sizeof liczba_calkowita
<< endl << endl
<< "Natomiast char ma " << sizeof (char)
<< " , a jego zmienna znak ma "
<< sizeof (znak) << endl << endl;

cout << "Natomiast pozostałe rozmiary to:\n"
<< "Tabela jednowymiarowa liczb "
<< sizeof tab_jedenwymiar_liczba << "\n"
<< "Tabela jednowymiarowa znak "
<< sizeof tab_jedenwymiar_znak << "\n"
<< "Tabela wielowymiar liczb "
<< sizeof tab_wielowymiar_liczba << "\n"
<< "Tabela wielowymiar znak "
<< sizeof tab_wielowymiar_znak << "\n"
<< "Oraz rodzynek na sam koniec liczba"
" zmiennoprzecinkowa "
<< sizeof liczba_zmiennoprzecinkowa;
cout << "\n\n Wszystkie wartości podane"
" są w bajtach!!!";

getch();
return 0;
}
//-----

```

Kilka słów o programie, operator **sizeof** zwraca nam wartości typów(zmiennych) w bajtach. Nazwy typów muszą być podane w nawiasach (**int**). Dla nazw zmiennych nie jest to konieczne.

6.5 Ćwiczenia

1. Napisz program, który wykona poniższe zadania:

-> $2 + 7 * 16 - 8$

-> $22 * 2 : 11$

-> $8383 - 222 + 292 * 8$

-> $5 * 2 * 4$

Wyniki zapisz do tabeli, a następnie wyświetl je na ekranie.

2. Używając tabeli z pierwszego zadania napisz program, który wyniki z pierwszego zadania przekaże do drugiej tabeli następnie wykona dodatkowe obliczenia odpowiednio:

-> wynik * 2 - 13

-> wynik - 4 + 1

-> wynik + 88 - 250

-> wynik + (doda wyniki jakie zostaną uzyskane z 3 poprzednich wyrażeń).

Najlepiej napisać program do pierwszego zadania a potem rozszerzyć go by wykonywał podane przykład.

3. Napisz program, który pobierze od użytkownika dwie liczby i wykona na nich działania:

dodawanie, odejmowanie, mnożenie i dzielenie. Wynik ma być wyświetlony w takiej formie:

Użytkownik podał następujące liczby

Liczba 1 = 20

Liczba 2 = 10

Wynik dodawania

$20 + 10 = 30$

Wynik odejmowania

$20 - 10 = 10$

Wynik mnożenia

$20 * 10 = 200$

Wynik dzielenia

$20 : 10 = 2$

Użyj dwóch tablic jedną na wyniki drugą na dwie liczby które poda użytkownik. Następnie spróbuj użyć tylko jednej tablicy.