

Podstawy Programowania

Wykład nr 5: Tablice

dr hab. inż. Dariusz Dereniowski

Katedra Algorytmów i Modelowania Systemów
Wydział ETI, Politechnika Gdańska

Czym jest tablica?

- Typy proste służą do przechowywania pojedynczej wartości. Czasami zachodzi konieczność przechowania wielu wartości tego samego typu. Do tego celu wykorzystujemy tablicę.
- **Tablica** jest sekwencją elementów tego samego typu, które zajmują ciągły obszar w pamięci operacyjnej.
- Tablic używamy więc w sytuacji, gdy mamy do czynienia z sekwencją danych (ciąg liczb, ciąg znaków, itp.)
- Deklaracja tablicy o nazwie **t** zawierającej **X** elementów typu **typ**:

```
typ t[X];
```

- odwołanie się do elementu numer **i** w powyższej tablicy: **t[i]**
- Tablice można także tworzyć dynamicznie (w trakcie działania programu) oraz odwoływać się do poszczególnych elementów “poprzez wskaźnik”. O tych metodach powiemy później.

Odwołania do elementów tablicy

```
int t[5];
```

t[0] t[1] t[2] t[3] t[4]

- Numeracja elementów tablicy rozpoczyna się od **zera**.
- Element **t[5]** nie został zaalokowany w powyższym przykładzie.
- W języku C nie następuje automatyczne sprawdzanie zakresów tablic. To programista jest odpowiedzialny za to, aby kontrolować zakresy indeksów.
- Próba odwołanie się do **t[5]** nie zostanie zasygnalizowana przez kompilator jako błąd.
- Wpisanie wartości do nieistniejącego elementu **t[5]** spowoduje pisanie do obszaru pamięci znajdującego się bezpośrednio za tablicą. Może to spowodować "uszkodzenie" wartości zmiennej, która się tam znajduje.

Inicjalizacja wartości tablicy

- Sposób pierwszy: podajemy rozmiar tablicy i wszystkie jej elementy:

```
int t[5] = {1,3,5,7,9};
```

Skutkuje to: $t[0]=1$, $t[1]=3$, $t[2]=5$, $t[3]=7$, $t[4]=9$.

- Sposób drugi: nie podajemy rozmiaru tablicy, a jedynie wszystkie jej elementy:

```
int t[] = {1,3,5,7,9};
```

Skutkuje to: $t[0]=1$, $t[1]=3$, $t[2]=5$, $t[3]=7$, $t[4]=9$. Kompilator sam przeliczy ile wartości podano w nawiasach klamrowych i na tej podstawie dobiera odpowiedni rozmiar tablicy.

- Sposób trzeci: podajemy rozmiar tablicy i część jej elementów:

```
int t[5] = {1,3,5};
```

Skutkuje to: $t[0]=1$, $t[1]=3$, $t[2]=5$, $t[3]=0$, $t[4]=0$, gdyż brakujące elementy są inicjowane na wartość 0.

(Ciekawostka: w standardzie C99: `int t[]={ [0]=1, [2]=5, [4]=9 }.`)

Dyrektywa #define – wprowadzenie

#define NAZWA WARTOSC

- Dyrektywy są przetwarzane przez preprocesor przed przystąpieniem do kompilacji.
- Przetwarzanie polega na zastąpieniu każdego wystąpienia w kodzie literału NAZWA przez WARTOSC.
- WARTOSC może być stałą lub bardziej skomplikowanym wyrażeniem.
- Preprocesor nie wykonuje analizy składniowej wyrażeń – jedynie dokonuje podstawień. Ewentualne błędy są wskazywane podczas kompilacji.
- Ogólnie przyjęta konwencja: identyfikatory (NAZWA) piszemy dużymi literami.
- Dobrym stylem programisty jest używanie dyrektyw #define do nadawania identyfikatorów stałym używanym w programie, np.:

```
#define LICZBA_MIESIECY 12
```

```
#define KOLOR_TLA 17
```

```
#define ROZMIAR_TABLICY 1000
```

```
#define SZEROKOSC_EKRANU 1200
```

Dzięki temu zmiana wartości (jeśli jest taka potrzeba) następuje tylko w jednym miejscu programu (przy dyrektywie), a nie w każdym jej wystąpieniu w kodzie.

Przykładowy problem

Założmy, że potrzebujemy w programie zdefiniować tablicę, która zawiera liczby o określonej własności. Nazwijmy je *liczbami szczególnymi*. Na razie jest ich pięć, lecz pisząc program mamy na uwadze, że na dalszym etapie być może będziemy musieli rozszerzyć tą tablicę o więcej takich liczb.

```
#include <iostream>
using namespace std;

/* Aby dodać nową liczbę do tablicy i oczekiwać wypisania
wszystkich liczb na ekran należy:
    1) dopisać liczbę do tablicy
    2) zwiększyć rozmiar tablicy o 1
    3) zwiększyć stałą w warunku  $i < 5$ 

Uwaga: rozszerzanie staje się kłopotliwe w przypadku, gdy
miejsce wymagających zmiany 5 na 6 jest dużo więcej.
Taki sposób jest przykładem złego stylu programowania.
*/
int main() {
    int liczby_szczególne[5] = { 7, 13, 16, 21, 14 }, i;

    for ( i=0; i < 5; i++ )
        cout << liczby_szczególne[i] << endl;
    return 0;
}
```

Przykładowy problem c.d.

Poniższy program eliminuje wcześniejszy problem: dodanie elementu w tablicy wymaga **tylko** uaktualnienia wartości stałej **ROZMIAR**.

```
#include <iostream>
using namespace std;

#define ROZMIAR 5
/* Aby dodać nową liczbę do tablicy i oczekiwać wypisania
wszystkich liczb na ekran należy:
    1) dopisać liczbę do tablicy
    2) zmienić deklarację ROZMIAR (z 5 na 6)

Uwaga: Jeśli w kodzie będziemy używać zmiennej rozmiar zamiast
stałej numerycznej 5, to mamy gwarancję, że te dwie
czynności wystarczą przy rozszerzeniu tablicy
Pytanie: czy da się jeszcze wygodniej?
*/
int main() {
    int liczby_szczególne[ROZMIAR] = { 7, 13, 16, 21, 14 }, i;

    for ( i=0; i < ROZMIAR; i++ )
        cout << liczby_szczególne[i] << endl;
    return 0;
}
```

Przykładowy problem c.d.

- W celu dalszego usprawnienia programu z pomocą przychodzi operator `sizeof`.
- Oprócz omawianej dotychczas składni (`sizeof(typ)`) operator można użyć następująco: `sizeof x`, gdzie `x` jest wyrażeniem. Wówczas, wartością `sizeof x` (gdzie `x` jest wyrażeniem) jest liczba bajtów obiektu takiego typu jak `x`.
- Jeśli `x` jest tablicą, to `sizeof x` zwraca liczbę bajtów (nie elementów!) tablicy.
- Jeśli `x` jest tablicą, to `(sizeof x)/(sizeof x[0])` da nam liczbę elementów tablicy `x`.

```
#include <iostream>
using namespace std;

/* Aby dodać nową liczbę do tablicy i oczekiwać wypisania
   wszystkich liczb na ekran należy jedynie...
   dopisać liczbę do tablicy
*/
int main() {
    int liczby_szczególne[] = { 7, 13, 16, 21, 14 }, i;

    for (i=0; i<(sizeof liczby_szczególne)/(sizeof liczby_szczególne[0]); i++)
        cout << liczby_szczególne[i] << endl;
    return 0;
}
```

7
13
16
21
14

Inicjalizacja tablic typu char

- Oprócz wcześniej wymienionych metod inicjalizacji tablic, możemy wykorzystać fakt, że stała znakowa umieszczona w cudzysłowach (") jest traktowana jako tablica znaków, której ostatnim elementem jest znak `'\0'`. Ten ostatni element jest umieszczany dodatkowo za ostatnim znakiem stałej znakowej.
- Poniższe dwie definicje są równoważne:

```
char s[10] = { "napis" };
```

```
char s[10] = { 'n', 'a', 'p', 'i', 's', '\0' };
```

W pierwszym przypadku nawiasy `{ }` można pominąć. W obu przypadkach, pozostałe elementy, począwszy od indeksu numer 6, są inicjalizowane wartością 0:

0	1	2	3	4	5	6	7	8	9
'n'	'a'	'p'	'i'	's'	'\0'	0	0	0	0

- Ile elementów ma tablica: `char s[] = { "Halo!" };`? Odp.: 6 (`s[5]` jest równe `'\0'`).

Łańcuchy znaków

Powody, dla których należy pamiętać o znaku `'\0'` kończącym napis:

- Funkcje biblioteczne zakładają, że `'\0'` jest zakończeniem napisu. Na przykład, wypisywanie napisu na ekran monitora odbywa się poprzez wypisywanie kolejnych znaków, aż do napotkania `'\0'`. W przypadku gdy znak ten nie wystąpi w tablicy, funkcja wypisująca napis na ekran będzie kontynuować swoją pracę, odczytując kolejne bajty w pamięci operacyjnej znajdującej się poza tablicą, i traktując je jako kody ASCII będzie wypisywać znaki na ekran.
- Na podstawie zawartości tablicy (tzn. na podstawie analizy jej kolejnych elementów), wystąpienie znaku `'\0'` jest jedynym sposobem pozwalającym na określenie końca napisu.

Łańcuchy znaków – przykładowy program

```
#include <iostream>
using namespace std;

/* Przykłady manipulacji napisami */
int main() {
    char s[256] = { "Jakis ciekawy napis" };
    int i=0;

    cout << s << endl;
    cout << sizeof s << endl; // zwraca rozmiar tablicy, nie napisu!
    s[13] = '\0';
    cout << s << endl; // pozostałe znaki nadal są w tablicy,
                        // lecz nie są wypisywane

    s[14] = 'p'; // te dwie zmiany w tablicy następują "poza" pierwszym
    s[15] = 'o'; // wystąpieniem znaku '\0' w tablicy s, więc choć
    cout << s << endl; // zmieniają zawartość tablicy, nie wpływają na to,
                        // co się pojawi na ekranie

    s[13] = '\t';
    cout << s << endl;

    cin >> s;
    while ( s[i] != '\0' ) // obliczymy ile znaków ma słowo podane
        i++;              // przez użytkownika
    cout << i << endl;
    return 0;
}
```

```
Jakis ciekawy napis
256
Jakis ciekawy
Jakis ciekawy
Jakis ciekawy   popis
Abrakadabra
11
```

Tablice wielowymiarowe

Przykład deklaracji tablicy dwuwymiarowej o 4 wierszach i 3 kolumnach:

```
char t[4][3];
```

Do elementów tablicy odwołujemy się następująco:

t[0][0]	t[0][1]	t[0][2]
t[1][0]	t[1][1]	t[1][2]
t[2][0]	t[2][1]	t[2][2]
t[3][0]	t[3][1]	t[3][2]

Inicjalizacja tablicy dwuwymiarowej:

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

Tablice dwuwymiarowe – przykład

- Napiszmy program sprawdzający czy podana na wejściu 2-wymiarowa tablica o wymiarach $n \times n$ jest symetryczna. Tablicę nazywamy symetryczną, gdy element znajdujący się w i -tym wierszu i j -tej kolumnie jest taki sam jak element znajdujący się w j -tym wierszu i i -tej kolumnie.
- Na przykład: tablica A jest symetryczna, natomiast tablica B taka nie jest:

$$A = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 4 & 5 & 6 \\ 2 & 5 & 7 & 8 \\ 3 & 6 & 8 & 9 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 4 & 5 & 6 \\ 2 & 5 & 7 & 8 \\ 3 & 7 & 8 & 9 \end{bmatrix}$$

- Program będzie działał na tablicy o ustalonym rozmiarze, np. 3 lub 4, ale aby pozostawić łatwą możliwość zmiany rozmiaru, skorzystamy z dyrektywy `#define`.

Tablice dwuwymiarowe – przykład

```
#include <iostream>
using namespace std;
#define ROZMIAR 2

/* Program sprawdzający czy podana na wejściu tablica jest symetryczna */
int main() {
    int i, j, tablica[ROZMIAR][ROZMIAR], symetria;

    for ( i=0; i < ROZMIAR; i++ )
        for ( j=0; j < ROZMIAR; j++ ) {
            cout << "Podaj wartość elementu X[" << i << "][" << j << "] = ";
            cin >> tablica[i][j];
        }


    for ( symetria=1, i=0; i < ROZMIAR; i++ )
        for ( j=0; j < i; j++ )
            if ( tablica[i][j] != tablica[j][i] )
                symetria = 0;

    cout << "Tablica " << (symetria==1 ? "" : "nie ") << "jest symetryczna\n";
    return 0;
}
```

```
5
Podaj wartość elementu X[0][0] = 0
Podaj wartość elementu X[0][1] = 1
Podaj wartość elementu X[1][0] = 2
Podaj wartość elementu X[1][1] = 3
Tablica nie jest symetryczna
```

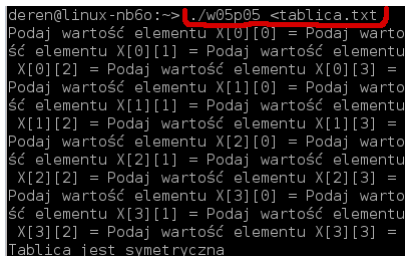
Usprawnienie testowania programów

- Założmy, że chcemy poprzedni program przetestować dla nieco większych tablic, np. rozmiaru 4×4 . (Czyli zmieniamy linijkę `#define ROZMIAR 4`.)
- Wielokrotne uruchamianie programu w procesie testowania staje się procesem żmudnym i mało efektywnym, gdy za każdym razem jesteśmy zmuszeni do wpisywania 16 elementów tablicy.
- Z pomocą przyjdzie nam mechanizm przekierowania wejścia w konsoli.
- Tworzymy plik tekstowy, np. o nazwie `tablica.txt`, o treści jak wyżej (każda jego linia kończy się znakiem enter).



0	1	2	3
1	4	5	6
2	5	7	8
3	6	8	9

- W linii poleceń korzystamy z przekierowania wejścia. Środowisko uruchomieniowe programu (konsola) dba o to, aby program nie dostrzegł żadnej różnicy w sytuacjach, gdy dane wejściowe pochodzą z klawiatury, czy są przekierowane z pliku.
- Efekty uboczne! (patrz obok, gdzie `w05p05` jest nazwą pliku wykonywalnego).
- Aby “oddzielić” wejścia programu od wyjścia: `./w05p05 <tablica.txt >wyjscie.txt` (To, co wypisuje program zostanie umieszczone w pliku `wyjscie.txt`.)



```
dereh@linux-nb6o:~$ ./w05p05 <tablica.txt
Podaj wartość elementu X[0][0] = Podaj wartość
Podaj wartość elementu X[0][1] = Podaj wartość elementu
Podaj wartość elementu X[0][2] = Podaj wartość elementu
Podaj wartość elementu X[0][3] =
Podaj wartość elementu X[1][0] = Podaj wartość elementu
Podaj wartość elementu X[1][1] = Podaj wartość elementu
Podaj wartość elementu X[1][2] = Podaj wartość elementu
Podaj wartość elementu X[1][3] =
Podaj wartość elementu X[2][0] = Podaj wartość
Podaj wartość elementu X[2][1] = Podaj wartość elementu
Podaj wartość elementu X[2][2] = Podaj wartość elementu
Podaj wartość elementu X[2][3] =
Podaj wartość elementu X[3][0] = Podaj wartość
Podaj wartość elementu X[3][1] = Podaj wartość elementu
Podaj wartość elementu X[3][2] = Podaj wartość elementu
Podaj wartość elementu X[3][3] =
Tablica jest symetryczna
```