

# Programowanie I

## Wykład 4

dr inż. Rafał Brociek

Wydział Matematyki Stosowanej  
Politechnika Śląska



25.10.2021

# Tablice

Tablica - ciąg obiektów tego samego typu, zajmuje ciągły obszar w pamięci.

## Schemat

```
typ nazwaTablicy[liczba_elementow];  
typ nazwaTablicy[ ] = {wart_1, wart_2, ..., wart_n};  
typ nazwaTablicy[liczba_elementow] = {wart_1, wart_2, ..., wart_n};
```

## Przykład

```
double tab[10]; // 10 przypadkowych elementów typu double  
char znaki[ ] = {'a', 'b', 'c'}; // 3 elementy typu char  
int liczby[5] = {1, 2, 3}; // 5 elementów typu int, dopełnione zerami {1,2,3,0,0}  
int numerki[ 5 ]{}; // 5 elementów typu int (inicjalizowanie zerami) {0,0,0,0,0}
```

## Tablice - informacje podstawowe

- Indeksowanie (numeracja) elementów tablicy zaczyna się od 0, kończy na  $N - 1$  (tablica  $N$ -elementowa).
- Do  $i$ -tego elementu tablicy odwołujemy się po przez nazwa[i].
- Kompilator nie czuwa, czy przekroczyliśmy zakres wielkości tablicy  

```
int tab[5] = {2,4,6,8,10};  
tab[2] + tab[8]; // błąd
```
- Nazwa tablicy jest równocześnie adresem jej zerowego elementu.

# Adres tablicy w pamięci

Dzięki operatorowi & jesteśmy w stanie uzyskać informację o adresie zmiennej w pamięci.

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int x = 5;
7     int tab[10] = { 1,2,3,4 };
8     cout << "adres x: " << (int)&x << endl;
9     cout << "adres tab[0]: " << (int)tab << endl;
10    cout << "adres tab[0] (inaczej): " << (int)&tab[0] << endl;
11    cout << "adres tab[1]: " << (int)&tab[1] << endl;
12    cout << "adres tab[2]: " << (int)&tab[2] << endl;
13    system("pause");
14 }
```

# Przekazywanie tablicy do funkcji

Tablice przesyła się do funkcji po przez podanie adresu zerowego elementu tablicy (nazwa tablicy jest właśnie adresem jej zerowego elementu).

```
1 // deklaracje funkcji
2 int oblicz(int tab[]); // argument to tablica
3 bool czyKontynuowac(int licz); // argument to int
4
5 // definicja tablicy i wywołanie funkcji
6 int liczby[5] = {1, 2, 5, 7, 9};
7 oblicz(liczby); // przekazanie adresu zerowego elementu
8 oblicz(&liczby[0]); // przekazanie adresu zerowego elementu
9
10 czyKontynuowac(liczby[2]); // przekazanie przez wartość
```

Ampersand & to jednoargumentowy operator uzyskania adresu danego obiektu (numeru komórki pamięci oraz informacji o typie obiektu).

# Przekazywanie tablicy do funkcji

```
1 #include <iostream>
2 using namespace std;
3
4 void powieksz(int tab[], int rozmiar);
5 void wypisz(int tab[], int rozmiar);
6
7 int main() {
8     int liczby[5] = { 1, 2, 5, 7, 9 };
9     wypisz(liczby, 5);
10    powieksz(liczby, 5);
11    wypisz(liczby, 5);
12 }
13
14 void powieksz(int tab[], int rozmiar) {
15     for (int i = 0; i < rozmiar; i++)
16         tab[i] *= 100;
17 }
18
19 void wypisz(int tab[], int rozmiar){
20     for (int i = 0; i < rozmiar; i++)
21         cout << "tab[ " << i << " ] = " << tab[i] << endl;
22 }
```

# Tablice znakowe

```
1 // tablica 10 zmiennych typu char
2 char znaki[10];
3
4 char napis[] = "Program";
5 // po ostatnim znaku zostaje wstawiony znak '\0'
6 // o wartości liczbowej 0
```

'P'	'r'	'o'	'g'	'r'	'a'	'm'	'\0'
-----	-----	-----	-----	-----	-----	-----	------

80	114	111	103	114	97	109	0
----	-----	-----	-----	-----	----	-----	---

# C-string

Ciąg liter zakończony znakiem *null* (koniec ciągu - znak 0) nazywamy **C-stringiem**.

```
1 char rzecz[8] = {"karta"}; // C-string
2 // C-string
3 char imie[8] = {'j', 'a', 'n', 'u', 's', 'z'};
4 char zwierze[] = {"kot"}; // C-string
5 // można też char zwierze[] = "kot";
6 // tablica znaków, ale nie C-string
7 char litery[] = {'a', 'b', 'c'};
```

'k'	'a'	'r'	't'	'a'	'\0'	'\0'	'\0'
-----	-----	-----	-----	-----	------	------	------

'j'	'a'	'n'	'u'	's'	'z'	'\0'	'\0'
-----	-----	-----	-----	-----	-----	------	------

'k'	'o'	't'	'\0'
-----	-----	-----	------

'a'	'b'	'c'
-----	-----	-----



# C-string

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     char imie[] = { "rysiek" };
7
8     cout << "imie: " << imie;
9     cout << ", rozmiar: " << sizeof(imie) << endl;
10    cout << "kody znakow: ";
11    for (int i = 0; i < sizeof(imie); i++)
12    {
13        cout << static_cast<int>(imie[i]) << " ";
14    }
15    cout << endl;
16 }
```

# Wczytanie C-stringa z klawiatury

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     char nazwaMiasta[30];
7     cout << "Podaj nazwe miasta: ";
8
9     //wczytanie napisu z klawiatury
10    cin >> nazwaMiasta; // lub użyć funkcji get?
11
12    //wypisanie nazwy na ekran
13    cout << nazwaMiasta << endl;
14
15    //wypisanie po jednym znaku
16    int i = 0;
17    while (nazwaMiasta[i] != 0)
18        cout << nazwaMiasta[i++] << endl;
19    cout << endl;
20 }
```

# Wczytanie C-stringa z klawiatury

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     char nazwaMiasta[30];
7     char nazwaKraju[30];
8
9     cout << "Podaj nazwe miasta: ";
10    //wczytanie napisu z klawiatury
11    cin.get(nazwaMiasta, 30);
12
13    //cin.get();
14    cout << "Podaj nazwe kraju: ";
15    //wczytanie napisu z klawiatury
16    cin.get(nazwaKraju, 30);
17
18    //wypisanie nazwy na ekran
19    cout << "_____" << endl;
20    cout << nazwaMiasta << endl;
21    cout << nazwaKraju << endl;
22 }
```

# `cin.get()`, `cin.getline()`

Funkcja `cin.getline()` odczytuje znaki z klawiatury do naciśnięcia klawisza ENTER. Znak nowej linii (przesyłany klawiszem ENTER) jest odrzucany.

Funkcja `cin.get()` odczytuje znaki z klawiatury do naciśnięcia klawisza ENTER. Znak nowej linii (przesyłany klawiszem ENTER) jest przechowywany w buforze.

# cin.get(), cin.getline()

```
1 #include<iostream>
2 using namespace std;
3
4 int main() {
5     char imie[30];
6     char nazwisko[30];
7     char znak{};
8     cout << "Podaj imie: ";
9     cin.get(imie, 30); //cin.getline(imie, 30);
10    //cin.get(znak);
11    //cout << "znak: " << static_cast<int>(znak) << endl;
12    cout << "Podaj nazwisko: ";
13    cin.get(nazwisko, 30);
14    cout << endl;
15    cout << "Imie: " << imie << endl;
16    cout << "Nazwisko: " << nazwisko << endl;
17
18    system("pause");
19 }
```

# C-string

```
1 char miasto[30];
2 miasto = "Gliwice"; // ŹLE
3 char imie1[20] = { "Karolina" };
4 char imie2[20] = { "Agata" };
5
6 if (imie1 == imie2) // ŹLE
7     cout << "Te same imiona" << endl;
8
9 bool czyRowne(char [], char []);
10 ...
11 bool czyRowne(char w1[], char w2[])
12 {
13     int i = 0;
14     while (w1[i] != 0 && w2[i] != 0)
15         if (w1[i] != w2[i++]) return false;
16     if (w1[i] == 0 && w2[i] == 0) return true;
17     return false;
18 }
```

# Biblioteka cstring

W bibliotece cstring zdefiniowano przydatne funkcje (np. porównanie, kopiowanie) operujące na tablicach znaków.

```
1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 int main()
6 {
7     char imie1[] = "Karolina";
8     char imie2[] = "Karolina";
9
10    cout << "Wynik porownania: ";
11    cout << strcmp(imie1, imie2) << endl;
12
13    if (strcmp(imie1, imie2) == 0)
14        cout << "te same imiona" << endl;
15
16    system("pause");
17 }
```

# Rozmiar tablicy

Rozmiar tablicy musi być stałą dosłowną lub stałą typu `const` lub `constexpr` znaną na etapie kompilacji.

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     const int rozmiar_1 = 100;
7     int rozmiar_2 = 50;
8     const int rozmiar_3 = rozmiar_2;
9     constexpr int rozmiar_4 = 30;
10
11     int liczby[rozmiar_1]; // Ok
12     //int liczby1[rozmiar_2]; // Źle
13     //int liczby2[rozmiar_3]; // Źle
14     int liczby3[rozmiar_4]; // Ok
15
16     system("pause");
17 }
```



# Stałe const/constexpr

Stałym const, constexpr muszą być nadane wartości w trakcie tworzenia (inicjalizacja). W przypadku constexpr wartość stałej musi być znana na etapie kompilacji (inaczej niż przy stałej const).

```
1 int liczba{};
2 cin >> liczba;
3 const int stala = liczba;
4 const int stala_2 = 145;
5 const int stala_3; // błąd
6 stala_2 = 14; // błąd
```

```
1 int liczba{};
2 cin >> liczba;
3 constexpr int stala = liczba; // błąd
4 constexpr int E = 2.718281828459;
5 constexpr int E2 = E*E;
```

# Tablice wielowymiarowe

Rozmiary tablic muszą być stałymi znanymi już w czasie kompilacji.  
 $n$ ,  $m$  - stałe dosłowne lub zmienne typu `const`, `constexpr`.

## Schemat

```
typ nazwaTablicy[n][m];  
typ nazwaTablicy[n][m] = wartość;
```

## Przykład

```
int liczby[2][3]={{1,2,3},{1,1,1}};  
double dane[3][2]={{1.0,2.0}};  
double macierz[2][3]={{1.0},{5.0}};  
int cyferki[2][3]={1,2,3,4,5,6};
```

# Tablice wielowymiarowe

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     const int n = 9, m = 9;
7     int tabliczka[n][m];
8     for (int i = 0; i < n; i++)
9     {
10         for (int j = 0; j < m; j++)
11         {
12             tabliczka[i][j] = (i <= j ? (i+1) * (j+1) : 0);
13             cout.width(3);
14             cout << tabliczka[i][j] << ' ';
15         }
16         cout << endl;
17     }
18 }
```

# Tablice wielowymiarowe

Jak komputer oblicza, gdzie w pamięci jest dany element tablicy?

```
double dane[5][10];
```

Element o indeksach **i,j** (`dane[i][j]`) jest oddalony od początku tablicy o  $(i * 10) + j$  elementów.

```
1 void fun(int tab[]); // OK
2 void fun(int tab[10]); // OK
3 void fun(int tab[][]); // ŹŁE
4 void fun(int tab[][10]); // OK
5 void fun(int tab[5][10]); // OK
6 void fun(int tab[][10][3]); // OK
```

Adres elementu o indeksach **i, j**:

$\text{adres dane}[i][j] = p + r * (N * i + j)$

**p** – adres początku tablicy (czyli `&dane[0][0]`),

**r** – rozmiar pojedynczego elementu (tutaj `double`),

**N** – liczba kolumn tablicy (drugi z rozmiarów tablicy).

# Tablice wielowymiarowe

Indeksy kolumn

0 1 2

Indeksy wierszy

0 1 2 3

t[4][3]

	0	1	2
0	12	7	5
1	34	21	8
2	1	3	14
3	66	44	37

12	7	5	34	21	8	1	3	14	66	44	37
----	---	---	----	----	---	---	---	----	----	----	----

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]

- Tablica przechowuje obiekty tego samego typu oraz zajmuje ciągły obszar w pamięci.
- Indeksowanie elementów zaczyna się od zera.
- Do elementów tablicy odwołujemy się przez nazwę tablicy, nawiasy kwadratowe i numer indeksu (`tab[i]`).
- Nazwa tablicy jest adresem początku tablicy (elementu o indeksie 0).
- Do funkcji tablicę przekazujemy poprzez podanie jej nazwy (adresu początku tablicy).
- Tablicę znaków zakończoną znakiem null nazywamy C-stringiem.

- W celu pobrania ze standardowego wejścia (klawiatury) napisu (c-string) i zapisania go w tablicy znaków możemy posłużyć się funkcjami `get()`, `getline()`.
- Rozmiar tablicy musi być stałą dosłowną, stałą typu `const` lub `constexpr`.
- Stałym `const` oraz `constexpr` nadajemy wartości w trakcie ich tworzenia. Późniejsza zmiana tych wartości jest niemożliwa.
- Przy obliczaniu adresu w pamięci, w którym „znajduje” się element tablicy wielowymiarowej o zadanych indeksach, nie jest brany pod uwagę pierwszy wymiar tablicy („zewnętrzny”).

**Zadanie 1.** Napisać funkcję:

```
double srednia(double dane[], int rozmiar),
```

która oblicza i zwraca średnią arytmetyczną liczb z tablicy dane.

**Zadanie 2.** Napisać program, który będzie wczytywał z klawiatury słowa do tablicy znakowej, aż do momentu, gdy napotka słowo *koniec*.



Dziękuję za uwagę