# Tablice (jedno i wielowymiarowe), łańcuchy znaków

wer. 8 z drobnymi modyfikacjami!

Wojciech Myszka

Katedra Mechaniki, Inżynierii Materiałowej i Biomedycznej

2024-03-22 13:50:06 +0100

# Zmienne

## Przypomnienie/podsumowanie

1. Wszystkie zmienne muszą być zadeklarowane.

# Zmienne

## Przypomnienie/podsumowanie

1. Wszystkie zmienne muszą być zadeklarowane.
2. Nazwa zmiennej składa się z liter i cyfr, a rozpoczyna się literą; znak podkreślenia zalicza się do liter.

# Zmienne

## Przypomnienie/podsumowanie

1. Wszystkie zmienne muszą być zadeklarowane.
2. Nazwa zmiennej składa się z liter i cyfr, a rozpoczyna się literą; znak podkreślenia zalicza się do liter.
3. Nazwy zmiennych nie powinny się zaczynać od znaku podkreślenia (tak nazywają się zmienne systemowe).

# Zmienne

## Przypomnienie/podsumowanie

1. Wszystkie zmienne muszą być zadeklarowane.
2. Nazwa zmiennej składa się z liter i cyfr, a rozpoczyna się literą; znak podkreślenia zalicza się do liter.
3. Nazwy zmiennych nie powinny się zaczynać od znaku podkreślenia (tak nazywają się zmienne systemowe).
4. Deklaracja obowiązuje wewnątrz bloku (i we wszystkich blokach znajdujących się „niżej”).

# Zmienne

## Przypomnienie/podsumowanie

1. Wszystkie zmienne muszą być zadeklarowane.
2. Nazwa zmiennej składa się z liter i cyfr, a rozpoczyna się literą; znak podkreślenia zalicza się do liter.
3. Nazwy zmiennych nie powinny się zaczynać od znaku podkreślenia (tak nazywają się zmienne systemowe).
4. Deklaracja obowiązuje wewnątrz bloku (i we wszystkich blokach znajdujących się „niżej").
5. W C występują zmienne globalne (zewnętrzne) i lokalne.

# Zmienne

## Przypomnienie/podsumowanie

1. Wszystkie zmienne muszą być zadeklarowane.
2. Nazwa zmiennej składa się z liter i cyfr, a rozpoczyna się literą; znak podkreślenia zalicza się do liter.
3. Nazwy zmiennych nie powinny się zaczynać od znaku podkreślenia (tak nazywają się zmienne systemowe).
4. Deklaracja obowiązuje wewnątrz bloku (i we wszystkich blokach znajdujących się „niżej").
5. W C występują zmienne globalne (zewnętrzne) i lokalne.
6. Deklaracja lokalna przysłania deklarację globalną (jeżeli nawa zmiennej jest taka sama).

# W szczególności...

...poniższa konstrukcja

```
for(int i = 0; i < 10; i++)
{
    printf("i = %d\n), i);
}
```

jest poprawna.

# W szczególności...

...poniższa konstrukcja

```
for(int i = 0; i < 10; i++)
{
    printf("i = %d\n), i);
}
```

jest poprawna.

Ale poniższa

```
for(int i = 0; i < 10; i++)
{
    printf("i = %d\n), i);
}
printf("i = %d\n), i);
```

już nie...

...gdyz zmienna i jest zmienną lokalną tylko dla pętli!

# Zmienne zewnętrzne i wewnętrzne

```
#include <stdio.h>
int a; // <—— Zmienna zewnetrzna
int main(void)
{
      int b; // <—— Zmienna wewnetrzna
      ...
```

Zmienne zewnętrzne nazywane bywają zmiennymi „globalnymi” (czyli dostępnymi dla każdej funkcji programu).

# Zmienne statyczne i automatyczne

## Trochę zamętu

1. Dodatkowo można zażądać od zmiennej żeby była „statyczna” (co deklaruje się dodając słowo kluczowe **static** przed nazwą typu).

```
static int x;
```

2. Zmienna statyczna zewnętrzna pozostaje zdefiniowana **tylko** dla funkcji zdefiniowanych w jednym pliku źródłowym (ukryta jest dla funkcji z innych plików źródłowych).
3. Zmienna statyczna wewnętrzna zachowuje swoją wartość pomiędzy kolejnymi wywołaniami funkcji.
4. Zmienne, które nie są statyczne **nie muszą** zachowywać wartości między wejściami do funkcji (ale mogą) — ale nie można na to liczyć!
5. Dodatkowo zmienne statyczne wewnętrzne inicjowane są na wartość zero (jeżeli programista nie zażąda żeby było inaczej).

# Zmienne statyczne i automatyczne

```c
#include <stdio.h>
void f(void)
{
    static int x ; /* zmienna statyczna */
    int y = 0;     /* zmienna automatyczna */
    x++;
    y++;
    printf("X=%d, Y=%d\n", x, y);
}
int main()
{
    f();
    f();
    f();
    return 0;
}
```

# Tablice

1. Gdy potrzebujemy przechować kilka zmiennych tego samego typu (i jakoś powiązanych ze sobą) stosujemy **tablicę**.

# Tablice

1. Gdy potrzebujemy przechować kilka zmiennych tego samego typu (i jakoś powiązanych ze sobą) stosujemy **tablicę**.
2. Tablica to ciąg zmiennych o tej samej nazwie; dostęp do poszczególnych elementów odbywa się przez podanie numeru zmiennej (indeksu/ów).

# Tablice

1. Gdy potrzebujemy przechować kilka zmiennych tego samego typu (i jakoś powiązanych ze sobą) stosujemy **tablicę**.
2. Tablica to ciąg zmiennych o tej samej nazwie; dostęp do poszczególnych elementów odbywa się przez podanie numeru zmiennej (indeksu/ów).


3. Elementy numerowane są **począwszy od zera**.

# Tablice

1. Gdy potrzebujemy przechować kilka zmiennych tego samego typu (i jakoś powiązanych ze sobą) stosujemy **tablicę**.
2. Tablica to ciąg zmiennych o tej samej nazwie; dostęp do poszczególnych elementów odbywa się przez podanie numeru zmiennej (indeksu/ów).


3. Elementy numerowane są **począwszy od zera**.
4. Deklaracja wygląda tak:
   **typ** nazwa_tablicy[rozmiar];

# Tablice

1. Tablica jest zmienną złożoną (strukturą pewnego rodzaju).

# Tablice

1. Tablica jest zmienną złożoną (strukturą pewnego rodzaju).
2. Słuzy do przechowywania danych tego samego typu.

# Tablice

1. Tablica jest zmienną złożoną (strukturą pewnego rodzaju).
2. Słuzy do przechowywania danych tego samego typu.
3. Jeżeli chcemy nadać elementom tablicy wartości początkowe
   **int** tablica[3] = {1,2,3};

# Tablice

1. Tablica jest zmienną złożoną (strukturą pewnego rodzaju).
2. Słuzy do przechowywania danych tego samego typu.
3. Jeżeli chcemy nadać elementom tablicy wartości początkowe
   **int** tablica[3] = {1,2,3};
4. To jest również poprawna deklaracja:
   **int** tablica[20] = {1,};
   (pierwszy element tablicy ma wartość 1, pozostałe mają wartość 0)

# Tablice

1. Tablica jest zmienną złożoną (strukturą pewnego rodzaju).
2. Słuzy do przechowywania danych tego samego typu.
3. Jeżeli chcemy nadać elementom tablicy wartości początkowe
   **int** tablica[3] = {1,2,3};
4. To jest również poprawna deklaracja:
   **int** tablica[20] = {1,};
   (pierwszy element tablicy ma wartość 1, pozostałe mają wartość 0)
5. Nie zawsze trzeba podawać rozmiar tablicy — czasami kompilator może się domyślić sam:
   **int** tablica[] = {1, 2, 3, 4, 5};
   zostanie zadeklarowana tablica o pięciu elementach.

# Wielkość tablic

```
1 #include<stdio.h>
2 int main(void)
3 {
4     int t[] = {1, 2, 3, 4, };
5     int i;
6     for (i = -1; i < 7; i++)
7         printf("t[%d] = %d\n", i, t[i]);
8     return 0;
9 }
```

# Wielkość tablic

```
1 #include<stdio.h>
2 int main(void)
3 {
4     int t[] = {1, 2, 3, 4, };
5     int i;
6     for (i = -1; i < 7; i++)
7         printf("t[%d] = %d\n", i, t[i]);
8     return 0;
9 }
```

Ile elementów ma tablica t?

# Wykonany po raz pierwszy

```
t[-1] = 11131
t[0] = 1
t[1] = 2
t[2] = 3
t[3] = 4
t[4] = -1296194160
t[5] = 32767
t[6] = 0
```

# Wykonany po raz drugi

```
t[-1] = 10955
t[0] = 1
t[1] = 2
t[2] = 3
t[3] = 4
t[4] = -868000288
t[5] = 32767
t[6] = 0
```

# Wykonany po raz trzeci

```
t[-1] = 11015
t[0] = 1
t[1] = 2
t[2] = 3
t[3] = 4
t[4] = -143761264
t[5] = 32767
t[6] = 0
```

# Inicjowanie I

1. W deklaracji obiektu można zawrzeć wartość początkową deklarowanego identyfikatora.
2. Inicjator, który poprzedza się operatorem = jest albo wyrażeniem, albo listą inicjatorów zawartą w nawiasach klamrowych.
3. Lista może kończyć się przecinkiem.
4. Dla obiektów i tablic statycznych wszystkie wyrażenia w inicjatorach muszą być wyrażeniami stałymi.
5. Nie inicjowany jawnie obiekt statyczny jest inicjowany tak, jakby jemu, (lub jego składowym) przypisano wartość zero.
6. Początkowa wartość nie zainicjowanego jawnie obiektu automatycznego jest niezdefiniowana.
7. Inicjatorem dla obiektu arytmetycznego jest pojedyncze wyrażenie (być może ujęte w nawiasy klamrowe).

# Inicjowanie II

8. Inicjatorem dla struktury jest albo wyrażenie tego samego typu albo ujęta w nawiasy klamrowe lista inicjatorów dla jej kolejnych składowych.
9. Inicjatorem dla tablicy jest ujęta w klamry lista inicjatorów dla jej kolejnych elementów.
10. Jeśli nie jest znany rozmiar tablicy — to rozmiar ten wylicza się na podstawie liczby inicjatorów.
11. Jeśli tablica ma ustalony rozmiar — liczba inicjatorów nie może przekroczyć liczby elementów tablicy; jeśli lista jest krótsza uzupełniana jest zerami.
12. Specjalnym przypadkiem jest tablica znakowa, która może być inicjowana napisem (kolejne znaki napisu inicjują kolejne elementy tablicy).

# Inicjowanie III

13. Jeżeli nie jest znany rozmiar tablicy znakowej jest on wyliczany na podstawie liczby znaków w napisie (włączając w to końcowy znak zerowy).

# Tablice wielowymiarowe

## Przykład

```c
#include<stdio.h>
int main(void)
{
    int a[4][3] = {
        {1, 3, 5},
        {2, 4, 6},
        {3, 5, 7},
    };
    int i, j;
    for (i = 0; i < 4; i++){
        for (j = 0; j < 3; j++)
        printf(" %d |", a[i][j]);
        printf("\n");
    }
    return 0;
}
```

# Tablice wielowymiarowe

## Wynik działania programu

```
1 | 3 | 5 |
2 | 4 | 6 |
3 | 5 | 7 |
0 | 0 | 0 |
```

# Tablice wielowymiarowe

## Zadanie domowe

Zmodyfikować tak przykładowy progam, żeby drukował wyniki w następującej postaci:

```
| 1 | 3 | 5 |
| 2 | 4 | 6 |
| 3 | 5 | 7 |
| 0 | 0 | 0 |
```

# Tablice wielowymiarowe

## Inicjowanie — warianty

```
int a[4][3] = {
    1, 3, 5, 2, 4, 6, 3, 5, 7
};
```

# Tablice wielowymiarowe

## Inicjowanie — warianty

```
int a[4][3] = {
    1, 3, 5, 2, 4, 6, 3, 5, 7
};
```

Niestety, powyższe nie do końca jest poprawne: pojawią się ostrzeżenia podczas kompilacji!

# Tablice wielowymiarowe

## Inicjowanie — warianty

```
int a[4][3] = {
    1, 3, 5, 2, 4, 6, 3, 5, 7
};
```

Niestety, powyższe nie do końca jest poprawne: pojawią się ostrzeżenia podczas kompilacji!

```
1 | 3 | 5 |
2 | 4 | 6 |
3 | 5 | 7 |
0 | 0 | 0 |
```

# Tablice wielowymiarowe

## Inicjowanie — warianty

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

# Tablice wielowymiarowe

## Inicjowanie — warianty

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

```
1 | 0 | 0 |
2 | 0 | 0 |
3 | 0 | 0 |
4 | 0 | 0 |
```

# Napisy

1. Stała znakowa (złożona z jednego znaku) zapisywana jest tak `'c'` („c” to dowolny znak lub specjalna stała złożona ze znaku „backslash” (\) i specjalnego symbolu.

# Napisy

1. Stała znakowa (złożona z jednego znaku) zapisywana jest tak `'c'` („c” to dowolny znak lub specjalna stała złożona ze znaku „backslash” (\) i specjalnego symbolu.
2. Stała tekstowa zapisywana jest w cudzysłowach (podwójne apostrofy) `"Ala ma kota"`

# Napisy

1. Stała znakowa (złożona z jednego znaku) zapisywana jest tak `'c'` („c” to dowolny znak lub specjalna stała złożona ze znaku „backslash” (\) i specjalnego symbolu.
2. Stała tekstowa zapisywana jest w cudzysłowach (podwójne apostrofy) `"Ala ma kota"`
3. Sąsiadujące ze sobą napisy łączone są w jeden napis (`"Ala" "ma" "kota"` tworzy napis "Alamakota"; `"Ala " "ma" " kota"` tworzy "Ala ma kota").

# Napisy

1. Stała znakowa (złożona z jednego znaku) zapisywana jest tak `'c'` („c” to dowolny znak lub specjalna stała złożona ze znaku „backslash” (\) i specjalnego symbolu.
2. Stała tekstowa zapisywana jest w cudzysłowach (podwójne apostrofy) `"Ala ma kota"`
3. Sąsiadujące ze sobą napisy łączone są w jeden napis ("Ala" "ma" "kota" tworzy napis "Alamakota"; "Ala " "ma" " kota" tworzy "Ala ma kota").
4. Na końcu napisu umieszczany jest znak o kodzie ASCII równym zero ('\x00') pozwalający rozpoznać koniec tekstu.

# Napisy

1. Stała znakowa (złożona z jednego znaku) zapisywana jest tak `'c'` („c” to dowolny znak lub specjalna stała złożona ze znaku „backslash” (\) i specjalnego symbolu.
2. Stała tekstowa zapisywana jest w cudzysłowach (podwójne apostrofy) `"Ala ma kota"`
3. Sąsiadujące ze sobą napisy łączone są w jeden napis ("Ala" "ma" "kota" tworzy napis "Alamakota"; "Ala " "ma" " kota" tworzy "Ala ma kota").
4. Na końcu napisu umieszczany jest znak o kodzie ASCII równym zero ('\x00') pozwalający rozpoznać koniec tekstu.
5. W napisach można używać wszystkich symboli specjalnych dostępnych w stałych znakowych.

# Napisy

1. Stała znakowa (złożona z jednego znaku) zapisywana jest tak `'c'` („c” to dowolny znak lub specjalna stała złożona ze znaku „backslash” (\) i specjalnego symbolu.
2. Stała tekstowa zapisywana jest w cudzysłowach (podwójne apostrofy) `"Ala ma kota"`
3. Sąsiadujące ze sobą napisy łączone są w jeden napis ("Ala" "ma" "kota" tworzy napis "Alamakota"; "Ala " "ma" " kota" tworzy "Ala ma kota").
4. Na końcu napisu umieszczany jest znak o kodzie ASCII równym zero ('\x00') pozwalający rozpoznać koniec tekstu.
5. W napisach można używać wszystkich symboli specjalnych dostępnych w stałych znakowych.
6. Typem do przechowywania znaków jest **char**.

# Napisy

1. Stała znakowa (złożona z jednego znaku) zapisywana jest tak `'c'` („c” to dowolny znak lub specjalna stała złożona ze znaku „backslash” (\) i specjalnego symbolu.
2. Stała tekstowa zapisywana jest w cudzysłowach (podwójne apostrofy) `"Ala ma kota"`
3. Sąsiadujące ze sobą napisy łączone są w jeden napis ("Ala" "ma" "kota" tworzy napis "Alamakota"; "Ala " "ma" " kota" tworzy "Ala ma kota").
4. Na końcu napisu umieszczany jest znak o kodzie ASCII równym zero ('\x00') pozwalający rozpoznać koniec tekstu.
5. W napisach można używać wszystkich symboli specjalnych dostępnych w stałych znakowych.
6. Typem do przechowywania znaków jest **char**.
7. Napisy trzeba przechowywać w tablicach typu **char**.

# Napisy

1. Stała znakowa (złożona z jednego znaku) zapisywana jest tak `'c'` („c” to dowolny znak lub specjalna stała złożona ze znaku „backslash” (\) i specjalnego symbolu.
2. Stała tekstowa zapisywana jest w cudzysłowach (podwójne apostrofy) `"Ala ma kota"`
3. Sąsiadujące ze sobą napisy łączone są w jeden napis ("Ala" "ma" "kota" tworzy napis "Alamakota"; "Ala " "ma" " kota" tworzy "Ala ma kota").
4. Na końcu napisu umieszczany jest znak o kodzie ASCII równym zero ('\x00') pozwalający rozpoznać koniec tekstu.
5. W napisach można używać wszystkich symboli specjalnych dostępnych w stałych znakowych.
6. Typem do przechowywania znaków jest **char**.
7. Napisy trzeba przechowywać w tablicach typu **char**.
8. Polskie znaki — na razie proponuję o tym zapomnieć!

# Tablice znakowe

To jest poprawna deklaracja. Tablica będzie miała rozmiar 14 (13 znaków napisu i znak null kończący napis). Można to sprawdzić za pomocą funkcji `sizeof(tekscik)`.

```
char tekscik[] = "Ala ma kotala";
```

# Tablice znakowe

To jest poprawna deklaracja. Tablica będzie miała rozmiar 14 (13 znaków napisu i znak null kończący napis). Można to sprawdzić za pomocą funkcji `sizeof(tekscik)`.

```
char tekscik[] = "Ala ma kotala";
```

Poniżej również poprawna deklaracja tablicy (o łącznym rozmiarze 21 znaków). Liczba wierszy wyliczana jest automatycznie podczas kompilacji.

```
char teksty[][7] = {
    {"Ala"},
    {"ma"},
    {"kotala"}
};
```

# Tablice znakowe

To jest poprawna deklaracja. Tablica będzie miała rozmiar 14 (13 znaków napisu i znak null kończący napis). Można to sprawdzić za pomocą funkcji `sizeof(tekscik)`.

```
char tekscik[] = "Ala ma kotala";
```

Poniżej również poprawna deklaracja tablicy (o łącznym rozmiarze 21 znaków). Liczba wierszy wyliczana jest automatycznie podczas kompilacji.

```
char teksty[][7] = {
    {"Ala"},
    {"ma"},
    {"kotala"}
};
```

To niepoprawna forma deklaracji:

```
char teksty[3][] = {
    {"Ala"},
    {"ma"},
    {"kotala"}
};
```

# Tablice jako argumenty funkcji

1. Trzeba bardzo uważać i myśleć zanim się coś zrobi!
2. Rzecz nie jest prosta (choć może nie aż tak skomplikowana).

# Przykład

- Chcemy napisać funkcję, która zeruje tablicę (wypełnia wartością zero wszystkie elementy tablicy).
- Sam algorytm jest bardzo prosty

```
int N = 10;
double A[N];
for(int i =0; i < N; i++)
    A[i] = 0.;
```

- Musimy go tylko obudować w funkcję
  - nie zwraca parametrów
  - ma dwa parametry:
    1. tablicę
    2. jej rozmiar

# Przykład c.d.

```
void zeruj(int N, double A[])
{
   for(int i = 0; i < N; i++)
      A[i] = 0.;
}
```

A używać jej będziemy tak:

```
int main()
{
  double X[1000];
  int M = 1000;
//   ...
  zeruj(M, X);
//  ...
}
```