

# Algorytmy i struktury danych

Przypomnienie języka C

Aleksander Lamża  
ZKSB · Instytut Informatyki  
Uniwersytet Śląski w Katowicach

[aleksander.lamza@us.edu.pl](mailto:aleksander.lamza@us.edu.pl)

- Praktycznie o podstawach C
- Tablice, wskaźniki, funkcje
- Przykładowy program „Oceny”

# Praktycznie o C – prosty program

## Napiszmy prosty program

```
int main() {
```



Funkcja, od której wszystko się zaczyna

```
    printf("Zaczynamy\n");
```



Tu wyświetlamy tekst

```
}
```

## Deklarujemy zmienne

```
int main() {  
  
    printf("Zaczynamy\n");  
  
    int ile;  
    ile = 10;  
  
    float a = 1.0;  
  
    printf("ile=%d a=%f\n", ile, a);  
  
}
```

Deklarujemy zmienną  
i przypisujemy wartość

To samo, ale krócej

Wyświetlamy wartości zmiennych

# Praktycznie o C – pętla for

## Wrzucamy pętlę for

```
int main() {  
  
    printf("Zaczynamy\n");  
  
    int ile;  
    ile = 10;  
  
    float a = 1.0;  
  
    printf("ile=%d a=%f\n", ile, a);  
  
    int i;  
    for (i=0; i<ile; ++i) {  
        printf("%f\n", a/i);  
    }  
}
```

← To jest licznik pętli

← Pętla leci od 0 do ile-1

← Wyświetlamy wartość wyrażenia  $a/i$

↑  
No dobra, a co się dzieje przy  $i=0$ ?

$a/0 = ???$

Trzeba coś z tym zrobić...

# Praktycznie o C – instrukcja if-else

## Dodajemy if-a

```
int main() {  
  
    printf("Zaczynamy\n");  
  
    int ile;  
    ile = 10;  
  
    float a = 1.0;  
  
    printf("ile=%d a=%f\n", ile, a);  
  
    int i;  
  
    for (i=0; i<ile; ++i) {  
        if (i != 0) {  
            printf("%f\n", a/i);  
        } else {  
            printf("pamiętaj cholero...\n");  
        }  
    }  
}
```

Jeśli i jest różne od 0...

W przeciwnym przypadku...

I już – prosty program gotowy.

Ale to nie wszystko – trzeba jeszcze pomówić  
o **tablicach, wskaźnikach, strukturach i funkcjach**.

# Praktycznie o C – tablice

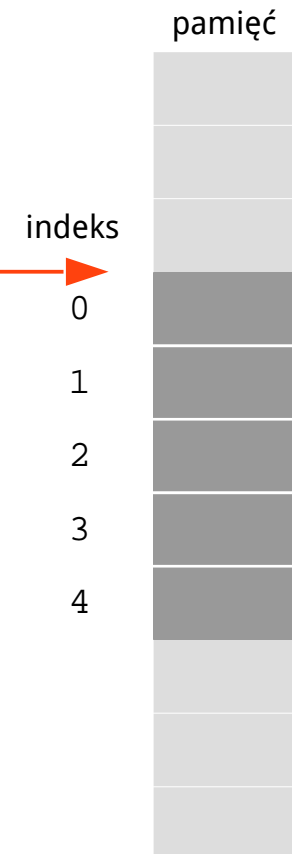
Po kolei – na początek **tablice**:

Najprostsza deklaracja tablicy wygląda następująco:

```
int tablica[5];
```

Tego typu tablice służą do przechowywania **określonej (niezmiennej w czasie działania programu) liczby danych tego samego typu.**

Do poszczególnych elementów tablicy odwołujemy się za pomocą tzw. **indeksu**, który przyjmuje wartości **od 0 do długości tablicy-1**





## Inne sposoby deklarowania tablic i ich inicjalizacji

```
int tablica[3];
```

?
?
?

Wartość elementów (komórek) tablicy jest nieznana i przypadkowa.

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

3
2
1

Poza deklaracją mamy tu inicjalizację, dzięki której ustalamy początkowe wartości komórek.

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

3
2
1

W deklaracji nie podajemy liczby elementów – jest ona ustalana na podstawie listy wartości.

```
int tablica[3] = {0};
```

0
0
0

W ten sposób możemy wyzerować wszystkie elementy tablicy.

Można też pominąć 0, czyli: { }

## Operowanie na elementach tablicy

```
int tablica[3] = {};  
  
tablica[0] = 13;  
  
int i;  
for (i=0; i<3; ++i) {  
    printf("%d\n", tablica[i]);  
}
```

indeks

0	0
1	0
2	0

indeks

0	13
1	0
2	0

**Uwaga!** Trzeba pilnować, by nie przekroczyć dozwolonych indeksów tablicy. Wyjście poza zakres (np. `tablica[5] = 1;` w powyższym przykładzie) może spowodować poważne problemy.

## Blizsze spotkanie ze zmiennymi (i ich adresami)

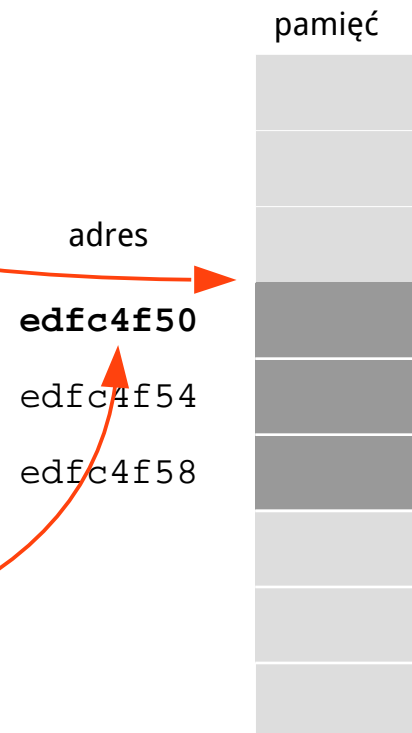
Tablica to nic innego jak zaalokowany fragment pamięci.

```
int t[3] = {};
```

Zmienna `t` określa początek tablicy, czyli pierwszy jej element (o indeksie 0).

Aby poznać adres, pod którym znajduje się tablica, musimy użyć **operatora adresu** (`&`):

```
printf("%x", &t);
```




## Blizsze spotkanie ze zmiennymi (i ich adresami)

Wiemy już, jak poznać adres tablicy w pamięci.

Czy ten sam mechanizm działa dla „zwykłych” zmiennych?

```
double e = 2.72;
```

```
printf("%f @ %x", e, &e);
```



2.720000 @ d8a70fb8

Oczywiście, że działa, ponieważ każda zmienna musi się znajdować w pamięci, a więc musi posiadać adres.

# Praktycznie o C – wskaźniki

## Wskaźniki

Uzyskiwanie adresów zadeklarowanych zmiennych to nie wszystko.

Mamy możliwość zadeklarowania zmiennych, które będą wskazywały na pewne obszary pamięci (np. inne zmienne). Są to tzw. **wskaźniki**.

```
double e = 2.72;  
printf("%f @ %x", e, &e);  
  
double* pe = &e;  
printf("%f @ %x", *pe, pe);
```

2.720000 @ d8a70fb8  
2.720000 @ d8a70fb8

Aby uzyskać wartość wskazywanej zmiennej, musimy ją **wyłuskać** (operatorem \*).

Zmienna **pe** zawiera **adres** zmiennej **e**.

d8a70fb8

**e**

2.72

Teraz napiszemy prosty program, na przykładzie którego uda się zilustrować kilka istotnych zagadnień

## **Wymagania i założenia:**

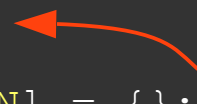
- Program ma służyć do przechowywania ocen
- Ma dawać możliwość wyliczenia średniej
- Zakładamy, że liczba ocen jest stała i niezmienna

# Program „Oceny” - struktura przechowująca dane

Zaczynamy od stworzenia struktury przechowującej oceny

Będzie to oczywiście tablica:

```
#define LICZBA_OCEN 5  
float oceny[LICZBA_OCEN] = {};
```



Dla ułatwienia definiujemy stałą `LICZBA_OCEN`, której będziemy używać zamiast wartości liczbowej.

Dzięki temu w przypadku konieczności zmiany liczby ocen wystarczy zmienić wartość w jednym miejscu.

Ustawianie wartości ocen jest proste:

```
int main() {  
    oceny[0] = 4.5;  
    oceny[1] = 5;  
    oceny[2] = 3.5;  
    ...  
}
```

Wypadałoby napisać teraz kod wyświetlający wszystkie oceny...

# Program „Oceny” - wyświetlanie ocen

Do wyświetlenia ocen użyjemy pętli `for`

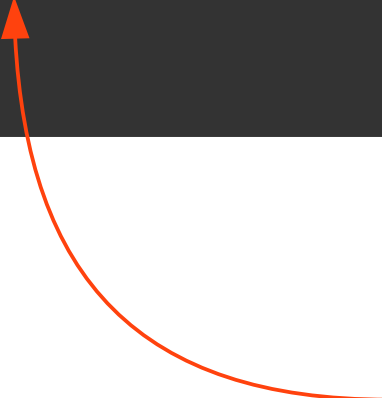
```
#define LICZBA_OCEN 5

float oceny[LICZBA_OCEN] = {};

int main() {

    oceny[0] = 4.5;
    oceny[1] = 5;
    oceny[2] = 3.5;

    int i;
    for (i=0; i<LICZBA_OCEN; ++i) {
        printf("%.1f\n", oceny[i]);
    }
}
```



Wartość oceny formatujemy tak, aby było wyświetlane tylko jedno miejsce po przecinku (`%.1f`).

Po uruchomieniu programu uzyskamy taki wynik:

4.5  
5.0  
3.5  
0.0  
0.0



# Program „Oceny” - wyświetlanie ocen w funkcji

A co jeśli będziemy chcieli ponownie wyświetlić oceny?

No właśnie – gdybym ustawił kolejną ocenę i chciał ponownie wyświetlić wszystkie, musiałbym jeszcze raz wstawić w kod pętlę wyświetlającą.  
To oczywiście niedorzeczny pomysł!

Trzeba utworzyć funkcję:

```
#define LICZBA_OCEN 5
float oceny[LICZBA_OCEN] = {};

void wyswietl() {
    int i;
    for (i=0; i<LICZBA_OCEN; ++i) {
        printf("%.1f\n", oceny[i]);
    }
}

int main() {
    oceny[0] = 4.5;
    oceny[1] = 5;
    oceny[2] = 3.5;
    wyswietl();

    oceny[3] = 3;
    oceny[4] = 5;
    wyswietl();
}
```

Pętlę wyświetlającą zawartość tablicy umieszczamy w funkcji `void wyswietl()`.

Dzięki temu za każdym razem, gdy chcemy wyświetlić oceny, wystarczy ją wywołać:

`wyswietl();`

Po uruchomieniu programu uzyskamy to:

4.5  
5.0  
3.5  
0.0  
0.0  
4.5  
5.0  
3.5  
3.0  
5.0

# Program „Oceny” - liczymy średnią

Idziemy za ciosem – tworzymy funkcję obliczającą średnią

```
#define LICZBA_OCEN 5
float oceny[LICZBA_OCEN] = {};

void wyswietl() {
    int i;
    for (i=0; i<LICZBA_OCEN; ++i) {
        printf("%.1f\n", oceny[i]);
    }
}

float srednia() {
    ...
}

int main() {
    oceny[0] = 4.5;
    oceny[1] = 5;
    oceny[2] = 3.5;
    wyswietl();

    oceny[3] = 3;
    oceny[4] = 5;
    wyswietl();

    printf("Średnia: %.2f", srednia());
}
```

Dodajemy funkcję `float srednia()`.

Zwróćcie uwagę na `float`.  
W funkcji `wyswietl()` było `void`. Czemu?

Tutaj wywołujemy funkcję. Wartość, którą zwróci, zostanie wyświetlona jako liczba z dwoma miejscami po przecinku.

# Program „Oceny” - liczymy średnią

A to wewnątrz funkcji `srednia()`

```
float srednia() {  
    float suma = 0;  
    int i;  
    for (i=0; i<LICZBA_OCEN; ++i) {  
        suma = suma + oceny[i];  
    }  
    return suma / LICZBA_OCEN;  
}
```

W pętli sumujemy wszystkie oceny.

Zamiast wyrażenia:

`suma = suma + oceny[i]`

można użyć skrótowego zapisu:

`suma += oceny[i]`

Zwracamy (`return`) wynik dzielenia sumy przez liczbę ocen.

No i gotowe!

## Omówiliśmy:

- budowę programu w C
- zmienne
- wskaźniki
- podstawowe instrukcje
- tablice
- funkcje

## Trzeba jeszcze omówić:

- struktury
- język C++

Ale to już na następnym wykładzie

