

# Algorytmy i struktury danych

Przypomnienie języka C  
(ciąg dalszy)

Aleksander Lamża  
ZKSB · Instytut Informatyki  
Uniwersytet Śląski w Katowicach

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

- Kombinujemy z tablicami
- Więcej wskaźników
- Struktury
- Jeszcze więcej wskaźników

# Program „Oceny” - przypomnienie

W poprzednim odcinku...

```
#define LICZBA_OCEN 5

float oceny[LICZBA_OCEN] = {};

void wyswietl() {
    ...
}

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());
}
```

Oceny przechowujemy w **globalnej** zmiennej (tablicy).

Mamy też dwie funkcje, które korzystają z tej tablicy.

W kodzie głównej funkcji również odwołujemy się do tej globalnej tablicy.

To nie za dobrze, że tablica jest zmienną globalną...

**Co należałoby zrobić, gdybyśmy chcieli przechowywać w programie oceny wielu studentów?**

# Program „Oceny” - więcej tablic

## Wielu studentów – wiele tablic?

Oczywiście moglibyśmy dla każdego studenta utworzyć osobną tablicę:

```
#define LICZBA_OCEN 5

float oceny1[LICZBA_OCEN] = {};
float oceny2[LICZBA_OCEN] = {};
float oceny3[LICZBA_OCEN] = {};
...
```

W takim przypadku należałoby zmodyfikować funkcje `wyswietl()` i `srednia()`:

```
void wyswietl(float* o) {
    ...
}

float srednia(float* o) {
    ...
}
```

Dzięki temu będziemy mieli możliwość wskazania tablicy, która ma zostać wyświetlona:

```
wyswietl(&oceny1);
```

lub

```
wyswietl(oceny1);
```

# Program „Oceny” - tablica wielowymiarowa

## Wielu studentów – tablica wielowymiarowa?

Moglibyśmy też użyć tablicy dwuwymiarowej:

```
#define LICZBA_OCEN 5
#define LICZBA_STUD 10

float oceny[LICZBA_STUD][LICZBA_OCEN] = {};
```

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

# Program „Oceny” - tablica wielowymiarowa

## Wielu studentów – tablica wielowymiarowa?

Funkcje `wyswietl()` i `srednia()` trzeba zmodyfikować tak samo, jak wcześniej. Wypełnianie tablicy i wywołanie funkcji wyglądałoby tak:

```
oceny[0][0] = 4.5;  
oceny[0][1] = 5;  
oceny[0][2] = 3.5;  
oceny[0][3] = 3;  
oceny[0][4] = 5;  
  
wyswietl(oceny[0]);
```

	0	1	2	3	4
0	4.5	5	3.5	3	5
1					
2					
3					
4					
5					
6					
7					
8					
9					

# Program „Oceny” - imię i nazwisko studenta

A co, gdybyśmy chcieli – poza ocenami – zapisywać też imię i nazwisko studenta?

Można utworzyć **osobną tablicę** dla imienia i nazwiska.

Sprawa zacznie się jednak komplikować, gdy postanowimy dodać kolejne dane, np. numer indeksu, adres mailowy itd.

Powstałoby kilka do kilkunastu tablic – praca z tak zorganizowaną strukturą danych nie należy do przyjemności...

Dużo lepszym rozwiązaniem będzie zastosowanie **struktur**.

# Program „Oceny” - struktury

Struktury umożliwiają zgrupowanie danych różnych typów

W naszym przypadku struktura wyglądałaby następująco:

```
struct Student {  
    char imie_nazwisko[50];  
    float oceny[LICZBA_OCEN];  
};
```

Tekst przechowujemy  
w tablicach znaków.

Aby móc z niej korzystać, należy utworzyć jej egzemplarz:

```
struct Student s;
```

Zmienna `s` zawiera dwa pola, do których dostajemy się za pomocą notacji:

```
strcpy(s.imie_nazwisko, "Natalia Pilna");  
s.oceny[0] = 5;
```

W ten sposób kopiujemy stringi.



Struktury można też definiować inaczej:

```
typedef struct {  
    char imie_nazwisko[50];  
    float oceny[LICZBA_OCEN];  
} Student;
```

Dzięki takiemu zapisowi nie trzeba powtarzać słowa `struct` w deklaracjach:

```
Student s;
```

# Program „Oceny” - struktury

Co nieco trzeba będzie zmodyfikować:

Funkcjom `wyswietl()` i `srednia()` będziemy przekazywać strukturę, a nie tablicę:

```
void wyswietl(Student* student) {  
    ...  
}  
  
float srednia(Student* student) {  
    ...  
}
```

A w zasadzie wskaźnik  
na strukturę.



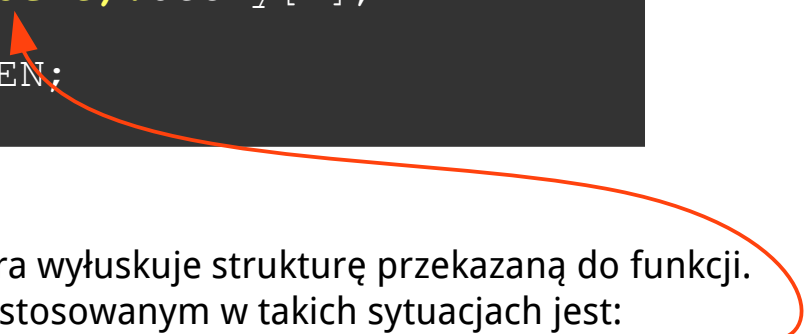
Kod z głównej funkcji również ulegnie zmianie:

```
Student s;  
strcpy(s.imie_nazwisko, "Natalia Pilna");  
s.oceny[0] = 4.5;  
s.oceny[1] = 5;  
s.oceny[2] = 3.5;  
s.oceny[3] = 3;  
s.oceny[4] = 5;  
wyswietl(&s);  
printf("Średnia: %.2f", srednia(&s));
```

## Modyfikacje funkcji

W funkcjach `wyswietl()` i `srednia()` musimy się odwoływać do pól struktury, a nie globalnych zmiennych:

```
void wyswietl(Student* student) {  
    printf("%s\n", (*student).imie_nazwisko);  
  
    int i;  
    for (i=0; i<LICZBA_OCEN; ++i) {  
        printf("%.1f\n", (*student).oceny[i]);  
    }  
}  
  
float srednia(Student* student) {  
    float suma = 0;  
    int i;  
    for (i=0; i<LICZBA_OCEN; ++i) {  
        suma = suma + (*student).oceny[i];  
    }  
    return suma / LICZBA_OCEN;  
}
```



Powtarza się tu konstrukcja `(*student)`, która wyluskuje strukturę przekazaną do funkcji. Znacznie bardziej wygodnym zapisem stosowanym w takich sytuacjach jest:

`student->oceny[i]` == `(*student).oceny[i]`

# Program „Oceny” - tablica struktur

Wróćmy do pytania, co by było, gdybyśmy chcieli więcej studentów

Możemy utworzyć tablicę struktur:

```
Student grupa[12];  
  
strcpy(grupa[0].imie_nazwisko, "Natalia Pilna");  
grupa[0].oceny[0] = 4.5;  
grupa[0].oceny[1] = 5;  
grupa[0].oceny[2] = 3.5;  
grupa[0].oceny[3] = 3;  
grupa[0].oceny[4] = 5;  
wyswietl(&grupa[0]);  
printf("Średnia: %.2f", srednia(&grupa[0]));
```

Tu tworzymy tablicę złożoną z 12 elementów typu Student.

Wypełniamy dane dla pierwszego studenta (o indeksie 0).

# Program „Oceny” - podsumowanie struktur

Struktury są dosyć wygodne, bo umożliwiają zamknięcie „w pudełku” różnych danych opisujących jedno „coś” (w tym przypadku studenta).

imie_nazwisko	Natalia Pilna
nr_indeksu	5471302
e_mail	natpil@buziaczek.pl
oceny	5.0   4.5   3.5   5.0   5.0   4.5

A gdyby pójść o krok dalej i zamknąć w pudełku z danymi również **funkcje**, które operują na tych danych?

O tym w kolejnej prezentacji...