Anna Jasielec

Podstawy programowania, grupa nr 4

Sprawozdanie z laboratorium nr 10, 11

Cel laboratorium: Opanowanie podstaw tworzenia i wykorzystania struktur w C.

Przebieg zajęć:
Utworzyłam katalog roboczy lab_10 i skopiowałam ze strony przedmiotu plik struktury_szablon.c. Zaprojektowałam własną strukturę, której parametry są zmiennymi różnych typów. Wykorzystałam do tego funkcjonalność typedef.

```
typedef struct
{
  char* imie;
  int wiek; //$ w latach
  double waga; //$ w kilogramach
}Kot;
```

Następnie zajęłam się pisaniem programu z funkcją main. Zdefiniowałam *kot_1* będący strukturą zaprojektowanego typu i nadałam jego parametrom wartości posługując się operatorem składowych (**.**). Wypisałam na ekranie wartości parametrów posługując się operatorem składowych (**.**).

```
//# 4.1 definicja zmiennej (np. obiekt_1) typu strukturalnego
Kot kot_1;

//# 4.2 nadanie wartości pól zmiennej obiekt_1 z pomocą operatora .
kot_1.imie = "Mariusz";
kot_1.wiek = 3;
kot_1.waga = 5.23;

//# 4.3 wypisanie wartości pól zmiennej obiekt_1 z pomocą operatora .
//#   np. printf("Początkowe wartości pól obiekt_1: ....", ....);
printf("\nPoczątkowa wartość pól kot_1: imię: %s, wiek: %d, waga: %.2lf\n"
  ,kot_1.imie, kot_1.wiek, kot_1.waga);
```

```
Początkowa wartość pól kot_1: imię: Mariusz, wiek: 3, waga: 5.23
```

Zdefiniowałam *kot_2* będący strukturą, inicjując jednocześnie wartości składowych oraz zdefiniowałam wskaźnik do struktury inicjując adresem kot_2. Wypisuje na ekranie zawartość struktury posługując się wskaźnikiem i operatorem **–>**.

```
//# 4.4 definicja drugiej zmiennej typu strukturalnego  (np. obiekt_2)
// #   połączona z inicjowaniem za pomocą listy wartości
Kot kot_2 = {"Rudy", 5, 7.46};


//# 4.5 definicja wskaźnika (np. obiekt_2_wsk) do struktury zainicjowanego adresem obiektu_2
Kot *kot_2_wsk = &kot_2;

//# 4.6 wypisanie wartości pól obiektu_2 z pomocą wskaźnika do obiektu i operatora ->
printf("\n\nPoczątkowe wartości pól kot_2: imię: %s, wiek: %d, waga: %.2lf\n",
  kot_2_wsk->imie, kot_2_wsk->wiek, kot_2_wsk->waga);
```

```
Początkowe wartości pól kot_2: imię: Rudy, wiek: 5, waga: 7.46
```

Następnie zdefiniowałam kot_3 będący strukturą, inicjując jednocześnie wartości składowych poprzez przypisanie (skopiowanie) wartości pól z kot_2. Wypisuje na ekranie wartości pól obiektu_3 posługując się operatorem składowych (.).

```
Kot kot_3 = kot_2;

//# 4.8 wypisanie wartości pól zmiennej obiekt_3 z pomocą operatora .
printf("\n\nPoczątkowe wartości pól kot_3: imię: %s, wiek: %d, waga: %.2lf\n",
       kot_3.imie, kot_3.wiek, kot_3.waga);
```

```
Początkowe wartości pól kot_3: imię: Franciszek, wiek: 12, waga: 7.99
```

Zmodyfikowałam program pisząc kolejne funkcje, wywoływanych następnie przez funkcje main. Napisałam funkcję, która przyjmuje jako argument wejściowy obiekt będący strukturą, modyfikuje składowe struktury i wypisuje na ekranie nowe wartości. W funkcji main dodałam wywołanie tej funkcji z kot_1 jako argumentem. Ponownie wypisałam wartości składowych kot_1 po powrocie z funkcji.

```
void fun_struct(Kot kot_w_funkcji) {
  //# wypisanie wartości pól struktury przesłanej jako argument
  printf("Wewnątrz fun_strukt - wartości pól obiektu argumentu:\nimię: %s, "
         "wiek: %d, waga: %.2lf\n",
         kot_w_funkcji.imie, kot_w_funkcji.wiek, kot_w_funkcji.waga);

  //# modyfikacja wartości pól struktury
  kot_w_funkcji.imie = "Michalina";
  kot_w_funkcji.wiek = 4;
  kot_w_funkcji.waga = 7.12;

  //# wypisanie zmodyfikowanych wartości pól struktury
  printf("\nWewnątrz fun_strukt - wartości pól po modyfikacji:\nimię: %s, "
         "wiek: %d, waga: %.2lf\n",
         kot_w_funkcji.imie, kot_w_funkcji.wiek, kot_w_funkcji.waga);
}
```

```
Wewnątrz fun_strukt - wartości pól obiektu argumentu:
imię: Mariusz, wiek: 3, waga: 5.23

Wewnątrz fun_strukt - wartości pól po modyfikacji:
imię: Michalina, wiek: 4, waga: 7.12

Po wywołaniu fun_strukt - wartości pól kot_1:
imię: Mariusz, wiek: 3, waga: 5.23
```

Kolejna funkcja przyjmuje obiekt będący strukturą jako argument wejściowy, modyfikuje składowe struktury, wypisuje na ekranie nowe wartości i zwraca obiekt ze zmodyfikowanymi wartościami. Uzupełniłam funkcje main o wywołanie tej funkcji z kot_1 jako argumentem. Przypisałam zwracaną strukturę kot_1 i wypisałam wartości pól kot_1 po skopiowaniu.

```
Kot fun_struct_out(Kot kot_out) {
  //# wypisanie wartości pól struktury przesłanej jako argument
  printf("\n\nWewnątrz fun_strukt_out - wartości pól obiektu argumentu:\nimię: "
         "%s, wiek: %d, waga: %.2lf\n",
         kot_out.imie, kot_out.wiek, kot_out.waga);

  //# modyfikacja wartości pól struktury
  kot_out.imie = "Andrzej";
  kot_out.wiek = 7;
  kot_out.waga = 6.98;

  //# wypisanie zmodyfikowanych wartości pól struktury
  printf("\nWewnątrz fun_strukt_out - wartości pól po modyfikacji:\nimię: %s, "
         "wiek: %d, waga: %.2lf\n",
         kot_out.imie, kot_out.wiek, kot_out.waga);

  //# zwrócenie struktury ze zmodyfikowanymi wartościami pól do funkcji
  // wywołującej
  return (kot_out);
}
```

```
Wewnątrz fun_strukt_out - wartości pól obiektu argumentu:
imię: Mariusz, wiek: 3, waga: 5.23

Wewnątrz fun_strukt_out - wartości pól po modyfikacji:
imię: Andrzej, wiek: 7, waga: 6.98

Po wywołaniu fun_strukt_out i przypisaniu wyniku do kot_1 - wartości pól kot_1:
imię: Andrzej, wiek: 7, waga: 6.98
```

Następna funkcja przyjmuje jako argument wejściowy wskaźnik do obiektu będącego strukturą, modyfikuje składowe struktury i wypisuje na ekranie nowe wartości. W funkcji main wywołałam funkcję z kot_2 jako argumentem i wypisałam wartości składowych po powrocie z funkcji.

```
void fun_struct_wsk(Kot *kot_2) {
  //# wypisanie wartości pól struktury wskaźnik do której został przesłany jako
  // argument
  printf("\nWewnątrz fun_strukt_wsk - wartości pól obiektu argumentu:\nimię: "
         "%s, wiek: %d, waga: %.2lf\n",
         kot_2->imie, kot_2->wiek, kot_2->waga);

  //# modyfikacja wartości pól struktury
  (*kot_2).imie = "Franciszek";
  kot_2->wiek = 12;
  kot_2->waga = 7.99;

  //# wypisanie zmodyfikowanych wartości pól struktury
  printf("\nWewnątrz fun_strukt_wsk - wartości pól po modyfikacji:\nimię: %s, "
         "wiek: %d, waga: %.2lf\n",
         kot_2->imie, kot_2->wiek, (*kot_2).waga);
}
```

```
Wewnątrz fun_strukt_wsk - wartości pól obiektu argumentu:
imię: Rudy, wiek: 5, waga: 7.46

Wewnątrz fun_strukt_wsk - wartości pól po modyfikacji:
imię: Franciszek, wiek: 12, waga: 7.99

Po wywołaniu fun_strukt_wsk - wartości pól kot_2:
imię: Franciszek, wiek: 12, waga: 7.99
```

Następnie napisałam funkcję, która przyjmuje jako argument wejściowy wskaźnik do obiektu będącego strukturą, przepisuje zawartości struktury do zmiennej lokalnej, modyfikuje składowe tej zmiennej, wypisuje na ekranie nowe wartości i zwraca strukturę będącą zmienną lokalną. W funkcji main dodałam wywołanie funkcji z adresem kot_3 jako argumentem, przypisałam zwracaną przez funkcję strukturę do nowej zmiennej kot_4 zaprojektowanego typu.

```
Kot fun_struct_wsk_kopia(Kot *kot_wsk) {
  //$ przepisania zawartości struktury do zmiennej lokalnej
  Kot nowy_kot = *kot_wsk;

  //$ modyfikacja zawartoci struktury do zmiennej lokalnej
  nowy_kot.imie = "Bobik";
  nowy_kot.wiek = 20;
  nowy_kot.waga = 10.23;

  //$ wypisanie nowych wartości
  printf("\nwewnątrz fun_strukt_wsk_kopia - wartości po modyfikacji:\nimię: "
         "%s, wiek: %d, waga: %.2lf\n",
         nowy_kot.imie, nowy_kot.wiek, nowy_kot.waga);

  return (nowy_kot);
}
Kot kot_4 = fun_struct_wsk_kopia(&kot_3);
```

```
wewnątrz fun_strukt_wsk_kopia - wartości po modyfikacji:
imię: Bobik, wiek: 20, waga: 10.23


Początkowe wartości pól kot_4: imię: Bobik, wiek: 20, waga: 10.23
```

Kolejna funkcja przyjmuje wskaźnik do obiektu będącego strukturą jako argument, przepisuje zawartość struktury do zmiennej lokalnej posługując się operatorami przypisania i wyłuskania, modyfikuje składowe struktury będącej zmienną lokalną i przepisuje zawartość struktury ze zmiennej lokalnej do struktury w funkcji main. Funkcję main uzupełniłam o wywołanie funkcji z adresem kot_4 jako argumentem.

```c
void fun_struct_wsk_inout(Kot *kot_wsk) {
  //$ przepisanie wartosci struktury do zmiennej
  Kot kotek_lokalny = *kot_wsk;

  //$ modyfikacja skladowych
  kotek_lokalny.imie = "Mojzesz";
  kotek_lokalny.wiek = 10;
  kotek_lokalny.waga = 9.99;

  //$ wypisanie nowych wartosci
  printf("\nWewnątrz fun_strukt_wsk_inout - wartości po modyfikacji:\nimię: "
         "%s, wiek: %d, waga: %.2lf\n",
         kotek_lokalny.imie, kotek_lokalny.wiek, kotek_lokalny.waga);

  //$ przepisanie zawartości struktury ze zmiennej lokalnej do struktury w funkcji wywołującej
  *kot_wsk = kotek_lokalny;
}
```

```
Wewnątrz fun_strukt_wsk_inout - wartości po modyfikacji:
imię: Mojzesz, wiek: 10, waga: 9.99

Po wywołaniu fun_strukt_wsk_inout - wartości pól kot_4: imię: Mojzesz, wiek: 10, waga: 9.99
```

Następna funkcja przyjmuje jako argument strukturę, dokonuje alokacji pamięci dla nowej struktury, przepisuje zawartość tej struktury do nowej struktury w obszarze dynamicznym i zwraca wskaźnik do zaalokowanej struktury. Sprawdzenie czy nie została zwrócona wartość NULL, zwolnienie pamięci w funkcji main.

```c
Kot* fun_struct_wsk_out(Kot kot_out) {

  //$ alokacja pamieci za pomoca funkcji malloc
  Kot *kot_wsk_out = (Kot *)malloc(sizeof(Kot));

  //$ przepisanie zawartosci kot_out do kot_wsk_out
  kot_wsk_out -> imie = "Alojzy";
  kot_wsk_out -> wiek = 2;
  kot_wsk_out -> waga = 4.5;

  //$ sprawdzenie czy malloc nie zrocil wartosci NULL
  if (kot_wsk_out == NULL) {
    printf("Funkcja malloc zwróciła wartość NULL. Przerwanie programu.");
    exit(0);
  }

  return (kot_wsk_out);
}
```

```c
//$ wywolanie funkcji fun_struct_wsk_out
  Kot *kot_5 = fun_struct_wsk_out(kot_4);
```

```c
//$ zwalnianie pamieci przydzielonej przez malloc
  free(kot_5);
```

```
Po wywołaniu fun_strukt_wsk_out - wartości pól kot_5: imię: Alojzy, wiek: 2, waga: 4.50
```

Zaprojektowałam nowy typ strukturalny zawierającej pola, pośród których znajduje się tablica znaków.

```
typedef struct {
  double waga;
  char imie[N];
  int wiek;
} Pies;
```

```
#define N 5
```

Zbadałam za pomocą operatora sizeof rozmiar pojedynczej zmiennej zaprojektowanego typu dla N równych 1,2,3,4,5 itd.

```
Rozmiar struktury Pies dla 5 elementow w tablicy znakow: 24
Rozmiar struktury Pies dla 4 elementow w tablicy znakow: 16
Rozmiar struktury Pies dla 3 elementow w tablicy znakow: 16
Rozmiar struktury Pies dla 2 elementow w tablicy znakow: 16
```

Wyrównanie jest zawsze do największego obsługiwanego na danej architekturze typu prostego (char-1, int-4, double-8).

Zaprojektowałam nowy typ strukturalny, którego jednym z pól jest wskaźnik do zmiennych tego właśnie typu.

```
struct Wskaznik {
 int zmienna;
 struct Wskaznik *p;
};
```

Polem nie może być zmienna tego samego typu, ponieważ wtedy struktura by sie zapętlała, natomiast możemy zdefiniować wskaźnik do zmiennej typu, wtedy wskazuje on na jej miejsce.

Wnioski:

- Słowo kluczowe *typedef* umożliwia przypisanie własnej nazwy do istniejących typów.
- Parametrom struktury można nadać wartość za pomocą operatora **.** (zmienna.parametr = …), za pomocą listy wartości (zmienna = {… , … , …}) lub poprzez skopiowanie (zmienna1 = zmienna2).
- Posługując się wskaźnikiem do struktury zainicjowanego adresem obiektu, zamiast operatora **.** użyjemy **->** (lub *(*wskaźnik).parametr* = …).
- Można zmienić wartości parametrów struktury tworząc funkcję, która jako argument przyjmuje obiekt będący strukturą i zwraca obiekt ze zmodyfikowanym wartościami, zwrócony obiekt należy skopiować do obiektu użytego jako argument. Szybszym sposobem jest stworzenie funkcji, która przyjmuje wskaźnik do obiektu będącego strukturą i za jego pomocą modyfikuje wartości parametrów obiektu.

- Umiejętne wykorzystanie wskaźników ułatwia operacje na obiektach będących strukturami.
- Funkcji *malloc* służy do alokowania pamięci, należy pamiętać o sprawdzeniu, czy nie została zwrócona wartość NULL oraz o zwolnieniu pamięci.
- Można sprawdzić rozmiar struktury za pomocą *sizeof(Struktura),* wyrównanie jest zawsze do największego obsługiwanego na danej architekturze typu prostego (char-1, int-4, double-8).
- Polem struktury nie może być zmienna tego samego typu, ponieważ wtedy struktura by sie zapętlała, natomiast możemy zdefiniować wskaźnik do zmiennej tego typu.