Laboratorium 8: „Struktury i unie”

dr inż. Leszek Ciopiński
dr inż. Arkadiusz Chrobot
dr inż. Grzegorz Łukawski

4 września 2026

1. Wprowadzenie

W niniejszej instrukcji zawarto informacje na temat wybranych integralnych typów danych języka C. W rozdziale pierwszym opisano struktury, a w rozdziale drugim poświęcono uniom.

2. Struktury

Struktura w języku C jest odpowiednikiem rekordu znanego z języka Pascal. Ich celem jest gromadzenie zmiennych, niekoniecznie tego samego typu, w jedną całość. Definiuje się ją za pomocą słowa kluczowego `struct`. Po nim należy podać **nazwę typu struktury** i w nawiasach klamrowych zadeklarować pola, które mają należeć do tej struktury. Definicja struktury kończy się średnikiem. Jego brak jest częstym błędem popełnianym przez początkujących programistów języka C. Przed średnikiem może wystąpić nazwa zmiennej lub nawet kilka nazw zmiennych rozdzielonych przecinkami. Możemy jednak je pominąć i zadeklarować zmienną będącą strukturą w innym miejscu programu. Struktury możemy (a nawet powinniśmy) przekazywać do funkcji przez parametr. Program z listingu 1 pokazuje w jaki sposób możemy posługiwać się strukturami. Jeśli chcemy, aby po zakończeniu funkcji, zmiany wartości pól

```c
#include<stdio.h>

struct numbers
{
        int a, b;
        double c;
} num;

void print_numbers(struct numbers a)
{
        printf("Wartości w strukturze: %d %d %lf\n",a.a,a.b,a.c);
}

int main(void)
{
        // Przypisanie wartości do poszczególnych elementów struktury.
        num.a = 5;
        num.b = 6;
        num.c = 0.5;
        print_numbers(num);
        /* Inna struktura, tym razem jako zmienna lokalna.
           Proszę zwrócić uwagę na sposób inicjacji pól. */
        struct numbers another_numbers = {3, 4, 4.4};
        print_numbers(another_numbers);
        return 0;
}
```

Listing 1: Przykład użycia struktur

struktury były utrwalone, to musimy strukturę przekazać przez wskaźnik. Listing 2 pokazuje w jaki sposób można odwołać się do pól struktury wskazywanej przez wskaźnik. W praktyce najczęściej stosowany jest sposób ze „strzałką” zamiast kropki, bo jest krótszy i bardziej przejrzysty.

2.1. Pola bitowe

Język C pozwala określać rozmiar pól struktury w bitach. Takie pola są nazywane polami bitowymi. Listing 3 zawiera odpowiedni przykład. Rozmiar struktury jest całkowitą, dodatnią wielokrotnością jednego bajta i jest zawsze równy co najmniej jeden bajt, niezależnie od tego jakie pola bitowe są wewnątrz

```c
#include<stdio.h>

struct numbers
{
        int a, b;
        double c;
} num;

void set_numbers(struct numbers *s)
{
        (*s).a=1;
        s->b = 2;
        s->c = 3.5;
}

int main(void)
{
        set_numbers(&num);
        printf("Wartości w polach struktury: %d %d %lf\n",num.a,num.b,num.c);
        return 0;
}
```

Listing 2: Wskaźnik na strukturę

```c
#include<stdio.h>

struct bit_fields
{
        unsigned char a1:1,
                      a2:2,
                      a3:3;
} bf;

int main(void)
{
        printf("Rozmiar struktury: %u\n",sizeof(bf));
        bf.a1=1;
        printf("Wartość pola a1: %u\n",bf.a1);
        bf.a2=3;
        printf("Wartość pola a2: %u\n",bf.a2);
        bf.a2=4;        // To się nie zmieści.
        printf("Wartość pola a2: %u\n",bf.a2);
        return 0;
}
```

Listing 3: Pola bitowe

niej zdefiniowane. Do pól bitowych nie można zastosować operatora `sizeof`. Proszę również zwrócić uwagę, że nie da się do nich zapisać wartości większej niż wynika to z ich rozmiaru.

2.2. Tablice struktur

Język C pozwala na budowanie rozmaitych struktur danych. Przy budowie złożonych struktur danych można łączyć kilka różnych sposobów. Nie ma przeszkód, żeby z zadeklarowanej struktury zrobić tablicę, której typem bazowym będzie właśnie ta struktura. Nie ma również przeciwwskazań, aby nowo zadeklarowana struktura zawierała jeszcze inne tablice, struktury czy unie.

```c
#include<stdio.h>
#include<string.h>

struct Tosoba{
   char imie[15];
   unsigned char wiek;
};

int main(void){
   struct Tosoba osoby[10];  //utworzenie listy 10-ciu osób
   int i;
   osoby[0].wiek=19; // przypisanie wieku pierwszej osobie
   strcpy(osoby[0].imie, "Agnieszka"); // przypisanie imienia pierwszej osobie
   osoby[1].wiek=23; // przypisanie wieku drugiej osobie
   strcpy(osoby[1].imie, "Adam"); // przypisanie imienia drugiej osobie
   for(i=2; i<10;i++){
     printf("Podaj wiek osoby nr %d: ", i);
     scanf("%d", &osoby[i].wiek);
     printf("\nPodaj Imię osoby %d:" , i);
     scanf("%s", osoby[i].imie);
     printf("\n");
   }
   for(i=0; i<10; i++){
     printf("%d %s\n", osoby[i].wiek, osoby[i].imie);
   }
   return 0;
}
```

Listing 4: Przykład zbudowania tablicy na typie złożonym.

Listing 4 pokazuje przykład zadeklarowania struktury **`Tosoba`**, która zawiera tablicę (typ złożony), a następnie sama została wykorzystana do utworzenia tablicy **`osoby`** przechowującej informacje o kilku osobach.

3. Unie

Unie definiuje się tak jak struktury, ale zamiast słowa kluczowego **`struct`** używa się słowa **`union`**. Różnica między tymi dwoma typami zmiennych polega na tym, że w unii pola w pamięci operacyjnej są nałożone na siebie, a więc rozmiar unii jest równy rozmiarowi jej największego pola. Unie można wykorzystać do konwersji typów, np. do zamiany adresu IP komputera z postaci czterech liczb na postać pojedynczej liczby[1] lub do konwersji liczb w nieupakowanym kodzie BCD. Listing 5 pokazuje sposób użycia unii.

[1] Odpowiedni przykład znajduje się w Wikibooks, do których adres podany jest na stronie przedmiotu.

```c
#include<stdio.h>

struct struktura
{
        unsigned char a,b,c,d;
        unsigned int x;
} s;

union unia
{
        unsigned char a,b,c,d;
        unsigned int x;
} u;

int main(void)
{
        printf("Rozmiar struktury: %u\n",sizeof(s));
        printf("Rozmiar unii: %u\n",sizeof(u));
        s.x = u.x = 0xabcd; // Zapis liczby szesnastkowej
        printf("Wartości w strukturze: %u %u %u %u %u\n",s.a,s.b,s.c,s.d,s.x);
        printf("Wartości w unii: %u %u %u %u %u\n",u.a,u.b,u.c,u.d,u.x);

        return 0;
}
```

Listing 5: Unia

Używając unii należy pamiętać, że sposób nakładania pól jest zależny od kompilatora, opcji jakie zostały mu przekazane, a także od architektury docelowej maszyny, na której program będzie uruchamiany.

4. Zadania

Uwaga! Wszystkie programy należy napisać z podziałem na funkcje z parametrami.

1. Przy pomocy debuggera wykonaj krokowo program z listingu nr 5. Sprawdź jak zmieniają się wartości pól unii.
2. Zadeklaruj w programie strukturę zawierającą pola różnych typów. Następnie napisz funkcję, która wypisze rozmiar tej struktury na ekranie, oraz wypisze sumę rozmiaru wszystkich jej pól. Jak wytłumaczyć zjawisko, że nie zawsze wartości te są równe?
3. Zadeklaruj w programie unię zawierającą pola różnych typów. Następnie napisz funkcję, która wypisze rozmiar tej unii na ekranie, oraz wypisze sumę rozmiaru wszystkich jej pól. Jak wytłumaczyć zjawisko, że nie zawsze wartości te są równe?
4. Napisz program, który będzie wykonywał podstawowe operacje arytmetyczne na liczbach zespolonych (dodawanie, odejmowanie, mnożenie i dzielenie). Użyj struktury do reprezentowania liczb zespolonych. Wartości tych liczb powinny być wprowadzane przez użytkownika.
5. Napisz program, w którym stworzysz przy pomocy struktur, typów wyliczeniowych i unii strukturę danych mogącą przechowywać współrzędne punktu w dwuwymiarowym układzie kartezjańskim lub układzie biegunowym. Napisz funkcje, które będą umożliwiały konwertowanie jednej postaci współrzędnych na drugą i odwrotnie.
6. Uzupełnij zaprezentowany na wykładzie program z tablicą przechowującą dane osobowe o funkcje, które posortują tę tablicę według nazwiska i imienia.

7. Napisz program, w który wypełni tablicę 10 struktur przechowujących współrzędne punktów w dwuwymiarowym układzie kartezjańskim, wartościami składowych współrzędnych losowanymi z zakresu od -10 do 10, a następnie znajdzie dwa punkty, które oddalone są od siebie najbardziej. Tablicę trzeba wypisać po wypełnieniu na ekran.