Język C – zajęcia nr 12

Struktury i unie

Struktura jest zgromadzonym pod jedną nazwą zbiorem elementów nazywanych **polami** lub **składowymi**. Pola mają swoje odrębne nazwy (podobnie jak zmienne) i mogą przechowywać dane różnych typów. Definicja struktury służy wprowadzeniu nowego typu danych:

struct *nazwa* **{** *lista_deklaracji_pól* **} ;**

gdzie *nazwa* (**opcjonalna**) jest nazwą struktury jako nowego typu danych (nie mylić z nazwą danych), a *lista_deklaracji_pól* jest ciągiem deklaracji określających typy poszczególnych pól struktury. Deklaracja pola ma składniowo identyczną formę jak definicja zmiennej.

Każde pole w oddzielnej deklaracji:

```
struct kartoteka
{
    char imie[20];
    char nazwisko[40];
    int rok_ur;
    int mies_ur;
    int dz_ur;
    char miejsce_ur[30];
    char adres[50];
};
```

Wiele pól w jednej deklaracji:

```
struct kartoteka
{
    char imie[20], nazwisko[40];
    int rok_ur, mies_ur, dz_ur;
    char miejsce_ur[30];
    char adres[50];
};
```

Definicja struktury wprowadza jedynie nowy typ danych **bez tworzenia obiektu** tego typu w pamięci komputera. Definicja obiektu typu strukturalnego ma formę:

struct *nazwa* **{** *lista_deklaracji_pól* **}** *lista_deklaratorów* **;**

lub:

struct *typ_str lista_deklaratorów* **;**

W pierwszym przypadku **definiowany** jest typ strukturalny i równocześnie **definiowane** są obiekty tego typu. Drugi przypadek przedstawia zwykłą definicję lub deklarację obiektu pod warunkiem, że wcześniej zdefiniowano strukturę o nazwie *typ_str* .

Pola struktury mogą być również typu strukturalnego, co pozwala na konstruowanie złożonych zagnieżdżonych struktur danych.

Struktura zagnieżdżona:

```
struct data
{
    int rok;
    int mies;
    int dzien;
};

struct nazw
{
    char imie[20];
    char nazwisko[40];
    char pseudonim[25];
};

struct pracownik
{
    struct nazw nazwisko;
    struct data urodzenia;
    struct data przyjecia;
    struct data zwolnienia;
    char adres[50];
} pr1;                  //Definicja struktury i jednoczesnie
                        //definicja jednego obiektu

struct pracownik pr2, pr3;        //Definicja dwóch obiektów
```

Definicji obiektu strukturalnego może towarzyszyć jego inicjalizowanie. Inicjalizator ma postać ujętej w nawiasy klamrowe { } listy wartości oddzielonych przecinkami. Inicjalizowanie pól będących strukturami wymaga dodatkowych wewnętrznych par nawiasów { }.

```
struct data {int rok, mies, dzien;};

struct nazw {char imie[20], nazwisko[40], pseudonim[25];};

struct pracownik { struct nazw nazwisko;

                   struct data urodz, przyjecia, zwolnienia;

                   char adres[50];};

struct pracownik k = {{"Jan","Kowalski","Puchatek"},
{1950,12,28}, {1990,4,12}, {1994,10,7},"Krakow, ul. Sowia 4"};
```

Ograniczenie: Typem pola definiowanej struktury nie może być ta struktura. Zabronione są definicje postaci:

```
struct alfa

{

    int k;

    struct alfa x;  //Blad! pole x nie może byc typu alfa

};
```

Natomiast typem pola definiowanej struktury **może** być **wskaźnik** do tej struktury.

Unia jest specyficznym rodzajem struktury o podobnej do niej deklaracji:

union *nazwa* **{** *lista_deklaracji_pól* **} ;**

Pola (składowe) **unii** w odróżnieniu od **struktury** lokowane są w pamięci komputera w tym samym obszarze. To powoduje, że wszystkie pola unii nałożone są jedno na drugie i każdorazowo tylko zawartość jednego z nich jest dostępna. Wykorzystanie unii daje możliwość oszczędności pamięci w przypadku, gdy nie wszystkie dane muszą być przechowywane podczas całego wykonania programu. Unie podobnie jak struktury mogą tworzyć zagnieżdżone konstrukcje. Podobnie jak w przypadku struktur *nazwa* w definicji unii jest opcjonalna.

Uwaga: Rozmiar unii jest równy największemu rozmiarowi wśród pól tej unii.

Dostęp do składowych struktur i unii. Operator składowej.

Do poszczególnych składowych struktury i unii można uzyskać dostęp za pomocą dwóch operatorów:

- **.** operator **składowej**
- **->** operator **wskaźnikowy składowej**

Jeżeli **x** jest obiektem typu strukturalnego, a **n** jest nazwą pola tej struktury, to wyrażenie **x.n** ma wartość pola **n** obiektu **x**. Jeżeli **p** jest wskaźnikiem na obiekt typu strukturalnego, a **n** jest nazwą pola tej struktury, to wyrażenie **p->n** ma wartość pola o nazwie **n** tego obiektu. W podobny sposób uzyskiwany jest dostęp do pól unii.

W przypadku zagnieżdżonych struktur lub unii wykorzystuje się złożenie operatorów składowej i wskaźnikowy składowej.

```
x.a.n

p->b.k

t.c->m->y
```

Uwaga: Jak łatwo zauważyć, wyrażenie:

p->a

jest jedynie skrótowym zapisem równoważnego wyrażenia:

(*p).a

Dokonaj analizy programu:

```
#include <stdio.h>

int main()

{

    union uu

    {

        float f;

        long int i;

    }x;

    scanf("%f",&(x.f));       //Wczytanie liczby rzeczywistej i

    printf("%8.8lX",x.i);     //wyswietlenie szesnastkowe jej
                              //obrazu w pamieci komputera

    getch();

}
```

```
174.37                        KLAWIATURA 174.37↵

432E5EB8
```

Deklaracje i definicja struktury

Podstawową zasadą dotyczącą struktur jest zakaz definiowania obiektu typu strukturalnego przed osiągnięciem końca definicji danej struktury. Wynika z tego, że typem pola definiowanej struktury nie może być ona sama, natomiast może tym typem być wskaźnik na deklarowaną strukturę, co jest często wykorzystywane przy tworzeniu listowych typów danych.

Element dwukierunkowej listy struktur:

```
struct element

{

   char *dane;

   struct element *poprzedni; //Wskaznik na poprzedni element
                              //listy

   struct element *nastepny;  //Wskaznik na nastepny element
                               //listy

}
```

Istnieje zasadnicza różnica pomiędzy definicją i deklaracją struktury:

Definicja struktury precyzuje układ i typ jej poszczególnych pól,
Deklaracja struktury ustala jedynie nazwę typu danej struktury.

Wykorzystanie deklaracji struktury staje się konieczne przy definiowaniu układu dwóch wzajemnie związanych struktur, z których każda zawiera pole będące wskaźnikiem na pozostałą strukturę. Deklaracja jednej ze struktur nie pozwala na definicję pola takiego typu, ale pozwala na definicję wskaźnika do zadeklarowanej struktury.

Deklaracja i definicja struktury:

```
struct alfa;   //Deklaracja struktury typu alfa
struct beta    //Definicja struktury typu beta
{
   struct alfa* p;  //Wskaznik na strukture typu alfa
};
struct alfa    //Definicja struktuty typu alfa
{
   struct beta* q;  //Wskaznik na strukture typu beta
};
```

Deklaracja `typedef`

W języku C można przypisać nową (alternatywną) nazwę istniejącemu typowi danych. Deklaracja poprzedzona słowem kluczowym **`typedef`** przekształca się ze zwykłej deklaracji zmiennej w deklarację typu. Identyfikator występujący w takiej deklaracji staje się nazwą użytego typu (a nie nazwą zmiennej tego typu).

```
typedef char ASCII;          // ASCII zastepuje typ char
typedef char znak,           // znak zastepuje typ char
             *wskznak,       // wskznak zastepuje typ char*
             tabznak[80];    // tabznak zastepuje typ char[80]

                    // Faktyczne typy zmiennych:
ASCII a,b;          // a oraz b sa typ char
znak c,d[12];       // c jest typu char
                    // d jest tablica 12 elementow typu char
wskznak p,*q,s[8];  // p jest wskaznikiem na char
                    // q jest wskaznikiem na wskaznik na char
                    // s jest tablica 8 wskaznikow na char
tabznak t,m[20];    // t jest tablica 80 elementow typu char
                    // m jest 20 elementowa tablica tablic
                    //        80 elementow typu char
```

Wykorzystanie deklaracji **typedef** jest szczególnie przydatne przy operowaniu złożonymi składniowo wyrażeniami określającymi typ.

Deklaracja typu strukturalnego:

```
typedef struct
{
    char imie[20];
    char nazwisko[40];
    int rok_ur;
    int mies_ur;
    int dz_ur;
    char miejsce_ur[30];
    char adres[50];
}pracownik;             // zadeklarowano typ pracownik jako
                        //    strukture odpowiednich pol

struct pracownik prac;  // definicja zmiennej strukturalnej
struct pracownik tablprac[120];   // definicja tablicy 120
                        //    elementow typu strukturalnego
```

Tablice struktur

Przykład definicji tablicy struktur:

```
 struct data
{
    int rok;
    int mies;
    int dzien;
} tab[100];

               // albo

struct data tab[100];  // jeśli wcześniej zdefiniowano
                       // strukturę data
```

Zdefiniowana jest 100-elementowa tablica struktur (elementy tej tablicy są zdefiniowanymi wcześniej obiektami strukturalnymi).

Zadanie do wykonania w trakcie zajęć:

Napisz program pozwalający na wczytanie danych (imię i 9-cyfrowy numer telefonu) dla *n* telefonujących osób (zakładamy *n*<=100, *n* podane przez użytkownika), po czym program losuje i wyświetla jednego zwycięzcę spośród wczytanych *n* osób.

W celu rozwiązania zadania:

- zdefiniuj strukturę zawierającą składowe: *imie, nr_telefonu, wygrywa*
- zdefiniuj 100-elementową tablicę tych struktur
- zdefiniuj funkcję służącą do wprowadzenia danych kolejnej osoby,
- zdefiniuj funkcję losującą zwycięzcę,
- zdefiniuj i dopracuj funkcję main().

Zadanie dodatkowe (nadobowiązkowe) dla zainteresowanych (dostarczenie zadania możliwe wyłącznie na kolejnych zajęciach w formie wydruku – tekst źródłowy + okno z wynikami wykonania, wraz z ustnym zaliczeniem)

Napisz program pozwalający na wczytanie danych (imię i 9-cyfrowy numer telefonu) dla *n* telefonujących osób (zakładamy *n*<=100) i wyłonienie zwycięzców w następujący sposób: program losuje liczbę całkowitą z przedziału [0, 99], po czym wygrywają te osoby, których numer telefonu kończy się na wylosowaną liczbę (dwie cyfry); jeśli nikt nie wygrał – losowanie i procedura wyłaniania zwycięzców są powtarzane.

W celu rozwiązania zadania:

- zdefiniuj strukturę zawierającą składowe: *imie, nr_telefonu, wygrywa*
- zdefiniuj tablicę tych struktur
- zdefiniuj funkcję służącą do wprowadzenia danych kolejnej osoby,
- zdefiniuj funkcję losującą dwucyfrową liczbę,
- zdefiniuj funkcję wypełniającą składowe *wygrywa* dla wszystkich osób; funkcja zwraca liczbę wygrywających osób,
- zdefiniuj funkcję drukującą imiona wygrywających osób,
- zdefiniuj i dopracuj funkcję main().