C-struktury – wykład

C-struktury

- W języku C++, jak w każdym języku obiektowym, mamy możliwość definiowania własnych typów danych, wraz z określeniem operacji, jakie na tych danych można wykonywać. Służą do tego klasy, występujące w C++, z powodów głównie historycznych, pod dwoma nazwami: klas i struktur.
- Jednak typy złożone istnieją również w czystym C. Mają one postać unii i struktur. W C++ implementacja struktur została rozszerzona (i jest w zasadzie taka jak dla klas), ale warto zdawać sobie sprawę, jak wyglądają struktury zgodne z implementacją w C. Tego typu struktury, które będziemy nazywać C-strukturami, są bardzo często używane nawet w programach napisanych w C++, głównie ze względu na to, że C-strukturami są często typy zdefiniowane w standardowych bibliotekach.

- ***Struktura*** jest zbiorem zmiennych występujących pod wspólna nazwą.

- Zmienne wchodzące w skład struktury nazywane są ***polami*** lub elementami, a czasem członkami struktury.

- Struktury używamy, jeśli chcemy zgrupować zmienne jako jeden rekord.

C-struktura jest kolekcją nazwanych składowych, które mogą być różnych typów, również innych typów strukturalnych.

DEKLARACJA STRUKTURY

Strukturę definiujemy korzystając ze słowa kluczowego **`struct`**.
Ogólna postać deklaracji struktury jest następująca:

```
struct nazwa_typu   //staje się nazwa nowego typu
{
   [ typ nazwa_pola[,nazwa_pola[ ,...]];]
   [ typ nazwa_pola[,nazwa_pola[ ,...]];]
   ...
} lista_zmiennych_strukturalnych;
```

DEKLARACJA STRUKTURY

UWAGI:

- można albo pominąć `nazwę typu` (struktura anonimowa) albo `listę zmiennych`, ale nie można opuścić obu jednocześnie;

- ponieważ każda deklaracja struktury jest instrukcją, to deklarację struktury zawsze kończymy średnikiem;

- elementy `lista_zmiennych_strukturalnych` są oddzielane przecinkami i mogą być identyfikatorami zmiennych strukturalnych lub mogą identyfikatorami zmiennych z nadawanymi im wartościami początkowymi. Wtedy

```
identyfikator_zmiennej = wartość_początkowa_struktury
```

DEKLARACJA STRUKTURY- przykład

Przykład. Zdefiniujmy strukturę do przechowywania danych samochodu takich jak:

- marka,
- model,
- rok produkcji,
- pojemność silnika:

```
struct Auto
{
  string marka;
  string model;
  int rok_produkcji;
  double pojemnosc;
};
```

W tym przypadku nie mamy żadnej zmiennej. Określony jest typ `Auto` będący strukturą o czterech polach.

Deklarowanie zmiennych typu strukturalnego

Aby zadeklarować inne zmienne strukturalne niż w definicji typu korzystamy z następującej instrukcji:

```
struct nazwatypu listazmiennych;//C i C++
```

lub bez słowa kluczowego `struct`

```
nazwatypu listazmiennych; //tylko w C++
```

Np.

```
int main()
{
    Auto opel;//Auto jest typem, a opel nazwą zmiennej
    return 0;
}
```

W C++ można opuścić słowo **`struct`**.

Dopiero po napotkaniu deklaracji zmiennej kompilator przydziela odpowiedni obszar pamięci .

Deklaracja ze strukturą zagnieżdżoną

Struktury można zagnieżdżać tzn. polem struktury może być inna struktura nie anonimowa (nie może być ta, którą aktualnie definiujemy) – tzn. musi być nazwa pola. Np.:

```
struct Punkt //definicja tylko typu
{
   float x,y;
};


struct Okrag
{
   float promien;
   Punkt srodek;
};
```

Inicjalizacja zmiennych strukturalnych

Inicjalizacja struktury polega na umieszczeniu w nawiasach klamrowych inicjalizacji poszczególnych pól zgodnie z kolejnością ich występowania w definicji typu struktury. Ilość inicjalizacji nie może przekraczać ilości pól. W przypadku mniejszej ilości inicjalizacji niż pól, pozostałe pola liczbowe będą zainicjowane zerami, a pola typu wskaźnikowego adresami pustymi.

Przykłady:

```
//samochód renault
/*ten przykład nie zadziała w Builder 6.0 ze względu na string
  w strukturze (wtedy string zastąpić tablicą char)  */
Auto renault = {"renault","megane",2013,2.0};
//punkt (1,3)
Punkt A = {1.0, 3.0};
//okrąg o środku w punkcie (1,1)i promieniu 5.5
Okrag K = {5.5, {1.0, 1.0}},
//okrąg o środku w punkcie (0,0) i promieniu 1
      K0 = {1.0};
//w C++ 11 można ominąć znak =
//np. Okrag K0{1.0};
```

Dostęp do pól struktury

Dostęp do pól struktury uzyskujemy operatorem kropka.

```
int main()
{
        Auto auto1;
        //pobranie danych  o marce i pojemnosci auta
        cout<<"podaj marke samochodu: ";
        cin>>auto1.marka; //odwołanie się do pola marka
        cout<<"podaj pojemnosc samochodu: ";
        cin>>auto1.pojemnosc; //i pojemność
        cout<<"Marka: "<<auto1.marka
            <<", pojemnosc: "<<auto1.pojemnosc<<endl;
        return 0;
}
```

Dostęp do pól struktury - przykład

Przykład. Program wczyta środek i promień koła i wyświetli informację o tym czy początek układu współrzędnych leży w kole czy na zewnątrz. Następnie sprawdzi, to również dla koła o promieniu o 1 większym.

```
#include <iostream>
using namespace std;

struct Punkt
{
  double x,y;
};

struct Kolo
{
  double r;
  Punkt s;
};
```

Program przykładowy

```
int main()
{
  Kolo k, kw;
  cout<<"Położenie punktu względem koła\n";
  cout<<"podaj środek koła\nx = ";
  cin>>k.s.x;
  cout <<"y = ";
  cin>>k.s.y;
  cout<<"promień: ";
  cin>>k.r;
  cout<<"Początek układu leży ";
  cout<< (k.s.x*k.s.x + k.s.y*k.s.y <= k.r*k.r
        ? "w kole":"poza kołem");
```

Program przykładowy cd.- przypisywanie struktur

```
/*wartość jednej zmiennej typu struktura można
przypisać innej zmiennej tego samego typu za pomocą
pojedynczej instrukcji przypisania "=", powoduje to,
że każde pole struktury do której robimy przypisanie,
otrzymuje wartość odpowiedniego pola struktury
przypisywanej
*/
  kw=k;//przypisanie całej struktury
  (kw.r)++; //zwiększenie o 1 promienia koła kw
  cout<<"\nPoczątek układu leży ";
  cout<< (kw.s.x*kw.s.x + kw.s.y*kw.s.y <= kw.r*kw.r
            ? "w kole":"poza kołem")<<endl;
  return 0;
}//main
```

Struktury jako parametry i wyniki funkcji.

Struktury zachowują się podobnie jak typy wbudowane, można na przykład przekazywać je jako parametry funkcji, można definiować też funkcje zwracające strukturę jako swój wynik.

Użycie struktury jako parametru funkcji jest przekazaniem przez wartość (parametr aktualny musi być tego samego typu).

Uwaga: Oczywiście jeśli chcemy korzystać ze struktur w funkcjach musimy strukturę zdefiniować globalnie (przed `main()`)

Przykład. Zadanie obliczania godziny i minuty po upływie zadanego okresu czasu w godzinach i minutach.

PROGRAM- struktury i funkcje

```
#include <iostream>
#include <iomanip>

using namespace std;

struct Czas
{
  int g,m;//godziny i minuty odpowiednio
} ;


//Napisz funkcję, która zwraca jako wynik czas jaki mamy po
//upływie czasu przekazanego przez drugi parametr od czasu
//startowego danego pierwszym parametrem funkcji.
Czas NowyCzas(Czas teraz, Czas ileDodac)
{  Czas nowy;
   int r = teraz.m + ileDodac.m;
   nowy.m = r%60;
   nowy.g = (teraz.g + ileDodac.g + r/60)%24;
   return nowy;
}
```

PROGRAM- struktury i funkcje

```
/* Napisz funkcję, która zwiększa czas będący
pierwszym parametrem funkcji o czas dany przez drugi
parametr. Otrzymany czas funkcja przekazuje w
pierwszym parametrze. Funkcja nie zwraca żadnego
wyniku.
*/
void NowyCzas1(Czas &teraz, Czas ileDodac)
{
   int r = teraz.m + ileDodac.m;
   teraz.m = r % 60;
   teraz.g = (teraz.g + ileDodac.g + r/60)%24;
}
```

PROGRAM- struktury i funkcje cd.

```
int main()
{
  Czas cz={8,45}, w, d={4,20};

  cout<<cz.g<<":"<<setw(2)<<setfill('0')<<cz.m<<endl;

  w = NowyCzas(cz,d);

  cout<<w.g<<":"<<setw(2)<<w.m<<endl;

  NowyCzas1(cz,d);

  cout<<cz.g<<":"<<setw(2)<<cz.m<<endl<<setfill(' ');

  return 0;
}
```

Tablice struktur

Tablice struktur definiujemy analogicznie jak inne tablice.

Przykłady definiowania tablic struktur:

```
struct Punkt{double x,y; } ;

Punkt tab[100];//tablica 100 punktów

Punkt tab2[2]={{1,1},{-3,4}};
```

oraz korzystania z elementów tablic:

```
tab[1] = tab2[1];// struktura przypisana strukturze

tab[1].x = tab2[0].x + 11;
```

Tablice struktur – przykład

Dla danej struktury

```
struct Auto
{
  string marka;
  string model;
  int rok_produkcji;
  double pojemnosc;
};
```

napisz funkcję `wczytajAuto` typu `void`, która pobierze z klawiatury informacje o samochodzie i zwróci je jako parametr typu `Auto`.

```
void wczytajAuto(Auto &sam)
{
  cout<<"marka: "; cin>>sam.marka;
  cout<<"model: "; cin>>sam.model;
  cout<<"rok produkcji: "; cin>>sam.rok_produkcji;
  cout<<"pojemnosc: "; cin>>sam.pojemnosc;
}
```

Tablice struktur – przykład

Wykorzystując strukturę `Auto` z poprzedniego zadania oraz strukturę

```
struct Komis
{
        int ilsam; //liczba samochodow
        Auto tabsam[100];
};
```

A) napisz funkcję która wyświetla dane komisu

```
void wyswietlKomis(Komis &k)//& by nie kopiować dużej struktury
{
   if(k.ilsam>0)
      for (int i=0;i<k.ilsam;i++)
          cout<<k.tabsam[i].marka<<", "
              <<k.tabsam[i].model<<", "
              <<k.tabsam[i].rok_produkcji<<", "
              <<k.tabsam[i].pojemnosc<<endl;
   else cout<<"brak aut w komisie"<<endl;
}
```

Uwaga: Gdy funkcja nie modyfikuje wartości struktury lepiej strukturę przekazać do funkcji przez stałą referencję:

```
void wyswietlKomis(const Komis &k)
```

Tablice struktur – przykład c.d.

B) napisz dwuparametrową funkcję

```
int ileAut(Komis &k, int granica)
```

zwracającą liczbę całkowitą równą liczbie samochodów w komisie `k` o roku produkcji większym od wartości drugiego parametru `granica`.

```
int ileAut(Komis &k, int granica)
{
    int ile=0; //zerujemy licznik
    //przeglądamy wszystkie auta w komisie
    for (int i=0;i<k.ilsam;i++)
        if (k.tabsam[i].rok_produkcji > granica)
            ile++;
    return ile;
}
```

W tym przypadku również mogliśmy deklarację funkcji uzupełnić słowem `const`:

```
int ileAut(const Komis &k, int granica)
```

Tablice struktur – przykład c.d.

C) napisz funkcję o nagłówku

```
int najstarszeAuto(Komis & k);
```

która dla przekazanego jako parametr komisu `k` wyznaczy indeks najstarszego samochodu w komisie (czyli o najmniejszym roku produkcji) lub `-1` w przypadku braku samochodów w komisie. Jeśli w komisie jest kilka aut o tym samym najmniejszym roku produkcji, funkcja ma zwracać indeks pierwszego z nich.

```
int najstarszeAuto(Komis & k)
{
    if (k.ilsam <= 0) return -1;
    //na początku przyjmujemy, ze 0-we auto ma min rok prod
    int indMin=0;
    int wiekMin=k.tabsam[0].rok_produkcji;
    //przeglądamy pozostałe auta w komisie
    for (int i=1;i<k.ilsam;i++)
        //jeśli spotkamy starsze
        if(k.tabsam[i].rok_produkcji < wiekMin)
        {   //zapamiętujemy jego indeks i rok produkcji
            indMin=i;
            wiekMin=k.tabsam[i].rok_produkcji;
        }
    return indMin;
}
```

Tablice struktur – przykład c.d.

```
int main()
{
    Komis autokomis = {3,{{"opel","zafira",2015,2.0},
                          {"citroen","c3",2012,1.7},
                          {"kia","sportage",2005,1.9}} };
    //ta inicjalizacja nie zadziała w C++ Builder 6.0
    wyswietlKomis(autokomis);
    int ind = najstarszeAuto(autokomis);
    if(ind >= 0)
       cout<<"indeks najstarszego auta "<<ind<<endl;
    else
       cout<<"nie ma aut w komisie"<<endl;
    cout<<"w komisie mamy "<<ileAut(autokomis,2010)
        <<" aut wyprodukowanych po 2010 roku";
    return 0;
}
```

Tablice struktur – przykład c.d.

Wyniki działania powyższego programu:

```
opel, zafira, 2015, 2
citroen, c3, 2012, 1.7
kia, sportage, 2005, 1.9
indeks najstarszego auta 2
w komisie mamy 2 aut wyprodukowanych po 2010 roku
```