Štruktúry

Úvod do štruktúr

Predpokladajme, že chcete uložiť meno, vek a BMI index niekoľkých osôb. Pole, môže obsahovať niekoľko položiek, ale to neurobí, pretože musí byť rovnakého typu. Odpoveďou je štruktúra v C++. Ak chceme vytvoriť zoznam všetkých zamestnancov, vytvoríme pole štruktúr. Štruktúra vedie do tried.

Štruktúry vytvárame najčastejšie mimo main. Nazýva sa externou deklaráciou štruktúry.

Deklarácia štruktúry:

```
struct lide
{
   char meno[10];
   int vek;
   double bmi;
};
```

Inicializácia štruktúry

(za názov_premennej sme si zvolili lide):

```
lide ales =
{
   "Ales",     //jmeno
   22,          // vek
   11.49      // bmi index
};
```

Volanie niektorých premenných zo štruktúry

Ak chceme vyvolať vek alese, napíšeme:

```
cout << ales.vek;
```

To vypíše 22, čiže vek Aleše.

. je operátor príslušnosti. (ales.vek – premenná vek v štruktúre ales)

Vytvoriť štruktúr si môžeme, koľko chceme, kľudne celý počet zamestnancov. Napr.:

```
lide petr =
{ ... }
```

…atd

Alebo si môžeme vytvoriť prázdne štruktúry:

```
lide martin, honza, alena;
```

Vytvorili sme tri štruktúry, každá má svoje premenné ako ales, ale nemajú žiadne hodnoty. Ak chceme napríklad zapísať vek do štruktúry alena, napíšeme:

```
alena.vek = 17;
```

Je to ľahké, ako premenné, len tu je naviac bodka, ktorá nám ukazuje, kde daná premenná patrí.

Vytvoriť štruktúry môžeme už pri ich deklarácii:

```
struct lide
{
   int vek;
} milan;
```

Je to rovnaké ako:

```
struct lide
{
   int vek;
};
lide milan;
```

Kto presne stále nechápe štruktúry, malé vysvetlenie:

```
struct osoba
{
char jmeno[20];
int vek[20];
char prace[20];
} zamestnanec;
```

Prečo nefunguje napr. *cout << osoba.jmeno;*, ale *cout << zamestnanec.jmeno;*? Čo značí „osoba“ a čo „zamestnanec“?

Pretože:

```
struct osoba
{
char jmeno[20];
int vek[20];
char prace[20];
};
```

…je len DEFINÍCIA, nemá žiadne hodnoty.

Príklad použitia:

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

struct rostliny
{
char jmeno[20];
float vaha;
double cena;
};

int main()
{
rostliny venkovni =
{
"slunecnice",
0.20,
12.49
};
rostliny vyber;
cout << "kytice: " << venkovni.jmeno << " za ";
cout << venkovni.cena << " Kc\n";
vyber = venkovni; // přiřadí jednu strukturu druhé
cout << "vyber: " << vyber.jmeno << " za ";
cout << vyber.cena << " Kc\n";
return 0;
}
```

Tu sme si ukázali i kopírovanie jednej štruktúry do druhej.

Pole struktur

Je možné vytvoriť i pole, ktorého prvky sú štruktúry:

```
rostliny vnitrni[100];    // pole o 100 strukturách rostliny
vnitrni[0].cena;     // pouzije promennou cena ze struktury vnitrni[0]
```

Bitové polia

C++ vám umožňuje špecifikovať štruktúrne členy, ktoré zaujímajú určitý počet bitov. To mažete použiť napr. na vytvorenie dátových štruktúr, ktoré odpovedajú registru nejakého hardwarového zariadenia.

```
struct vypocet
{
   int key : 4;                // 4 bity pro hodnotu key
   bool otazka : 1;        // platný vstup (1 bit)
};
```

Pre prístup k bitovým poliam použite rovnakú syntax ako u štruktúr.

vypocet.key = ...

Bitové polia sa často používajú na úrovni strojovo orientovaného programovania.

Uniony

Union je dátovým formátom, ktorý môže používať rôzne dátové typy, len jeden typ v danom okamihu. Zatiaľ čo štruktúra môže používať typy int, long a float, union môže byť buď int, alebo long, alebo float. Syntax je podobná ako u štruktúr, len význam je rôzny.

```
union zkouska
{
  int hodn1;
  long hodn2;
  double hodn3;
};
```

Premennú zkouska môžete využiť na uchovanie int, long, double len tak dlho, dokiaľ potrebujete:

```
zkouska stetec;
stetec.hodn1 = 15;            // uloží int
cout << stetec.hodn1;
stetec.hodn2 = 1.25;        // uloží double, hodnota int se ztratí
cout << stetec.hodn2;
```

Pretože v jednom okamžiku premenná obsahuje len jednu hodnotu, musí mať dostatok priestoru na úschovu svojho najväčšieho člena. Z toho dôvodu je veľkost unionu veľkosťou najväčšieho člena. Kde sa používa? Úspora priestoru…