Základy programování v C++ ... 10. cvičení

Zuzana Petříčková

5. listopadu 2018

Přehled

1. Neobjektové datové typy definované uživatelem
   - Zavedení nového jména pro existující typ
   - Výčtový typ
   - Struktura

Neobjektové datové typy definované uživatelem

Zavedení nového jména pro existující typ

```
typedef typ nazev; // starsi zpusob
using nazev = typ; // novejsi zpusob od C++11
```

- podobný význam jako deklarace pojmenované konstanty → zpřehlednění programu, později stačí změnit typ na jednom místě (např. na požádání změnit přesnost z **int** na **long long**)

```
typedef int cislo;
// using cislo = int;
typedef cislo male_pole[10];
typedef unsigned int UInt;
typedef const UInt CUInt;
...
cislo x = 7; // deklarace promenne
...
cislo minimum(cislo a, cislo b)
{
    return (a < b) ? a : b;
}
```

Odbočka: podmíněný výraz

- syntaxe: **E ? E1 : E2**
- je-li hodnota výrazu **E** rovna **true**, je výsledkem podmíněného výrazu **E1**, jinak **E2**.

Příklad

```
double x, y, m;
cin >> x >> y;
...
x = (x >= 0)  ? x : -x; // absolutni hodnota
m = (x >  y)  ? x : y;  // maximum dvou cisel
```

- Napište a zavolejte funkci, která pomocí podmíněného výrazu spočte hodnotu (pro $a < b, a, b \in R$):

$$f(x) = \begin{cases} a, & x < a \\ x, & a \leq x \leq b \\ b, & x > b \end{cases}$$

Odbočka: podmíněný výraz

Řešení příkladu

- Funkce, která pomocí podmíněného výrazu spočte hodnotu (pro $a < b, a, b \in R$):

$$f(x) = \begin{cases} a, & x < a \\ x, & a \leq x \leq b \\ b, & x > b \end{cases}$$

```
double funkce(double x, double a, double b)
{
    return (x < a) ? a : (x > b) ? b : x;
}
```

Výčtový typ

- usnadňuje práci s veličinami, které mohou mít malou množinu hodnot (dny v týdnu, barvy, chyby programu,...)

Deklarace

```
enum nazev {seznam_polozek};
enum nazev {seznam_polozek} seznam_deklaratoru;
```

- **nazev** je název (identifikátor) nového typu
- položky jsou identifikátory (popř. doplněné o hodnotu, kterou představují)
- **seznam_deklaratoru** je seznam proměnných daného typu

```
enum Vsedni_den {po, ut, st, ct, pa} dnes, zitra;
enum Svetova_strana {sever, jih, vychod, zapad};
...
Svetova_strana x = sever;
```

Výčtový typ

```
enum Vsedni_den {po, ut, st, ct, pa} dnes, zitra;
enum Svetova_strana {sever, jih, vychod, zapad};
```

- položky jsou reprezentovány celými čísli (počínaje od 0, hodnota každé položky je vždy o 1 větší než je hodnota předchozí)
- hodnoty položek lze předefinovat konstantním výrazem:

```
enum Vsedni_den {po, ut, st, ct=8, pa} dnes, zitra;
enum Svetova_strana {sever=1, jih, vychod=4, zapad=8};
```

- **cout** nevypíše jméno položky, ale odpovídající celočíselnou hodnotu.

Výčtový typ

- jedná se o celočíselný datový typ a lze jej použít v podmínce příkazu **switch**:

```
Svetova_strana tam;
...
switch (tam)
{
case sever:
    cout << "Jdu na sever";
    break;
case jih:
    cout << "Jdu na jih";
    break;
    ...
}
```

Výčtový typ

Příklady

- Definujte výčtové typy **Den** (pro 7 dní v týdnu) a **Svetova_strana** (pro 4 světové strany).
- Napište a zavolejte funkci **bool vikend(Den den)**, která vrátí **true**, pokud je **den** sobota nebo nedele.
- Napište a zavolejte funkci **bool povoleny_smer(Svetova_strana tam)**, která vrátí **true**, pokud je hodnota **tam** sever nebo jih.
  (lze řešit s využitím bitových operací & a |)
- Napište a zavolejte funkci **void vypis_den(Den den)**, která vypíše na konzoli, jaký je **den**, např. "Dnes je streda".

Výčtový typ - řešení příkladů

```
enum Svetova_strana {sever=1,jih=2,vychod=4,zapad=8};
enum Den {pondeli,utery,streda,ctvrtek,patek,sobota,nedele };
const int dovoleny_smer = (sever | jih);

void vypis_den(Den den)
{
    const static string jmena[] = {"pondeli","utery",
                "streda","ctvrtek","patek","sobota","nedele"};
    if (den < 7 && den >= 0)
        cout << "Dnes je " << jmena[den] << endl;
}
bool vikend(Den den)
{
    return (den == sobota || den == nedele);
}
bool povoleny_smer(Svetova_strana tam)
{
    return ((int)tam & dovoleny_smer);
}
```

Struktura

- skupina proměnných různých typů, se kterou se pracuje jako s celkem

Deklarace

```
struct identifikator
{
    deklarace_slozek
};
struct identifikator
{
    deklarace_slozek
} seznam_deklaratoru;
```

- **identifikator** je název (identifikátor) struktury
- složka může být proměnná libovolného (jiného) typu
- **seznam_deklaratoru** je seznam proměnných daného typu

Struktura

```
struct Datum
{
    unsigned short den;
    unsigned short mesic;
    int rok;
};

struct Komplexni_cislo
{
    double re;
    double im;
};
```

- je konvence umísťovat definice struktur (a definice datových typů obecně) na začátek zdrojového souboru (nad definice funkcí), popřípadě do **hlavičkového souboru**

Struktura

Deklarace proměnné a inicializace

```
Datum datum; // bez inicializace
Datum narozen = {1,1,1998};
Datum dnes = {29,10,}; // rok bude 0
Datum terminy_zkousek[10]; // staticke pole
```

Struktura

Přístup ke složkám struktury

- pomocí operátoru **.** (tečka)

```
Datum dnes = {29,10,2018};

Datum datum;
datum.den = 28;
datum.mesic = 10;
datum.rok = 1918;

int uplynulo_let = dnes.rok - datum.rok;

Datum vcera = dnes;
vcera.den--;
```

Struktura

Kopírování struktury

- při přiřazení se kopírují všechny složky struktury včetně obsahu statických polí
- pokud je složkou statické pole, nemusím ho kopírovat prvek po prvku

```
struct Predmet
{
    string nazev;
    Datum terminy_zkousek[5];
    ...
};
...
Predmet matalyza = {"Matematicka analyza", {{1,2,2019},{10,2
Predmet lingebra = matalyza;
/* netreba:
for (int i = 0; i < 5; i++)
    lingebra.terminy_zkousek[i] = matalyza.terminy_zkousek[
*/
```

Struktury - příklady

- Definujte výčtový typ **Pozice** (pro několik pracovních pozic) a struktury **Datum** (se složkami den, mesic a rok) a **Zamestnanec** se složkami jmeno (string), prijmeni (string), nastup (Datum), plat (int) a pozice (Pozice).
- Deklarujte lokální proměnné typu Zamestnanec a naplňte je hodnotami.
- Deklarujte pole zaměstnanců **pobocka_Praha** a naplňte ho hodnotami.
- Napište uživatelsky přívětivou funkci **void vypis_zamestnance(Zamestnanec pobocka[], int n)**, která za využití FOR-cyklu vypíše obsah pole **pobocka** délky **n** na konzoli.
- (na procvičení) Napište uživatelsky přívětivou funkci **void nacti_zamestnance(Zamestnanec pobocka[], int n)**, která za využití FOR-cyklu načtěte obsah pole **pobocka** z konzole.

Struktury - část řešení příkladů

```
struct Datum
{
        unsigned short den;
        unsigned short mesic;
        int rok;
};
enum Pozice  {vratny, programator, ucetni};
const string nazev_pozice[] = {"vratny",
                "programator", "ucetni"};
struct Zamestnanec
{
        string jmeno;
        string prijmeni;
        Datum nastup;
        int plat;
        Pozice pozice;
};
```

Struktury - část řešení příkladů

```
void vypis_zamestnance(Zamestnanec pobocka[], int n)
{
    for (int i = 0; i<n; i++)
    {
        cout << "Jmeno: " << pobocka[i].jmeno <<endl;
        cout << "Prijmeni: " << pobocka[i].prijmeni <<endl;
        cout <<"Pozice: " << nazev_pozice[pobocka[i].pozice]
        cout <<"Plat: " << pobocka[i].plat << endl;
        cout <<"Nastup: " << pobocka[i].nastup.den << "."
                << pobocka[i].nastup.mesic << "."
                << pobocka[i].nastup.rok  << endl << endl;
    }
}
```

Struktury - další příklady na procvičení

- Definujte strukturu **Zlomek** s celočíselnými složkami **citatel** a **jmenovatel**.
- Napište funkce **Zlomek secti(Zlomek x, Zlomek y**, **Zlomek odecti(Zlomek x, Zlomek y**), **Zlomek vynasob(Zlomek x, Zlomek y**), **Zlomek vydel(Zlomek x, Zlomek y**) pro základní operace se zlomky. Funkce převedou výsledek do základního tvaru (vydělením čitatele i jmenovatele jejich největším společným dělitelem a úpravou znaménka).
  Příklad: součet 2/3 a -1/6 je 1/2
- **Pomocné funkce**
  - Napište funkci **int nsd(int x, int y)**, která spočítá největšího společného dělitele čísel x a y.
  - Napište funkci **Zlomek zakladni_tvar(Zlomek z)**, která převede zlomek do základního tvaru (a opraví znaménko minus).
  - Napište funkci **void vypis_zlomek(Zlomek x)**, která vypíše zlomek na konzoli.

Struktury - další příklady na procvičení

Euklidův algoritmus pro výpočet největšího společného dělitele (pro $x, y > 0$)

$$nsd(x,y) = \begin{cases} x, & x == y \\ nsd(x-y,y), & x > y \\ nsd(x,y-x), & x < y \end{cases}$$

Lépe: parametry typu struktura předávat funkcím jako (konstantní) reference (označené pomocí **&**)

```
Zlomek secti(const Zlomek& x,const Zlomek& y};
Zlomek odecti(const Zlomek& x,const Zlomek& y};
Zlomek vynasob(const Zlomek& x,const Zlomek& y};
Zlomek vydel(const Zlomek& x,const Zlomek& y};
int nsd(int x, int y)
void zakladni_tvar(Zlomek& z)
void vypis_zlomek(const Zlomek& x)
```

Pro struktury mohu definovat (přetížit) i operátory

klíčové slovo **operator**:

```
...
Zlomek operator + (const Zlomek& x,const Zlomek& y)
{
  return secti(x, y);
}
Zlomek operator - (const Zlomek& x,const Zlomek& y)
{
  return odecti(x, y);
}
int main()
{
  Zlomek x= {2, 3}, y ={-2, 12}, z;
  z = x + y;
  ...
}
```