6. Statické proměnné a metody, šablony v C++

Globální konstantní proměnné

Konstantní proměnné specifikujeme s klíčovým slovem **`const`**, tyto konstantní proměné musí mít hodnotu inicializovanou při definici, jejich hodnotu nelze v průběhu programu měnit.

Konstantní proměnné mohou být lokální, globální nebo členy třídy.

V C++ používáme globální konstantní proměnné namísto symbolických konstant definovaných direktivou `#define` používaných v jazyce C.

Globální konstantní proměnné lze využívat např. pro specifikaci velikosti polí.

Názvy konstantních proměnných často píšeme velkými písmeny.

```
const int MAX_ITEMS = 1000;

int main()
{
  int itemsArray[MAX_ITEMS];
    // Zde bude libovolny dalsi kod
  return 0;
}
```

Statické členy třídy

Statické členy třídy deklarujeme s klíčovým slovem `static`.

Statické proměnné se chovají podobně jako globální proměnné, program pro ně alokuje paměť ihned po spuštění programu, při vytváření objektových proměnných se již pro ně žádná další paměť nealokuje.

Statické proměnné tedy můžeme použít aniž bychom vytvořili příslušný objekt (tj. definovali objektovou proměnnou).

Ke statickým proměnným přistupujeme přes jméno třídy a **::** (`Jmeno_tridy::jmeno_promenne`), pokud však k nim přistupujeme z metod téže třídy, stačí použít přímo jméno proměnné.

Statické proměnné inicializujeme v samostatné definici mimo třídu (narozdíl od nestatických proměnných, které musíme inicializovat v konstruktoru).

Statické proměnné často definujeme v sekci **`public`** aby byly přístupné i z funkcí a metod jiných tříd (pokud však toto nepotřebujeme, mohou být `protected` nebo `private`).

```
class Vector3D
{
  public:
    // Nasledujici staticka promenna bude
    // obsahovat celkovy pocet aktualne
    // existujicich promennych typu Vect3d
    static int totalVectCount;
    Vector3D();  // Konstruktor
    ~Vector3D(); // Destruktor
};

// Inicializace staticke promenne:
int Vector3D::totalVectCount = 0;

Vector3D::Vector3D()
{
  // Pri vytvoreni (definici) promenne
  // zvysime hodnotu o 1
  totalVectCount++;
}

Vector3D::~Vector3D()
{
  // Pri zruseni promenne snizime
  // hodnotu o 1
  totalVectCount--;
}

int main()
{
  //Vola se trikrat konstruktor Vector3D()
  Vector3D v1, v2, v3;

  // Pokud volame staticke promenne mimo
  // metody tridy, musime pred jmeno
  // promenne uvest jemno tridy oddelene
  // dvema dvojteckama
  cout << "Celkovy pocet vektoru"
       << Vector3D::totalVectCount
       << endl;
  // Vypise se hodnota 3 protoze jsou
  // aktualne definovane tri promenne
  // typu Vector3D
  return 0;
}
```

Konstantní statické členy třídy

Statické proměnné můžeme zároveň definovat jako konstantní tam kde je to vhodné. Pokud jsou typu `int`, můžeme je inicializovat přímo ve třídě.

```
class Graphic
{
  public:
    // Nasledujici konstantni staticka
    // promenna bude pouzita pro cislo
    // modre barvy
    static const int COLOR_BLUE = 3;
    void draw(int dev);
};

void Graphic::draw(int dev)
{
  g2_pen(dev, COLOR_BLUE);
  // Zde muze byt dalsi kod metody
}

int main()
{
  Graphic g1;
  cout << "Modra barva ma cislo: "
       << Graphic::COLOR_BLUE << endl;
  return 0;
}
```

Statické metody

Statické metody deklarujeme s klíčovým slovem **`static`**.

Klíčové slovo **`static`** uvádíme pouze v deklaraci ve třídě, v definici mimo třídu již ne.

Statické metody se chovají se podobně jako nečlenské funkce, tj. můžeme je volat aniž bychom vytvořili objekt (tj. definovali objektovou proměnnou).

Ke statickým metodám přistupujeme přes jméno třídy a **`::`** (`Jmeno_tridy::jmeno_metody()`), pokud je však voláme z metod třídy, stačí použít přímo jméno metody.

Statické metody mohou operovat pouze nad statickými daty dané třídy.

Výhodou statické metody oproti nečlenským funkcím je vyšší přehlednost programu a zejména zabránění kolizím v názvech (několik tříd může mít statickou metodu se stejným názvem).

```
cclass Graphic
{
  public:
    static int getColorNumberFromString(
           const string &colorName);
    // Definice dalsich clenu tridy
};

  // Tady jiz slovo static neuvadime
int Graphic::getColorNumberFromString(
            const string &colorName)
{
  if (colorName == "blue")
    return 3;
  return 0;  // Pokud je nazev barvy
             // neznamy, vrati 0
}


int main()
{
  // Metodu
  // Graphic::getColorNumberFromString()
  // muzeme volat aniz bychom definovali
  // promennou tridy Graphic
  cout << "Cislo modre barvy: " <<
 Graphic::getColorNumberFromString("blue")
       << endl;
  return 0;
}
```

Šablony funkcí

Šablony funkcí používáme tam kde potřebujeme použít stejnou funkci pro odlišné typy argumentů a návratových hodnot.

Šablonu definujeme podobně jako funkci ale před hlavičkou funkce uvedeme `template <typename T>` nebo `template <class T>`, kde T je volitelný symbol zastupující jméno typu.

Typů můžeme specifikovat i více: `template <typename T1, typename T2, typename T3>`.

V okamžiku volání jména šablony posoudí překladač typ předávaných argumentů a vygeneruje příslušnou funkci.

Šablony funkcí se v praxi používají jen v malé míře, ve standardní knihovně jsou však definovány některé užitečné šablonové funkce (např. `swap()`, `min()`, `max()`).

Šablony lze definovat také pro metody (podobným způsobem jako pro funkce).

```
// Sablona funkce pro ziskani maximalni
// hodnoty ze dvou hodnot
template <typename T>
T getMax(T value1, T value2)
{
  if (value1 > value2)
    return value1;
  else
    return value2;
}

int main()
{
  int i1 = 3, i2 = 5, imax = 0;
  double d1 = 2.12, d2 = 6.78, dmax = 0.0;

  // Na zaklade sablony vygeneruje funkci
  // pro int
  imax = getMax(i1, i2);

  // Na zaklade sablony vygeneruje funkci
  // pro double
  dmax = getMax(d1, d2);
  return 0;
}
```

```
// Sablona funkce pro zamenu dvou hodnot
template <typename T>
void swapItems(T &item1, T &item2)
{
  T itemAux;
  itemAux = item1;
  item1 = item2;
  item2 = itemAux;
}

int main()
{
  int i1 = 3, i2 = 5;
  string s1 = "Prvni retezec";
  string s2 = "Druhy retezec";
  Vector3D v1(3.1, 4.7, -9.0);
  Vector3D v2(6.4, -1.2, 7.8);

  // Vygeneruje se funkce pro typ int
  swapItems(i1, i2);
  // Vygeneruje se funkce pro typ string
  swapItems(s1, s2);
  // Vygeneruje se funkce pro typ Vector3D
  swapItems(v1, v2);
  return 0;
}
```

Šablony tříd

Šablony tříd fungují podobným způsobem jako šablony funkcí, ale na jejich základě se generují celé třídy.

Šablony tříd se v praxi používají jen v malé míře, standardní knihovna však šablony tříd hojně využívá.

Příslušné objektové proměnné deklarujeme:
`jmeno_sablony<jmeno_typu> jmeno_promenne`

```
// Sablona tridy pro 3D vektor
template <typename T> class Vector3D
{
  public:
    Vector3D();
    Vector3D(T ax, T ay, T az);
    void printValues();
  private:
    T x, y, z;
};

template<typename T> Vector3D<T>::Vector3D()
{
  x = 0; y = 0; z = 0;
}

template<typename T>
Vector3D<T>::Vector3D(T ax, T ay, T az)
{
  x = ax; y = ay; z = az;
}

template<typename T>
void Vector3D<T>::printValues()
{
  cout << "Souradnice vektoru:"
       << x << y << z << endl;
}

int main()
{
  Vector3D<int> vectInt;
  Vector3D<double> vectDouble;

  vectInt.printValues();
  vectDouble.printValues();
  return 0;
}
```

STL – standardní šablonová knihovna

Součástí standardní knihovny jazyka C++ je standardní šablonová knihovna STL (standard template library).

STL obsahuje například šablony funkcí implementující některé algoritmy nad souborem dat (třídění, vyhledávání, kopírování).

Z šablonových tříd jsou nejpoužívanější kontejnery (zásobníky), které slouží k ukládání objektů libovolného typu.

Nejpoužívanějším kontejnerem je šablona `vector`, která je zobecněním polí, narozdíl od klasických polí není počet prvků v kontejnerech `vector` fixní, ale zvyšuje se podle počtu vložených prvků (velikost je limitována pouze dostupnou pamětí).

Podrobnější dokumentace k STL na http://www.cppreference.com/wiki/

Kontejner `vector`

Pro použití kontejneru `vector`, musíme vložit hlavičkový soubor `#include <vector>` a deklarovat `using namespace std;`

Kontejner `vector` obsahuje následující metody:

`operator[]` - vrací příslušný prvek podobně jako pole
`front()` - vrací první prvek
`back()` - vrací poslední prvek
`push_back()` - vloží prvek na konec (resp. jeho kopii)
`pop_back()` - vyjímá poslední prvek (nevrací nic)
`clear()` - vyjímá všechnt prvky (tj. vyprázdní kontejner)
`insert()` – vkládá na prvek na specifikovanou pozici
`size()` - vrací aktuální počet prvků
`empty()` - vrací informaci o tom, zda je kontejner prázdný (totéž jako `size() == 0`)

Operátor [] a metody `front()` a `back()` vracejí prvky formou referencí a nekontrolují meze (při překročení mezí pole je jejich chování nedefinované).

Podrobnější výčet všech metod kontejneru `vector` lze najít na: http://www.cplusplus.com/reference/stl/vector/

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

class Circle
{
  // Tady by byly data a metody tridy
  // Circle – viz drive
};

int main()
{
  Circle circle;
  vector<Circle> circlesContainer;

  // Nactou se data do promenne circle
  circle.readFile(sstream);

  // Kopie circle se vlozi na konec
  circlesContainer.push_back(circle);

  // Cyklus pres prvky kontejneru
  for (int i=0;
       i < circlesContainer.size(); i++)
  {
    circlesContainer[i].draw(dev);
  }
}
```

Dodržujte následující pravidla

- Definujte jako konstantní všechny metody, které nemění data třídy (to platí i pro šablony třídy)

Úloha 1 — 1 bod

Vytvořte program vycházející z programu z úlohy 1 z minulého cvičení s následujícími úpravami:

- Program bude používat globální konstantní proměnné pro specifikaci meze polí (tj. např. `const int MAX_GRAPHIC = 100;`)
- Ve třídě `Graphic` vytvořte statické konstantní proměnné pro čísla barev (pro barvy: 0 white, 1 black, 3 blue, 7 green, 19 red, 25 yellow)
- Metodu `getColorNumberFromString()` ve třídě `Graphic` předělejte na statickou metodu a upravte ji tak aby pro čísla barev využívala hodnoty konstantních proměnných vytvořených v předchozím kroku.
- V programu dále vytvořte šablonu pro funkci která v cyklu vykreslí grafické objekty daného typu. Bude přijímat tři argumenty: pole grafických prvků, počet platných prvků v poli a číslo grafického zařízení. Tuto šablonu využijte pro vykreslení grafických objektů.

Program otestujte se souborem *graphic2.dat* (v adresáři */home/martinp/C3220/data/*).

Nápověda:

Čísla barev definujte v sekci `public` a ihned ve třídě inicializujte (např.: `static const int COLOR_RED = 19;`).

Statickou metodu `getColorNumberFromString()` implementujte jak je uvedeno v příkladu v sekci "Statické metody", nevracejte včak čísla, ale hodnoty konstantních proměnných vytvořených v předchozím kroku (`COLOR_RED`, ...).

Šablona funkce pro vykreslování `template <class T> void drawItems(int dev, T items[], int count)` bude obsahovat cuklus od `0` po `count`, v každém kroku cyklu se zavolá metoda `items[i].draw(dev);`

Úloha 2 — 3 body

Modifikujte program následujícím způsobem:

- Vytvořte šablonu třídy (s názvem např. `Container`) která bude obsahovat pole objektů a počet platných prvků v poli a dále následující metody:
  - `Container()` - konstruktor bez parametrů
  - `int add(const T &item)` - přidá prvek (který přijme jako argument) na konec pole vrátí pořadí prvku v poli
  - `T get(int pos) const` - vrátí prvek na dané pozici (pozici prvku přijme jako argument)
  - `T &operator[](int pos)` - totéž jako `get()` ale prvek vrátí referencí
  - `int count() const` - vrátí počet prvků pole
- Program upravte tak aby pro uložení grafických objektů používal výše uvedenou šablonu `Container` namísto obyčejných polí.

Pozn.: pro vykreslování objektů nyní nepoužívejte šablonovou funkci z předchozí úlohy ale vykreslujte podobně jako jako v podobně jako v původní verzi programu. Pro správné fungování programu je potřeba, aby metoda ve které se provádí vykreslování (`Drawing::draw()`) nebyla konstantní, protože se z ní volá nekonstantní operátor `[]` šablony `Container`.

Nápověda:

Vytvořte třídu `template <class T> class Container` a v ní definujte soukromé proměnné `T itemsArray[MAX_GRAPHIC];` a `int itemsCount;` V konstruktoru inicializujte `itemsCount` hodnotou `0`. V metodě `add()` zkopírujte předávanou hodnotu na poslední pozici (`itemsArray[itemsCount] = item;`) pole a zvětšete `itemsCount` o 1.

Ve dřídě `Drawing` defunujte příslušnou proměnnou, která nahradí původní pole križnic, obdélníků a vyplnných kružnic (v `Drawing` místo nich budou `Container<Circle> circlesContainer;` `Container<Rectangle> rectanglessContainer;` `Container<FilledCircle> filledCirclesContainer;`). Ve funkci pro načítání souboru defunujte pomocnou proměnnou `Circle circle;` (podobně pro obdélník a vyplněnou kružnici). Načítání souboru je pak potřeba upravit tak, že se vždy nejdříve načtou data do pomocné proměnné (`circle.readFile(sstream);`), pak se přidají do kontejneru (`circlesContainer.add(circle);`).

Úloha 3 — 1 bod

Program modifikujte tak, že místo výše uvedené šablony `Container` bude používat kontejner `vector` ze standardní knihovny.