Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 17.05. 2001
Url: http://www.builder.cz/art/cpp/namespace.html

Prostory jmen v C++

Prostory jmen v C++

Prostor jmen je oblast platnosti identifikátorů. Představme si situaci, kdy budeme chtít mít v jednom programu stejné identifikátory (názvy tříd, proměnných, metod, funkcí atd...). Nelze mít v jednom programu dvě různé třídy se stejným názvem. Překladači by se nelíbil konflikt jmen. Přesto taková situace může lehce nastat. Může třeba více programátorů, aniž by se domluvili na názvech identifikátorů, pracovat na jednom programu. To by byla zrovna velmi trapná situace, kdyby nebyli schopni se domluvit na názvech identifikátorů v programu. Může ale velmi často nastat situace, kdy je potřeba pracovat s více knihovnami, jejichž autoři se nejspíš nikdy neviděli, a nejspíše o sobě ani nevědí. Také může nastat situace, kdy píšeme svůj program, ve kterém používáme nějakou knihovnu. I zde může nastat konflikt našeho identifikátoru s jiným identifikátorem. Právě tyto konflikty jmen řeší tak zvané jmenné prostory.

Jmenný prostor je oblast, ve které nesmí dojít ke konfliktu jmen. Jméno identifikátoru v prostoru jmen ale může být v konfliktu s jakýmkoliv jménem identifikátoru mimo tento prostor jmen. K definici prostoru jmen slouží klíčové slovo `namespace`. Za `namespace` následuje jméno prostoru jmen a v závorkách `{}` se nachází ona oblast platnosti identifikátorů. Jako příklad nyní vytvořím prostor jmen "RadimuvProstor" a vněm deklaruji a definuji třídu a funkci:

```
#include<iostream>
namespace RadimuvProstor
{
  int secti(int a, int b);
  class Trida
  {
    private:
      int Atribut;
    public:
      void metoda();
  };

  void Trida::metoda()
  {
    cout << "Ahoj" << endl;
  }
```

```
}

int RadimuvProstor::secti(int a, int b)
{
  return a + b;
}
```

V mém prostoru jmen jsem deklaroval funkci `sečti` a třídu `Třída`. Dále jsem definoval tělo metody `metoda`. Tělo funkce `sečti` jsem sice definoval "mimo" závorky, ale přesto patří do definovaného prostoru jmen. Vně závorky vymezující prostor jmen přistupujeme k identifikátorům z tohoto prostoru jmen jako by k prvkům třídy. Nyní zkusme definovat úplně jinou funkci `sečti`, která je mimo tento prostor jmen. K předchozímu zdrojovému textu dopište:

```
int secti(int a, int b)
{
  return 2 * a + 2 * b;
}

int main(void)
{
  cout << secti(2,3) << endl;
  cout << RadimuvProstor::secti(2,3) << endl;
  return 0;
}
```

V tomto případě nedojde ke konfliktům jmen. Je to proto, že vlastně neexistují dvě funkce `sečti`, ale funkce `sečti(int,int)` a `RadimuvProstor::sečti(int,int)`. Tím že jsem funkci zařadil do prostoru jmen, jsem vlastně rozšířil její název. Byla by veliká náhoda, kdyby se našly dva prostory jmen stejného názvu. U názvů tříd, funkcí, globálních proměnných atd... je to ale dost možné. Název prostoru jmen by měl být vybrán tak, aby byl jednoznačný. Identifikátory uvnitř jednoho prostoru jmen se nepoužívají s názvem tohoto prostoru jmen.

Implicitní prostor jmen

Dalo by se říci, že každý identifikátor je v nějakém prostoru jmen. Jestliže prostor jmen neuvedeme (Jako já u druhé funkce `sečti`, u funkce `main` a také ve všech svých příkladech z předchozích článků.), je tento identifikátor v implicitním prostoru jmen, někdy se také nepřesně říká "globální prostor jmen", který není pojmenován. Pro přístup k němu se před operátorem "čtyř-tečka" neudává jméno. V naší funkci `main` můžu tedy místo `cout << secti(2,3) << endl;` napsat `cout << ::secti(2,3) << endl;`. V tomto případě je tato čtyř-tečka samozřejmě zbytečná, ale mohla by se hodit, kdybychom někde uvnitř mého prostoru jmen chtěli vyvolat "globální" funkci sečti. Například přepíšu metodu `metoda`, ve které zavolám obě funkce `sečti`:

```
  void Trida::metoda()
  {
```

```
    cout << ::secti(1,1); << endl;/* "globální" sečti */
    cout << secti(1,1); << endl;/* sečti z tohoto prostoru jmen*/
  }
```

Prostor jmen "std"

Podle ANSI C++ existuje standardní prostor jmen `std`. V tomto prostoru jmen je definováno mnoho identifikátorů ze standardní knihovny jazyka C++. V `std` jsou také, jak jsem byl správně ve svých předchozích článcích čtenáři upozorněn, objekty `cin`, `cerr`, `cout` a mnoho dalších. Konkrétně u těchto 3 objektů se dnes překladače chovají dost tolerantně, a přeloží je i když jsou používány jako by v implicitním prostoru jmen. Postupem času se překladače budou blížit normě, a takové nepřesnosti nebudou tolerovat. Je tedy dobré tyto objekty identifikovat jako `std::cout`, `std::cin`, `std::cout`, nebo si pomoci klíčovým slovem `using`.

Klíčové slovo using

Jak už jsem se zmínil, prostory jmen jsou proto, aby nedošlo ke konfliktům jmen. Pomáhají programátorovi v uspořádání zdrojových textů, ale používání prostoru jmen by se mohlo zdát jako "psaní něčeho navíc". Aby programátor nemusel vždy psát jméno prostoru společně i se jménem, které chce napsat, existuje klíčové slovo `using`. Napíši-li `using namespace` a jméno prostoru, bude v následujícím zdrojovém textu přístup k identifikátorům tohoto prostoru jako by jsem byl "uvnitř" tohoto prostoru. Tedy místo:

```
#include<iostream>
int main(void)
{
  std::cout << "Ahoj svete" << endl;
  return 0;
}
```

Lze napsat:

```
#include<iostream>
using namespace std;
int main(void)
{
  cout << "Ahoj svete" << endl;
  return 0;
}
```

Musíme mít ale natolik nový překladač, aby měl objekt `cout` v prostoru jmen `std`.

Závěrem k prostorům jmen bych Vám chtěl doporučit prostory jmen používat. Narazíte-li na něco, co v prostoru jmen již je, nic jiného Vám ani nezbude. Je dobré také své identifikátory do prostoru jmen ukládat. Nemůže-li dojít ke konfliktu jmen, můžete použít slovo `using`, a v dalším zdrojovém textu si ani nevšimnete, že prostory jmen používáte. Naopak nebudete-li používat prostory jmen a ke konfliktu dojde, už Vám nezbude nic jiného, než přejmenovávat identifikátory. Já ve svých následujících článcích prostory jmen používat nebudu čistě z praktického hlediska. Moje příklady jsou natolik krátké a jednoduché, že prostory jmen by je udělaly jen méně přehledné.

Ohledně používání prostorů jmen bych chtěl ještě upozornit na jednu častou nepřesnost. Syntakticky je prostor jmen velice podobný třídě. Jedná se ale o něco úplně jiného. Třída je abstrakcí nějakých objektů, které mají podobné vlastnosti. U třídy se počítá s tím, že bude mít instance. Pokud možno, každá třída by měla být navržená tak, aby mohla mít více instancí. Naopak prostor jmen žádnou instanci nemá, ani mít nemůže. Jedná se jen o oddělení globálních proměnných, funkcí, uživatelem definovaných datových typů (tedy i tříd), které spolu nějak souvisejí od "zbytku" programu. Mnoho programátorů chybně prostory jmen nepoužívá, a místo prostoru jmen použije třídu.

Tolik tedy k prostorům jmen v C++. Příště se podíváme na řetězce v C++. V C++ existuje třída `string`. S instancemi této třídy se pracuje rozhodně pohodlněji, než s polem znaků jak tomu bylo u jazyka C. O tom ale až příště.