Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 14.03. 2001
Url: http://www.builder.cz/art/cpp/cpp_vicededicnost.html

Vícenásobná dědičnost v C++ - opakovaná dědičnost

V minulém článku jsem popsal vícenásobnou dědičnost, vyjmenoval problémy, které přináší a ukázal problém konfliktu jmen. V tomto článku objasním druhý problém - opakovanou dědičnost. Jak jsem uvedl minule, opakovaná dědičnost nastane, jestliže v třídním diagramu existuje mezi 2 třídami více, než jedna cesta. Tedy například:

```
class A
{
  public:
    int A;
};

class B : public A
{
  public:
    int B;
};

class C : public A
{
  public:
    int C;
};
```

Třídy B, C mají tribut `int A` zděděný z třídy A. Nyní vytvořme třídu D následovně:

```
class D : public B, public C
{
  public:
    int D;
};
```

Třída D obsahuje dva atributy `int A` . Vše je v pořádku jestliže z nějakého důvodu chceme mít atribut `int A` ve třídě D dvakrát. K atributům B::A a C::A potom přistupujeme jako při běžném konfliktu jmen, který jsem popsal v minulém článku. Zde je ale spíše logické chtít, aby atribut `int A` byl ve

třídy D jen jednou, protože se vlastně jedná o jeden atribut A::A, který byl opakovaně zděděn.

Virtuální nadtřída

Řešením je udělat třídu A virtuální nadtřídou tříd B, C. K vyjádření virtuální nadtřídy se používá klíčové slovo `virtual` , které známe jako slovo deklarující metodu volanou pozdní vazbou. "Virtuální dědění" nemá s pozdní vazbou nic společného. Opět vidíme v C++ situaci, kdy jedno klíčové slovo má více významů. Opravíme zdrojový text takto:

```
class A
{
  public:
    int A;
};

class B : virtual public A
{
  public:
    int B;
};

class C : virtual public A
{
  public:
    int C;
};

class D : public B, public C
{
  public:
    int D;
};
```

Nyní má třída D skutečně jen jeden atribut `int A` . Ještě jen upozorním, že všichni další potomci tříd B a C budou dědění virtuálně. Velkou nevýhodou virtuálního dědění je fakt, že pro virtuální dědičnost se musím rozhodnout dříve, než dojde k vícenásobnému dědění. Tedy když jsem v mém příkladě vytvářel třídy B a C, už jsem musel počítat s tím, že bude existovat nějaká třída D, u které nastane problém opakovaného dědění. Jinak bych u jednoduché dědičnosti nepoužil klíčové slovo virtual. Není příliš dobré preventivně u každé dědičnosti použít klíčové slovo virtual, protože třída, která má virtuální nadtřídu obsahuje navíc ještě jeden ukazatel. Takže třída `class B1 : public A` není stejná jako třída `class B2 : virtual public A` . Onen zmiňovaný ukazatel není ukazatelem na tabulku virtuálních metod. Jedná se o ukazatel, který bude nutný pro sdílení prvků třídy při opakované vícenásobné dědičnosti.

Právě problémy s virtuálním děděním jsou velmi silným argumentem odpůrců vícenásobné dědičnosti. Používáte-li vícenásobnou dědičnost, doporučuji

Vám vyhnout se virtuálnímu dědění a všem problémům, které přináší. V příštím článku dokončíme téma vícenásobné dědičnosti přehledem pravidel pro vytváření a likvidování instancí tříd vyniklých vícenásobnou dědičností. Tedy povíme si, v jakém pořadí se volají konstruktory a destruktory při vícenásobné dědičnosti.