Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 07.03. 2001
Url: http://www.builder.cz/art/cpp/cpp_vicededic.html

Vícenásobná dědičnost v C++

Vícenásobná dědičnost je dědičnost, ve které je nová třída bezprostředním následovníkem více tříd. Narozdíl od jednoduché dědičnosti, kdy každá nová třída je bezprostředním následovníkem pouze jedné třídy.
Vícenásobná dědičnost se chová podle stejných pravidel jako dědičnost jednoduchá, jen na rozdíl od jednoduché dědičnosti vzniká při vícenásobné dědičnosti několik "drobných" problémů. Vícenásobná dědičnost je tedy opět specializace. Nová třída "dědí" po svých předchůdcích vlastnosti, s tím, že další vlastnosti mohou být přidány. Přístup ke zděděným položkám při vícenásobné dědičnosti má stejná pravidla jako přístup ke zděděným položkám při dědičnosti jednoduché. Ve svém článku "Jednoduchá dědičnost v C++" jsem uvedl tabulku přístupových práv, která beze změn platí i pro vícenásobnou dědičnost, proto ji zde nebudu uvádět znovu.

Syntaxe pro vícenásobnou dědičnost je velmi podobná jako syntaxe pro vyjádření dědičnosti jednoduché. Vlastně je zřejmé, že jednoduchá dědičnost je specielním případem dědičnosti vícenásobné, kdy počet předků je roven 1. Další předkové jsou oddělení čárkou. Tedy například:

```
class potomek : public Predek1, public Predek2, public Predek3 {};
```

Nyní jsem definoval třídu potomek, která má 3 předky, každý předek je děděn veřejnou dědičností. Nová třída nemá žádné nové atributy ani metody. Instanci třídy potomek v paměti počítače je dobré si představit jako-by byly "za sebou" 3 instance jednotlivých předků a dále nové vlastnosti třídy potomek.

Problémy při vícenásobné dědičnosti

Vse by bylo krásné nebýt několika velmi nepříjemných problémů, které vícenásobná dědičnost přináší. Problémy:

- Konflikt jmen

Při vícenásobné dědičnosti může nastat situace, že třída dědí z nadtříd, které mají stejné názvy atributů, nebo metod. Který potom zdědit?

- Opakovaná dědičnost

Nějaký atribut, nebo metoda mohou být zděděné "vícekrát". Opakovaná dědičnost nastává, jestliže v třídním diagramu existuje mezi 2 třídami více než jedna cesta. Například třída B dědí ze třídy A. Třída C dědí ze třídy A. Třída D dědí ze třídy C a B. Třída D vlastně dvakrát dědí ze třídy A. Má mít třída D položky třídy A dvakrát, nebo jen jednou?

- Volání konstruktorů a destruktorů

Jak jsem uvedl ve článku o jednoduché dědičnosti při vytváření instance se implicitně zavolají konstruktory předka, při rušení se volá implicitně destruktor potomka. Ve vícenásobné dědičnosti je ale bezprostředních předků více. V jakém pořadí se tedy mají volat konstruktory a destruktory?

Dalším problémem často se projevujícím u vícenásobné dědičnosti je správné (spíše špatné) přetypování potomka na předka. Jedná se ale spíše o problém s přetypováváním, proto se mu nebudu věnovat ve článcích o vícenásobné dědičnosti, ale až někdy v budoucnu ve článku věnovanému přetypování v C++.

Konflikt jmen

Tedy problém jaké atributy a metody dědit, jestliže se vyskytují ve více předcích se stejným názvem. Upozorňuji jen, že v tomto případě nesmí mít předci žádnou společnou nadtřídu. Mají-li, dojde sice také ke kolizi jmen, ale jedná se o problém opakovaného dědění. Problém kolize jmen se řeší tak, že třída zdědí všechny atributy a metody i v případě, že mají stejné jméno. Prostě ve třídě existuje více položek se stejným názvem. Protože přístup k položkám musí být jednoznačný, přistupuje se k položce pomocí tak zvaného úplného jména. Úplné jméno je jméno třídy následováno "čtyřtečkou" a poté následuje název atributu, nebo metody. Vše vysvětlím na příkladu:

```
#include <iostream.h>
class A
{
  public:
    int Atribut;
    void nastav(int a) { Atribut = a; }
    int vrat() { return Atribut; }
    virtual void metoda();
};

class B
{
  public:
    int Atribut;
    void nastav(int a) { Atribut = a; }
    int vrat() { return Atribut; }
    virtual void metoda();
};
```

```
class C : public A, public B
{
  public:
    void nuluj();
};

void A::metoda()
{
    cout << "A" << endl;
}
void B::metoda()
{
    cout << "B" << endl;
}
void C::nuluj()
{
    A::Atribut = 0;
    B::Atribut = 0;
/* Odkomentujete-li následující řádek, překladač Vás chybou upozorní na nejednoznačnost. */
    /* Atribut = 0; */
};
```

Dobře si všimněte jak v metodě C::nuluj přistupuji k atributu Atribut. Jedná se o úplné jméno atributu, čímž zamezím nejednoznačnosti. Kdybych chtěl zavolat metodu, jejichž jméno je v nějaké kolizi, musel bych také uvést úplné jméno. Každá z metod pracuje nad "svými" atributy. Tedy metoda A::nastav pracuje s atributem A::Atribut a metoda B::nastav pracuje s atributem B::Atribut. Pro pochopení přidávám následující funkci main:

```
int main()
{
     C c;
     c.A::nastav(10);
     c.B::nastav(20);
/* Odkomentujete-li následující řádek, překladač Vás chybou upozorní na nejednoznačnost. */
    /* c.nastav(0); */
     cout << c.A::vrat() << '\t' << c.B::vrat() << endl;
     c.nuluj();
     cout << c.A::vrat() << '\t' << c.B::vrat() << endl;
     c.A::metoda();
     c.B::metoda();
     return 0;
}
```

Jak vidíte každá metoda pracuje nad "svými" daty. U virtuálních metod je velkou výhodou, že je lze předefinovat. Proto by bylo vhodné metodu metoda ve třídě C předefinovat, aby se při jejím volání nemuselo psát úplné jméno. Nic mi nebrání tělo metody předefinovat například takto:

```
void C::metoda()
{
   A::metoda();
}
```

nebo:

```
void C::metoda()
{
   A::metoda();
   B::metoda();
}
```

Nebo jakkoliv jinak.

Tolik tedy k jednomu z problému vícenásobné dědičnosti. Příště budu řešit problém opakovaného dědění a poté v dalším článku problém konstruktorů a destruktorů při vícenásobné dědičnosti.