Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 20.03. 2001
Url: http://www.builder.cz/art/cpp/cpp_dedicnost_konstruktor.html

Vícenásobná dědičnost v C++ - volání konstruktorů a destruktorů

V tomto článku si ukážeme jak to je s voláním konstruktorů a destruktorů při vytváření a rušení instancí tříd vzniklých vícenásobnou dědičností. Stejně jako při jednoduché dědičnosti se i u vícenásobné dědičnosti nedědí konstruktory a destruktory, ale lze je vyvolat implicitně (stejně jako při jednoduché dědičnosti).

Konstruktory

Při vytváření instance třídy odvozené jednoduchou dědičností jsem napsal, že kromě konstruktoru této třídy je implicitně vyvolán bezparametrický konstruktor "nejvyšší" nadtřídy, potom konstruktor její podtřídy a tak dále až je vyvolán konstruktor dané třídy. U vícenásobné dědičnosti je situace obdobná. Není-li použito virtuální dědění (Viz předchozí článek Vícenásobná dědičnost v C++ - opakovaná dědičnost) je situace jednoduchá. Nejprve je volán konstruktor nejvyšší nadtřídy prvního předka (uvedeného v seznamu předků nejvíce vlevo), poté jeho potomka a tak dále, až je zavolán konstruktor předka naší třídy. Poté jsou stejným způsobem volány konstruktory ostatních předků naší třídy. Po vyvolání konstruktorů všech předků je vyvolán konstruktor naší třídy. Vše objasní příklad:

```
#include <iostream.h>
class A
{
  public:
    A() { cout << "A" << endl; }
};

class B : public A
{
  public:
    B() { cout << "B" << endl; }
};

class C : public A
{
  public:
    C() { cout << "C" << endl; }
};
class D : public B, public C
```

```
{
  public:
    D() { cout << "D" << endl; }
};

int main()
{
    D d;
    cout << "Instance d je vytvorena" << endl;
    return 0;
}
```

Program vypíše:
A
B
A
C
D
Instance d je vytvořena

Stejně jako u jednoduché dědičnosti lze i při vícenásobné dědičnosti volání bezparametrických konstruktorů potlačit a volat konstruktory s parametry.

Trochu problematičtější je použití virtuální dědičnosti. S virtuální dědičností jsou bohužel jen problémy. Rozdíly jsou vlastně dva:

- Ze všeho nejdříve budou vyvolány konstruktory virtuálních nadtříd a to bez ohledu na pořadí předků.
- Konstruktor virtuální nadtřídy bude volán jen jednou.

Pokud v mém předchozím příkladu upravíte třídy B a C tak, aby dědily virtuálně, program po spuštění vypíše:
A
B
C
D
Instance d je vytvořena
Tedy přesně podle pravidel, která jsem uvedl. Nyní můžeme trochu experimentovat a třídu deklarovat třeba takto: `class D : public B, public E, public C { /* To co tu bylo */ }` . Před tím ale třídu E deklarovat jako nevirtuálního potomka třídy A. Vznikne zajímavá situace. Třída D má nadtřídy B a C, které mají virtuální nadtřídu A. Tedy D bude mít po B a C zděděné atributy a metody jen jedenkrát. Ale dále ještě přímo dědí z E, kde A není virtuální předek, takže zde budou atributy a metody třídy A podruhé. Stejně tak volání konstruktoru třídy A bude dvakrát. Jednou se A::A() zavolá jako virtuální předek tříd B a C (Platí pravidlo, že konstruktor virtuálního předka je volán jen jednou.) a poté se zavolá znovu A::A() jako nevirtuální

přímý předek třídy E. Doporučuji Vám takto opravit zdrojový text a program spustit. Jak vidíte, použití jedné třídy jako virtuálního i nevirtuálního předka nám sem vneslo trochu zmatku.

Destruktory

Destruktory se chovají podle stejných pravidel jako konstruktory. S tím rozdílem, že jejich volání probíhá v opačném pořadí než volání konstruktorů. Tedy jejich volání neprobíhá směrem od "nadpředka" k potomkům ale od potomků k "nadpředkovi". Situace je obdobná jako při jednoduché dědičnosti.

Závěrem o vícenásobné dědičnosti

Používání vícenásobné dědičnosti (Hlavně použití virtuální dědičnosti při opakovaném dědění.) v C++ je složité a velmi nepřehledné. Představte si ne můj příklad, ale velký program, který má třeba desítky, nebo stovky tříd různě propojených jednoduchou a vícenásobnou dědičností. Bude asi radost orientovat se ve virtuálních a nevirtuálních nadtřídách a při volání konstruktorů a destruktorů virtuálních a nevirtuálních nadtříd. Některé programovací jazyky (Například Java) řeší tento problém velmi svérázným způsobem - vícenásobnou dědičnost vůbec nepodporují. O používání vícenásobné dědičnosti se vedou veliké spory. Vícenásobná dědičnost má své nadšence, kteří ji používají kde mohou, i velké odpůrce. Proti vícenásobné dědičnosti je silný argument. Není totiž znám způsob, kdy vícenásobnou dědičnost nelze korektně obejít. Další argumenty proti říkají, že vícenásobná dědičnost v C++ (hlavně virtuální dědění) je příliš složitá a nepřehledná. Ale podle mne je správné, že vícenásobná dědičnost v C++ je. Je-li tak špatná, jak tvrdí její odpůrci, nemusí ji programátor použít. Nedojde-li ke konfliktu jmen a opakovanému dědění, není vícenásobná dědičnost vlastně příliš složitá. S konfliktem jmen se při troše trpělivosti také dá pracovat. Jediné co Vám nedoporučuji, je pouštět se do opakovaného dědění a virtuální dědičnosti - to chce opravdu nervy :(

Tolik k vícenásobné dědičnosti. V příštím článku se podívám na přetěžování operátorů a poté v dalším článku na vstupní a výstupní operace v C++.