Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 27.07. 2001
Url: http://www.builder.cz/art/cpp/cpppretyp.html

Přetypování v C++

Dnes se podíváme na přetypování objektů a proměnných v C++. Jazyk C++ "zdědil" po jazyce C operátor přetypování. V jazyce C se přetypuje výraz například takto: `char a; int b; a = (char) b;`. Použít tento operátor přetypování v C++ není vhodné a někdy je dokonce i nebezpečné. Jazyk C++ nabízí nové operátory k přetypování: `dynamic_cast, static_cast, reinterpret_cast`. Jejich syntaxe je operátor<cílový typ>(výraz). Proberme si jednotlivé operátory podrobněji.

Operátor reinterpret_cast

Operátor `reinterpret_cast` slouží k přetypování spolu nijak nesouvisejících datových typů. Převod ukazatelů na instance nijak nesouvisejících tříd, nebo struktur (bez společného předka). Převody ukazatele na celá čísla a podobně. Operátor `reinterpret_cast` se používá především při programování na velmi nízké úrovni a jeho chování může být závislé na dané platformě. Uveďme si jednoduchý příklad.

```
#include <iostream>

using namespace std;

struct Struktura1
{
  short int a;
  short int b;
};

struct Struktura2
{
  int cislo;
};

int main()
{
  Struktura2 *s2 = new Struktura2;
  s2->cislo = 255;
  Struktura1 *s1 = reinterpret_cast<Struktura1*> (s2);
  cout << s1->a << " " << s1->b << endl;
```

```
  s2->cislo = 1000000;
  cout << s1->a << " " << s1->b << endl;
  int &c = reinterpret_cast<int&> (*s2);
  cout << c << endl;
  return 0;
}
```

Jak je zřejmé v mém příkladě předpokládám, že 2*sizeof(short int) == sizeof(int). To ale nemusí být na různých typech počítačů, nebo i na různých operačních systémech pravda. Jak napovídá název operátoru, dojde k změně interpretace nějakého místa v paměti. Je dobré si ještě všimnout, že v mém příkladě ukazatele s1, s2, i reference c se odkazují na stejné místo v paměti.

Operátor static_cast

Operátor `static_cast` je v podstatě lepší náhrada operátoru `(typ)` z jazyka C. Lze jej použít k různým konverzím mezi primitivními datovými typy, pro přetypování z potomka na předka, pro přetypování ukazatelů i referencí z potomka na předka, atd. Operátor `static_cast` není vhodný pro přetypování z předka na potomka. Příklad:

```
int main()
{
  int a = 65;
  char A = static_cast<char>(a);
  cout << a << " " << A << endl;
  return 0;
}
```

Převádím-li `static_cast<Cílový typ>(výraz)`, a u cílového typu existuje bezparametrický konstruktor s parametrem stejného typu jako výraz, bude výsledek vytvořen pomocí něj. Postará se o to operátor `static_cast`. Bude-li výraz objektového typu (třída) s přetíženým operátorem přetypování na cílový typ, bude použita metoda `operátor typ()` (přetížený operátor přetypování). Příklad:

```
#include <iostream>

using namespace std;

class Trida
{
private:
  int Atribut;
public:
  Trida(){}
  Trida(int i):Atribut(i) {cout << "Konstruktor" << endl; }
```

```
  operator int() { cout << "Operator (int) " << endl; return Atribut; }
};

int main()
{
  int a = 65;
  Trida t = static_cast<Trida>(a);
  int b = static_cast<int>(t);
  cout << b << endl;
  return 0;
}
```

Doporučuji všem tento program spustit a podívat se, co vše operátor `static_cast` provádí.

Jak je vidět z příkladu, operátor přetypování z jazyka C je zbytečný, dokonce i když je přetížen. Operátor `static_cast` přebírá veškerou práci za něj. Jak jsem se již zmínil, `static_cast` není vhodný pro přetypování předka na potomka. K těmto účelům se hodí následující operátor.

Operátor dynamic_cast

Operátor `dynamic_cast` slouží výhradně k přetypovávání ukazatelů, nebo referencí. Z ukazatele lze vytvořit pouze ukazatel a naopak z reference lze vytvořit pouze reference. Operátor `dynamic_cast` je vhodné použít pro přetypování ukazatele (nebo reference) na předka na ukazatel (referenci) na potomka. Operátor `dynamic_cast` bezpečně přetypovává polymorfní třídy (Třída obsahující alespoň jednu virtuální metodu.), protože k přetypování dojde až za běhu programu s pomocí dynamické identifikace typů. Nelze-li přetypovat ukazatele, je výsledný ukazatel roven `NULL`. Nelze-li přetypovat reference je vyvržená výjimka typu `bad_cast`. Příklad na operátor `dynamic_cast`:

```
#include <iostream>
#include <typeinfo>

using namespace std;

class Predek
{
private:
  int A;
public:
  void nastavA(int a){ A = a; }
  virtual int dejA() { return A; }
};

class Potomek : public Predek
{
```

```
private:
  int B;
public:
  void nastavB(int b) { B = b; }
};


int main()
{
  Predek *pr = new Predek, *po = new Potomek;
  Potomek *p;
  pr->nastavA(10);
  po->nastavA(20);
  /* Ukazatel po ukazuje na potomka, lze s ním pracovat jako s potomkem! */
  if ((p = dynamic_cast<Potomek*>(po)) != NULL)
  {
    cout << "OK" << endl;
    p->nastavB(10);
  }
  /* Ukazatel pr ukazuje na předka, nelze s ním pracovat jako s potomkem! */
  if ((p = dynamic_cast<Potomek*>(pr)) != NULL)
  {
    cout << "Nestane se" << endl;
    p->nastavB(10);
  }
  try
  {
    Potomek &ref = dynamic_cast<Potomek&>(*po);
    cout << "Přetypováno" << endl;
    ref = dynamic_cast<Potomek&>(*pr); /* Bude vyvolána výjimka. */
    cout << "Přetypováno" << endl;
  }
  catch (bad_cast& e)
  {/* Odchycení výjimky chybného dynamického přetypování referencí.
        (Chybné přetypování v době běhu programu.) */
    cerr << "Nelze pretypovat" << endl;
  }
  return 0;
}
```

Je důležité v takových případech používat operátor `dynamic_cast`. Zajistí přetypování až v době běhu programu, zajistí také bezpečnost přetypování. Znovu bych chtěl připomenout, že má-li být přetypování předka na potomka v dědičné hierarchii bezpečné, musí `dynamic_cast` přetypovávat polymorfní typy. Mnoho programátorů dělá velkou chybu, když operátor `dynamic_cast` ignorují, a používají místo něj přetypování z jazyka C. Takové přetypování není bezpečné. Nevěříte-li, zkuste v mém poslední příkladu všechny operátory `dynamic_cast` přepsat na `(typ)`, tedy přetypovat ukazatele tak, jak se to dělá v jazyce C. Program vypíše "Nestane se", a na řádku `p->nastavB(10);` zapíše 10 do nealokované paměti se všemi následky!

Tolik tedy pro začátek k přetypování. Předpokládali jsme, že dědičnost v posledním příkladě bude jednoduchá, nikoliv vícenásobná. Ve svých článcích o vícenásobné dědičnosti jsem slíbil, že se ještě vrátím k přetypovávání instancí tříd vzniklých vícenásobnou dědičností. Dnes jsme se seznámili s operátorem `dynamic_cast`, proto se příště můžeme podívat na použití `dynamic_cast` při vícenásobné dědičnosti. Tím téma přetypování skončíme a v dalším článku se začneme zabývat šablonami v C++.