Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 29.03. 2001
Url: http://www.builder.cz/art/cpp/cpp_pretizeni2.html

Přetěžování operátorů v C++ 2. díl

V tomto článku navazuji na svůj předchozí článek o přetěžování operátorů v C++. Všem, kteří můj předchozí článek "Přetěžování operátorů v C++ 1. díl" nečetli, doporučuji si jej přečíst. Všechny zdrojové texty, které napíši v tomto článku, přímo navazují na článek předchozí.

Dalším operátorem, který přetížím, je operátor volání funkce. Chtěl bych ukázat, že i tento operátor lze přetížit. Nenapadá mně ale žádný rozumný význam tohoto operátoru pro vektor. Tento operátor vrátí x-tý prvek vektoru zvětšený o y. Je to nesmyslný příklad, ale nic lepšího mě nenapadá. Díky přetížení tohoto operátoru se budu moci chovat k instancím třídy Vektor jako k funkcím s dvěma parametry typu `int`.

```
int Vektor::operator()(int x, int y) const
{
  return ((x >= 0) && (x < 3))? (*this)[x] + y : y;
}
```

Dalším operátorem, který přetížím, je operátor indexování pole. Právě díky přetížení tohoto operátoru můžu pracovat s instancemi třídy vektor jako s jednorozměrným polem. Použil jsem jej již v těle operátoru `==`, kde jsem napsal výraz `druhy[i]`. V těle této metody jednoduše ošetřím případnou chybu indexování. Bude-li požadavek mimo rozsah pole, vrátí metoda prvek s indexem 2.

```
int& Vektor::operator[] (int i) const
{
  if ((i>=0) && (i<3))
  {
    return pole[i];
  }
  return pole[2];
}
```

Je dobré si všimnout, že metoda nevrací přímo prvek `int`, ale referenci na něj. Jak ukážu později, díky tomu může být výraz s operátorem indexování i na levé straně výrazu. Což se od něj také intuitivně očekává. Nevýhodou je, že tímto způsobem (Když operátor indexování bude ve výrazu vlevo) lze změnit i konstantní objekt, což začíná dělat zdrojový text nepřehledným.

Nyní ještě přetížím operátor `*` jako funkci.

```
Vektor operator*(const int cislo, const Vektor &b)
{
   Vektor navrat;
   for(int i = 0; i < 3; i++)
      navrat[i] = b[i] * cislo; /* I zde je operátor [] vlevo */
   return navrat;
}
```

Tolik tedy přetížených operátorů. Nyní ukáži jednoduchou funkci `main`, ve které předvedu několik operací s instancemi mé třídy Vektor. Jsou-li například m, n vektory, a chci je sečíst pomocí mého operátoru, měl bych "správně" napsat: `a.operator=(m.operator+(n));` . Ale právě proto přetěžuji operátory, abych je mohl zapisovat přirozeně. Tedy mohu klidně napsat `a = m + n;` .

```
#include <iostream.h>
int main(void)
{
   Vektor a,b,c;
   for (int i = 0; i < 3; i++)
   {
      a[i] = i + 3; /* Nebo také a.operator[](i) = i + 3; */
   }
   b = c = a;
   b = 10 * c;
   if (a == c)
      cout << "OK" << endl;
   cout << b(1,2) << endl; /* Operátor volání funkce */
   a = b + c;
   for (int i = 0; i < 3; i++)
   {
      cout << a[i] << endl;
      cout << b[i] << endl;
      cout << c.operator[](i) << endl;
   }
   return 0;
}
```

Závěrem bych vám jen chtěl doporučit, abyste operátory přetěžovali s rozvahou. Je skutečně fantastické věc, mohu-li přetěžovat operátory `+, *` , atd... pro nějaké matematické objekty jako třeba matice, vektory, body, souřadnice, zlomky, iracionální čísla a podobně. Co ale u jiných tříd objektů? Co je například součtem dvou oken? Je mnoho tříd, u kterých přetěžování operátorů vede jen k nepřehlednostem ve zdrojovém textu. Dále také přetěžujte operátory tak, aby se chovali, jak se od nich intuitivně očekává. Překladači nebude vadit, jestliže operátor `+` bude odčítat. Zmatete tím sice dokonale "západní rozvědky", ale hlavně také sebe, nebo své kolegy, s nimiž programujete.

Tolik tedy k přetěžování operátorů a v příštím článku se podíváme na vstupní a výstupní operace. Budeme při tom přetěžovat operátor `<<` a operátor `>>`,

jejichž původní význam jsou binární posuny. Jak si ale mnozí určitě všimli, používám je i já ve svých článcích v souvislosti s doposud "záhadným a tajemným" objektem `cout` k výpisu na stdout.