Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 26.03. 2001
Url: http://www.builder.cz/art/cpp/cpp_pretez1.html

Přetěžování operátorů v C++ 1.díl

Nejprve se pokusím vysvětlit co pojem přetěžování operátorů znamená. Pro lepší představu považujme libovolný operátor v C++ (například operátor `+` , nebo `*`) jako normální funkci. Například operátor `+` může být považován za funkci, která má dva parametry a vrací výsledek. Oba parametry mohou být třeba `int` a návratová hodnota je potom také `int`. O přetěžování funkcí, nebo metod jsem se částečně již zmínil ve svém úvodním článku mého seriálu. Jedná se o to, že lze definovat i deklarovat metody, nebo funkce se stejným jménem ale s různými parametry. Stejně tak existují i přetížené operátory. Například již zmiňovaný operátor `+` je také přetížen. Lze jej totiž také použít například pro sčítání čísel `float`, atd... Fakt, že operátory lze použít pro operandy různých typů (Tedy že jsou přetížené) nebude asi pro nikoho, kdo zná nějaký programovací jazyk, žádnou novinkou. V C++ ale existuje způsob jak přetížit operátory pro programátorem definované typy.

Pravidla pro přetěžování operátorů

Nejprve se podíváme, jaké pravidla a omezení musíme používat pro přetěžování operátorů.

- Přetížené operátory se nemohou lišit pouze v typu návratové hodnoty, ale v typech parametrů. Toto omezení platí i pro přetěžování funkcí, nebo metod.
- Přetížíme-li binární operátor, musí být zase binární. Přetížíme-li unární operátor, musí být zase unární. U operátorů tedy nelze měnit počet parametrů. Například operátor `/` je binární (má dva operandy), nemůžeme jej přetížit na unární operátor.
- Nelze vytvářet nové operátory.
- Nelze přetěžovat operátory: `. .* :: ?:` Ostatní přetěžovat lze. Lze dokonce přetěžovat operátor `[]` - indexování, `()` - volání funkce, i přetypování atd...
- Pro operátory platí stále stejná priorita, ať jsou přetížené jakkoliv. Prioritu operátorů v C++ nelze nijak změnit.

Jak přetěžovat operátory

Operátory v C++ lze přetěžovat dvěma způsoby. Jednak jako členské metody a jednak jako funkce. V mých předchozích článcích jsem příliš nedoporučoval používání funkcí, jen členských metod. Také jsem ale hned uznal, že v C++ existuje mnoho případů, kdy je použití funkcí výhodnější, nebo i nezbytné. Přetěžování operátorů je jedním z těchto případů. Mnohdy bude výhodné operátor přetížit jako funkci. Operátor se přetíží jako funkce, ne jako metoda, je-li jeho 1. parametrem primitivní datový typ, nebo třída, kterou nemůžeme měnit. Binární operátor přetížen jako funkce bude mít dva parametry, unární jen jeden. Binární operátor přetížený jako metoda bude mít jen jeden parametr, protože druhým parametrem bude implicitní parametr

`this`. Unární operátor přetížený jako členská metoda nebude mít parametr, protože jeho jediným parametrem je `this`. K přetěžování operátorů slouží klíčové slovo `operator`. Jako příklad uvedu třídu pro tří rozměrný vektor a přetížím pro ni mnoho operátorů. Pro jednoduchost budou souřadnice vektoru celá čísla. Všechny operátory budou přetíženy jako členské metody kromě operátoru `*`. Tento operátor přetížím jako funkci.

```
class Vektor
{
  private:
    int *pole;
  public:
    Vektor():pole(new int[3]){}
    Vektor(const Vektor &kopie);
    ~Vektor() { delete[] pole;}
    Vektor operator=(const Vektor &kopie);
    bool operator==(const Vektor &druhy) const;
    bool operator!=(const Vektor &druhy) const;
    Vektor operator+(const Vektor &druhy) const;
    int operator() (int x, int y) const;
    int& operator[] (int i) const;
};
Vektor operator*(const int cislo, const Vektor &b); /*Deklarace ("hlavička") funkce */

Vektor::Vektor(const Vektor &kopie) /*Nutný kopírovací konstruktor */
:pole(new int[3])
{
   int i;
   for(i = 0; i < 3; i++)
      this->pole[i] = kopie[i];
}
```

Jistě by se dalo najít více operátorů, které přetížit. Například `+=` a podobné. Myslím, že pro příklad stačí tyto operátory. Nyní budu postupně implementovat těla metod. Nejprve operátor =. O významu tohoto operátoru jsem se již zmiňoval ve svém článku "Kopírovací konstruktor v C++". Nepřetížený operátor = vytváří pouze plytkou kopii, což v našem případě není výhodné. Proto jej musíme přetížit. U přetěžování všech operátorů nezapomeňte na jejich návratové hodnoty. Každý přetížený operátor by se měl chovat i vracet hodnoty tak, jak se od něj intuitivně očekává. Například operátor `+` by neměl odečítat a podobně. Stejně tak by operátory měly vracet hodnotu, která se intuitivně očekává. Například operátor = vrací svou pravou stranu. Mám-li například `int a; a=3;` , potom výraz `a = 3` vrací 3. Normálně lze napsat `b = a = 3;` . Proto bude i náš operátor vracet pravou stranu.

```
Vektor Vektor::operator=(const Vektor &kopie)
{
   for (int i = 0; i < 3; i++)
   {
      pole[i] = kopie[i];
```

```
  }
  return kopie;
}
```

Operátor `=` není konstantní metoda, protože mění objekt `this`.

Dalším operátorem, který přetížíme bude operátor `=` , tedy operátor pro porovnání. Od tohoto operátoru se intuitivně očekává, že bude vracet proměnnou typu bool. V případě rovnosti objektů vrátí `true`, v opačném případě `false`. Při psaní této metody se dopustím veliké nepřesnosti. Pro jednoduchost budu považovat dva vektory za shodné, budou-li obsahovat stejné prvky. Tento příklad slouží pouze jako ukázka jak přetěžovat operátory, ne jak počítat s vektory. Matematicky správně je převést oba porovnávané vektory do normovaného tvaru a ty potom porovnat. Nechci Vás ale zatěžovat zdrojovým textem převádějícím vektory do normovaného tvaru, když téma článku je jiné.
Tato metoda vlastně srovnává objekty `this` a druhy. Objekt this se nezmění, proto operátor `==` může být deklarován jako konstantní metoda.

```
bool Vektor::operator==(const Vektor &druhy) const
{
  for (int i = 0; i < 3; i++)
  {
    if (pole[i] != druhy[i])
      return false;
  }
  return true;
}
```

Operátor != lze nejjednodušeji přetížit takto:

```
bool Vektor::operator!=(const Vektor &druhy) const
{
  return !(*this == druhy);
}
```

Dále přetížíme operátor `+` pro sečtení dvou vektorů. Objekt `this` se opět nemění, proto je metoda deklarována jako konstantní.

```
Vektor Vektor::operator+(const Vektor &druhy) const
{
  Vektor navrat;
  for (int i = 0; i < 3; i++)
    navrat[i] = (*this)[i] + druhy[i];
  return navrat;
}
```

Text článku je již příliš rozsáhlý a proto téma přetěžování operátorů dokončím příště. Další článek bude přímo navazovat na tento. Dokončím v něm přetěžování všech operátorů, které jsem deklaroval a ukážu i jednoduchý příklad ve funkci `main`, jak různě lze přetížené operátory použít.