Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 06.12. 2000
Url: http://www.builder.cz/art/cpp/cpp_oop.html

Základy OOP v C++: Od C k C++

Úvod

Úvodem musím říci, že můj seriál článků "Základy objektově orientovaného programováni v C++" je určen lidem, kteří znají jazyk C, a chtějí se naučit objektově programovat v jazyce C++. Pro ty, kteří jazyk C neznají, bych nejprve doporučil seriál článků mého kolegy pana Růžičky, který také vychází na serveru Builder.cz. Dále bych chtěl také upozornit, že veškerý můj výklad se bude týkat ANSI normy jazyka C++, a budu-li vycházet z jazyka C, budu vycházet jen z ANSI normy jazyka C. Je tak zaručen bezproblémový přenos zdrojových textů mezi překladači různých firem, a dokonce i mezi různými operačními systémy. Bohužel ANSI norma C++ narozdíl od ANSI C, je známa teprve 2 roky, a mnoho starších překladačů ji nedodržuje, i když to o sobě prohlašují. Já všechny příklady zdrojových textů, které zde budu prezentovat, budu zkoušet na překladačích Borland C++ Builder BETA 0.0.4.212, a GNU C/C++ Compiler. Druhý zmiňovaný je standardní překladač pod OS Linux, který by měl ANSI C++ dodržovat. První zmiňovaný je testovací verze Builderu (ani ne verze 1) fy Borland, která jej uvolnila pro otestování. Můžete jej získat volně na některém z CD časopisu Chip, které vyšlo v roce 1996, nebo 1997. Tento překladač je již poměrně starý, ale dosud jsem nenarazil na nic z ANSI normy co by nepodporoval. Narazím-li na něco v průběhu psaní tohoto seriálu článku, upozorním vás na to. Máte-li však novější verzi Borland C++ Builderu, je to dobře pro vás. Od fy Borland by měl prý vyhovovat i překladač Borland C/C++ 5. Starší už ale asi ne. O překladačích jiných firem bohužel nevím nic, proto si budete muset zjistit sami nakolik podporují ANSI normu.

Protože snad 99.9% všech knih, či článků začíná na úvod malým programem, který pozdraví svět, začnu takto i já. Protože čtenář, který teprve začíná s C++ asi nebude vědět co se děje, doporučuji tento program jen opsat bez přemýšlení. Vstupně výstupním operacím se chci věnovat až někdy kolem 10. článku. A nyní slibovaný program pozdrav.cpp:

```
#include <iostream.h>

void main(void)
{
  cout << "Ahoj svete" << endl;
}
```

Je C podmnožinou C++?

Tuto otázku si klade asi každý, kdo s C++ začíná. Odpověď není vůbec tak jednoduchá jak se zdá. Já bych odpověděl: "Až na několik málo vyjímek si ano". Obecně platí, že C++ je rozšíření jazyka C. Jazyk C je čistě strukturovaný programovací jazyk. V jazyce C++ lze programovat objektově, nebo také strukturovaně. Bohužel jazyk C++ umožňuje i jakýsi mix strukturovaného, a objektově orientovaného programování. Tento mix vám rozhodně nedoporučuji. Zbytek tohoto článku budu věnovat strukturovanému programování v C++, a počínaje příštím článkem se budu věnovat jen objektově orientovanému programování.

Problémy při přechodu z C do C++

Zde popíšu v čem není C s C++ kompatibilní. Je to těch několik málo vyjímek, o kterých jsem se zmínil v minulém odstavci. Není totiž každý program napsaný pro C kompilovatelný jako program napsaný v C++.

- **Znakové typy** - C++ zná 3 typy reprezentující znaky: *char*, *signed char*, *unsigned char*. Zatímco *char* napsaný v C je totéž jako *signed char*. Například předáte-li funkci, která jako parametr očekává *signed char*, parametr *char*, pravděpodobně vás upozorní překladač na nekompatibilitu typů. *Char* bude nakonec implementován nejspíš jako *signed char*, nebo jako *unsigned char*, ale překladač k nim přistupuje jako k různým typům. Platí to však jen pro typ znak. U všech ostatních typů, jako třeba *int*, zůstává vše stejné jako u jazyka C. Tedy např. *int* je stejný typ jako *signed int*.
- **Implicitní návratová hodnota** - Další nekompatibilitou je implicitní návratová hodnota funkce. Nenapíšete-li návratovou hodnotu funkce v jazyce C, překladač C podle ANSI normy předpokládá, že návratová hodnota je *int*. Překladač C++ však v takovém případě předpokládá jako návratový typ *void* - tedy "nic". Takže deklarace funkce:

MojeFunkce(int cislo);

v jazyce C je funkce, která vrací *int*, kdežto v C++ se jedná o deklaraci funkce vracející void.

Možná, že těchto drobných problémů bude více. Já jsem však přišel jen na tyto. Jinak lze program napsaný v jazyce C považovat za program v jazyce C++. Pro strukturované programování však C++ nabízí řadu příjemných rozšíření.

Rozšíření jazyka C++ pro strukturované programování

Protože se v tomto seriálu článků chci zabývat objektově orientovaným programováním, popíšu jen letmo ty rozšíření, které se mi zdají opravdu nejvýznamnější.

- **Jednořádkový komentář** - Jednořádkové komentáře začínají dvěma znaky // a končí koncem řádku. Je asi dost překvapující, že tento komentář jazyk C podle ANSI normy nepodporuje, protože jej stejně snad všechny překladače jazyka C dovolují. Přesto v popisu ANSI normy C není, je až součástí jazyka C++. Abych uvedl nějaký příklad:

```
int  PocetCtvercu; // Udává počet čtverců
```

- **Deklarace proměnných** - V C++ oproti C se mohou proměnné deklarovat kdekoliv v textu, nejenom na začátku aktuálního bloku.
- **Datový typ bool** - Dalším příjemným rozšířením jazyka C je datový typ *bool*. Typ *bool* může nabývat dvou hodnot *true* (logická 1), *false* (logická 0). Nad datovým typem *bool* jsou definovány logické operace && (logický and), || (logický or), *!* (negace), ^ (logický xor). Tedy všechny logické operace, které lze použít v jazyce C v podmínkách. Příklad:

```
bool a;
bool b = true;
int c = 20;
bool d = ((c == 20) || ( c<=5));
a = !b;

if (a)
{
. . .
}
```

- **Návratový typ funkce main** - Návratový typ funkce main už v C++ nemusí být int, ale mže být také void. Viz můj první program pozdrav.cpp.
- **Implicitní hodnoty parametrů funkcí** - Implicitní hodnoty parametrů v C++ jsou již asi pro programátora v jazyce C novinkou. Jestliže předpokládáte, že budete funkci často volat s nějakou hodnotou parametru, můžete tuto hodnotu uvést jako implicitní, a při volání ji v seznamu parametrů neuvádět. Nejlépe asi demonstruji na příkladu:

```
int soucet(int a, int b, int c = 0)     /* 3. parametr je implicitně 0. */
{
  return (a+b+c);     /* vracím součet parametrů*/
}

void main(void)
{
int cislo;
. . . // Nějaký program
cislo = soucet(1,2,3);     /* Zavolá se funkce soucet s parametry a = 1 , b = 2, c = 3 - To asi nikoho nepřekvapí.*/
. . . // Nějaký program
cislo = soucet(2,3);     /* Zavolá se funkce soucet s parametry a = 2 , b = 3, c = 0. Tedy je to stejný výsledek, jako bych napsal cislo = soucet(2,3,0). */
}
```

Implicitní hodnoty parametrů mohu zadávat jen v seznamu parametrů zprava. Pokud by tedy v naší funkci součet nemel parametr c implicitní hodnotu,

parametr b by ji také nemohl mít. Parametr c (1. zprava) implicitní hodnotu má, proto lze definovat i implicitní hodnotu parametru b (2. zprava).

- **Přetěžování funkcí** - Přetížení funkcí je možnost deklarovat, i definovat více funkcí stejného jména s různým počtem, nebo s různými typy parametrů. Tuto vlastnost jazyk C neměl. Příklad: Mohu definovat dvě funkce se stejným jménem *max*.

```
int max(int a, int b)
{
  return (a>b)? a:b;
}

float max(float a, float b)
{
  return (a>b)? a:b;
}
```

Zavolám-li funkci max s parametry typu *float* vrátí mi větší z nich jako typ *float*. Zavolám-li funkci max s parametry typu *int* vrátí mi větší z nich jako typ *int*. Název funkce *max* jsem přetížil. Při přetěžování funkcí musíte dávat pozor na případné nejednoznačnosti. Kdyby jste v našem příkladě zavolali funkci *max(1.0,2.0)*, překladač by vás upozornil na chybu. Neví totiž, zda volat max s parametry float, nebo parametry implicitně přetypovat na int a volat max s parametry int. Tedy volání funkce je nejednoznačné, překladač neví, kterou funkci vybrat. Tento problém vyřešíte, jestliže budete volat *max ((float)2.0,(float)3.5)*. Nejlepší způsob jak předcházet případným nejednoznačnostem je nepřetěžovat funkce tak, aby mezi parametry šlo provést implicitní přetypování.

- **Reference** - V C++ existuje vedle proměnné nějakého typu a ukazatele na proměnnou nějakého typu také reference na proměnnou nějakého typu. Mám-li to říci hodně neformálně, tak reference je ukazatel, se kterým se pracuje stejně jako se statickou proměnnou. S pojmem ukazatel se jistě každý programátor v jazyce C již musel setkat. Referenci demonstruji na příkladu:

```
int a;    // Proměnná typu int
int &b = a;    /* Reference na proměnnou a, b je nyní jiný název pro číslo, které reprezentuje proměnná a, b je přezdívka a (alias). */
a++;     /* Nyní jsem změnil hodnotu a, i hodnotu b. Změnil jsem vlastně jen jednu hodnotu, proměnná a i b reprezentují tutéž proměnnou. */
```

Jedná se vlastně o jakousi obdobu klasického ukazatele (pointeru) v jazyce C. Ale POZOR reference NENÍ ukazatel. Narozdíl od ukazatele nemůže ukazovat "nikam", tedy mít hodnotu NULL. Právě tento fakt je důvod, proč já osobně reference nepoužívám, ale používám "staré a dobré" pointery - ostatně jak uvidíte v následujících článcích. I když jsem někde četl, že právě tato vlastnost je výhodou referencí od ukazatelů, mně se naopak zdá, že je nevýhodou. U reference totiž musím již v době deklarace znát proměnnou, kterou bude zastupovat, což u ukazatele nemusím. Chci-li někde nějakým způsobem použít sdílení proměnných, použiji raději ukazatele. Reference má výhodu oproti ukazatelům snad jen při předávání parametrů funkcí odkazem. Například funkci swap, která vymění hodnotu dvou čísel, je zbytečné psát pomocí ukazatelů, když v C++ lze zapsat následovně:

```
swap(int &a, int &b)
{
  int c = a;
  a = b;
  b = c;
}

// A někde v programu ji volat takto:
int a = 2, b = 9;
swap(a,b);
```

Tento zápis je přece jenom čitelnější a "lehčí" než implementace stejné funkce v jazyce C.

Závěr

Závěrem bych chtěl jen dodat, že rozšíření jazyka C++ oproti C pro strukturované programování je jistě mnohem více. Tento článek byl jen orientační, a hlavně odbočení od tématu seriálu o objektově orientovaném programování v C++. Zdálo se mi jen vhodné se o možnosti strukturovaného programování v jazyce C++ zmínit. Příště se chci podívat na základní pojmy objektově orientovaného programování. Vysvětlím v něm pojmy OOP jako jsou objekt, třída objektů, zpráva atd... A poté v dalších článcích chci ukázat, jak objektově programovat v jazyce C++.