Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 03.07. 2001
Url: http://www.builder.cz/art/cpp/cpp_vyjimky2.html

Výjimky v C++ - dokončení

Výjimka opustí tělo funkce main

Nyní se podívejme, co se stane, jestliže výjimka není odchycená a opustí tělo funkce `main`. K tomuto účelu si napíšeme velice jednoduchý program:

```
#include <iostream>
using namespace std;

int main()
{
        throw 1; /* Výjimka typu int */
        cout  << "Nestane se" << endl;
        return 0;
}
```

V minulém článku jsem nedoporučoval používat jako výjimky primitivní datové typy, ale teď jsem to sám udělal. Snad mi to odpustíte, o typ výjimky teď vůbec nejde. Můžeme tento program zkompilovat a spustit. Uvidíme, že program se ukončí s nějakou chybovou hláskou na stderr. Neošetřená výjimka je dost závažná chyba, proto takto "tvrdý" konec. Výjimka bude vyvržená tak, jak jsme si ukázali minule. Opustí-li výjimka tělo naší funkce `main`, dojde k zavolání funkce `terminate`. Tato funkce, není-li předepsáno jinak, vypíše onu chybovou hlášku a ukončí program zavoláním funkce `abort`. Funkce `abort` se postará o to, aby volajícímu procesu byla vrácena návratová hodnota 3. Takové chování nemusí být žádoucí. Můžeme například chtít vrátit jinou návratovou hodnotu, vypsat jinou, nebo žádnou hlášku (stderr u programů pod OS Windows není). Toho lze docílit pomocí funkce `set_terminate`. Funkce má jako parametr ukazatel na funkci, kterou má vyvolat volání funkce `terminate`. Funkce `terminate` i `set_terminate` jsou deklarovány v hlavičkovém souboru `exception`. Ve starších překladačích se tento soubor může jmenovat jinak. Například `except.h` Uveďme příklad:

```
#include <exception>
#include <iostream>
using namespace std;

void mojeTerminate()
{
        cerr <<"Bohužel, programátor nebyl schopen ošetřit výjimky."<<endl;
        exit(-10); /* Doporučuji vždy ukončit program. */
```

```
}

int main()
{
        set_terminate(mojeTerminate);
        throw 1; /* Výjimka typu int */
        cout  << "Nestane se" << endl;
        return 0;
}
```

Takto je chování programu v případě, že výjimka opustí funkci `main`, dáno ANSI normou. Problém nastane u aplikací pro Windows, které žádný stderr nemají. Chováni funkce `terminate` je jiné, a nejspíše se pro každý překladač liší. Například Borland C++ Builder vytvoří dialogové okno s hláškou: "Abnormal program termination". Budete-li ale chtít experimentovat s výjimkami v GUI aplikaci, zjistíte, že všechny výjimky vyvržené z metod formulářů jsou ošetřeny nějakým mechanizmem, který je pro programátora "neviditelný". Chcete-li si zkusit v GUI aplikaci v BCPPB vyvrhnout výjimku z funkce `WinMain` (obdoba `main`), musíte výjimku vyvrhnout přímo v této funkci. Pokud možno mimo blok `try`. U GUI aplikací v OS Linux tento problém nenastává, protože v Unixových systémech mají i GUI aplikace svůj standardní chybový výstup.

Vyvržení nedeklarované výjimky

Je-li z těla funkce, nebo metody vyvržená výjimka, která není v seznamu výjimek, které mohou být z těla funkce, nebo metody vyvrženy, dojde k zavolání funkce `unexpected`. Tato funkce implicitně zavolá funkci `terminate`. Chování funkce `unexpected` lze změnit pomocí funkce `set_unexpected`, která má jako svůj parametr ukazatel na funkci bez parametrů a vracející `void`. Použití funkce `set_unexpected` je obdobné jako použití funkce `set_terminate`. Funkce `unexpected` je deklarována v souboru `exception`. Příklad:

```
#include <exception>
#include <iostream>
using namespace std;

void mojeUnexpected()
{
        cerr << "Bohužel, programátor špatně pracuje s výjimkami"  << endl;
        exit(-10);
}

void f() throw()
{
        throw 1;
}

int main()
{
```

```
        set_unexpected(mojeUnexpected);
        f();
        return 0;
}
```

Standardní výjimky

V standardní knihovně C++ se výjimky příliš nepoužívají. Informace o chybě se často předává jako předem určená návratová hodnota z funkce, nebo metody, nebo pomocí chybového stavu - viz třeba objekt `cin`, nebo jiný objekt třídy `istream`, se kterým nelze pracovat v případě, že přečte chybná data. Několik standardních výjimek v C++ ale přece jenom je. Budeme se k nim postupně dostávat v dalších článcích. Všechny tyto výjimky jsou děděny ze třídy `exception`. Jedna výjimka, o které by jsme už ale měli vědět je výjimka typu `bad_alloc`, nebo nějaký její potomek, kterou vyvrhne operátor `new` v případě, že selže. V některých starších překladačích, které neodpovídají normě, se setkáme s výjimkou typu `xalloc`. Výjimka `bad_alloc` je definována v prostoru jmen `std`. Operátor `new` v případě selhání může buď vrátit NULL, nebo vyvrhnout tuto výjimku. Operátor `new` může selhat v situaci, kdy nelze alokovat požadovanou paměť. Úplné ošetření činnosti operátoru `new` je následující:

```
{
        int *pointer;
        try
        {
                if ((pointer = new int[3]) == NULL)
                { /* Přece jenom - pro jistotu */
                        cout << "Vrátil NULL" << endl;
                }
        }
        catch (std::bad_alloc &b)
        {
                cout << "Vyhodil výjimku" << endl;
        }
}
```

Jak se chová `new`, když selže, se můžete přesvědčit například tak, že v tomto příkladě místo čísla 3 (velikost pole) dáte hodně velké číslo. Chcete-li možnost vyvržení výjimky u operátoru `new` potlačit, lze to pomoci parametru operátoru `new`, který je `nothrow`:

```
{
        int *i;
        i = new(nothrow) int[10]; /* Určitě new nevyvrhne výjimku. */
}
```

Nevyvrhne-li výjimku operátor `new`, neznamená to samozřejmě, že nějaká nemůže být vyvržena z konstruktoru, který bude zavolán. Výjimka bad_alloc,

i parametr nothrow (Je to vlastně prázdná struktura.) se vztahují pouze na selhání alokace paměti.

Závěrem k výjimkám

Výjimky jsou určitě skvělá a pohodlná věc. Vše ale něco stojí. Při používání výjimek platíme tu největší daň za používání objektově orientovanému programování. Výsledný program používající výjimky bude jednak o mnoho větší a také mnohem pomalejší. Bude pomalejší dokonce i tehdy, nebude-li žádná výjimka vyvržená. Stačí pouze, že existuje blok `try`. Uvědomme si, co vše se musí v hlídacím bloku kontrolovat. To vše stojí čas a také jsou k tomu potřebné instrukce navíc. Záleží-li Vám opravdu na rychlosti programu, potom je používání výjimek na pováženou. Naopak ale například v programovacím jazyce Java se to výjimkami jen "hemží" a s rychlostí si nikdo nedělá starosti. Stejně tak podíváme-li se například na funkci `WinMain` v BCPPB, zjistíme, že celé její tělo je vlastně v hlídacím bloku. Používání výjimek je sice drahé, ale pro programátora velmi užitečné.

To by k výjimkám bylo asi tak vše. Příště se podíváme na dynamickou identifikaci typů.