C++ Tricks 1.2 逗號運算符(,)、邏輯運算符(&&,||)與運算符重載的陷阱

從 farseerfc.wordpress.com 導入

1.2 逗號運算符(,)、邏輯運算符(&&,||)與運算符重載的陷阱

很多人甚至不知道逗號(,)也是個C++運算符。與語法上要求出現的逗號(比如分隔函數參數的逗號)不同的是，出現在表達式中的逗號運算符在語義上表示多個表達式操作的連續執行，類似於分隔多語句的分號。比如：

for(**int**i=0,j=9;i<10;++i**,**--j)std::cout<<i<<”+”<<j<<”=9\n”;

在這句語句中，出現了兩個逗號，其中前者是語法上用來分隔聲明的變量的，並非逗號運算符，而後者則是一個逗號運算符。根據C++標準，逗號運算符的執行順序爲從左到右依次執行，返回最後一個子表達式的結果。由於只有最後一個表達式返回結果，所以對於一個語義正常的逗號表達式而言，前幾個子表達式必須具有副作用。同時，從語言的定義中也可以看出，逗號表達式對求值的順序有嚴格要求。

對求值順序有要求的，除了逗號表達式和條件表達式(參見1.1)，在C++中還有邏輯運算符(&&和||)。邏輯運算相較於數學運算和位運算而言，有個顯著的不同點：邏輯運算在計算到一半時，就有可能已經得到結果，這樣繼續運算另一半就不是必需的。對於A&&B，如果

A=false，那麼無論B爲何值，整個的結果都是false；同樣的A||B，如果A=true，那麼不考慮B，結果一定是true。

C++標準規定，如果邏輯運算到一半(算出A)時，就已經可以確定運算的結果，那麼就不運算剩下的另一半(B)。這種執行語義被稱作“短路”。在其它一些編程語言中，短路語義是可以選擇的：在Ada裏非短路的邏輯運算符爲and和or，短路的邏輯運算符爲and_then和or_else。但是在C++中，邏輯運算符的短路語義是語法上強制的，我們沒有非短路版本的運算符。如果確實需要非短路語義，我們總是可以通過增加一個bool中間變量加以解決。有時，短路對於保證正確執行是必須的，比如：

```
char*p=getString();

if(p&&*p)std::cout<<p;
```

這段代碼在得到了一個字符串後，在字符串不爲空時輸出它。在C++中判斷一個字符串不爲空需要兩個步驟：判斷指針是否爲0，以及指針不爲0時判斷指針指向的內容是否爲''。就像條件表達式中討論到的(參見1.1)，在p爲空時提領p是個極其危險的操作。邏輯運算符的短路語義則避免了這種危險。

以上對逗號運算符與邏輯運算符的討論，僅限於C++標準所定義的運算符語義。爲什麼這樣說呢？這是因爲在C++中，運算符的語義是可以由程序員自行定義的，這種機制叫做運算符重載(operator overload)。運算符重載可以將人們熟悉的運算符表達式轉換成函數調用，

使編程靈活而直觀，是個方便的語言特性。不過有時運算符重載也會使人困擾，那就是當運算符重載遇到求值順序問題時。

C++中，並不是所有合法運算符都可以被合法地重載。條件運算符雖然對求值順序有要求，但它並不在可重載運算符之列，所以運算符重載機制對它沒有影響。問題在於，逗號運算符和邏輯運算符都可以被合法地重載：

class BadThing{/* Some Bad and Stupid Thing*/};

BadThing& **operator**,(BadThing&, BadThing&);//重載了逗號運算符

bool operator&&(BadThing&, BadThing&);//重載了&&

BadThing b1,b2;

if(b1&&b2)b1,b2;//被替換成如下形式：

if(**operator**&&(b1,b2))**operator**,(b1,b2);

可以看到，重載了運算符之後，對運算符的使用被替換爲相應的函數調用形式。因此，舊有的運算符的執行順序不再適用，取而代之的是函數參數的壓棧順序。

根據C++標準規定，任何參數必須在進入函數之前壓棧，所以在進入**operator**&&之前，b1、b2就會被求值，這裏不再有短路規則，任何依賴於短路語義的不知不覺間操作BadThing的代碼(可能通過模板)都會混亂。

短路語義只是一個方面，更重要的在於壓棧順序。鑑於執行效率和舊代碼兼容性等細節問題，C++標準在壓棧順序上給編譯器的開發者留有很大自主性。標準的說辭是，編譯器可能以任何它覺得方便的順序將參數壓棧，從左到右，從右到左，甚至從中間到兩邊，在這一點上我們不能安全地做任何假設。在上面的例子中，編譯器生成的代碼可能先計算b1再計算b2，也可能是相反的順序。再看看編譯器的實際情況，在我試過的所有基於X86體系結構的編譯器中，參數都是以逆向壓棧，即從右到左，有悖於大多數人的閱讀習慣和直覺(別說你是來自伊斯蘭的……)。

在C時代使用函數調用時，壓棧順序並不是什麼大問題，畢竟大多數人會在函數調用的邊界稍稍小心一些。但是到了C++中，事情變得有些複雜，因爲簡單如a+b的使用，就有可能被運算符重載機制替換爲函數調用。更何況有模板參與之後，我們寫代碼時不能確定對象的真實類型，也就無法預知一個運算符是否真的被重載過，唯一穩妥的方法是，假定任何有可能被重載的運算符的使用都是函數調用。

```
<p style="margin:0;">
```

回到上文的示例中，由於,和&&都被替換爲函數調用，程序的執行順序將成爲壓棧順序，在X86上很有可能是從右到左，與標準定義的運算符的順序正好相反。逗號運算符原本就含有“先…後…”的語義，這種顛倒的執行順序勢必造成程序和程序員的混亂。以我的經驗而

言，含有**operator**,的類，完全沒有辦法和STL或者iostream相互協作，反而會導致巨量的錯誤報告(什麼叫巨量的錯誤報告有概念麼？如果沒有，那說明你還沒玩過範式編程(GP, Generic Programming)。去玩玩GP吧，看看你的編譯器對巨量的定義。在我手頭，針對3.5KB的代碼文件傾瀉出3.8**MB**的錯誤信息的編譯器不在少數……)。有鑑於此，我的結論是，除非你有充足的依據支持你這麼做(比如你的粗暴上司的鍵盤上只剩下逗號能用)，並且你清楚的瞭解這麼做的後果的嚴重性(比如至少要看過此文)，否則我奉勸你，永遠不要碰**operator**,、**operator**&&以及**operator**||！