C++ Tricks 1.1 條件運算符(?:)

從 farseerfc.wordpress.com 導入

1.1 條件運算符(?:)

條件運算符(?:)是C++中唯一的三目運算符(trinary operator)，用於在表達式中作條件判斷，通常可以替換if語句，與Visual Basic中的iif函數、Excel中的if函數有同樣的作用。語法形式如下：

condition ? *true_value* : *false_value*

其中*condition *條件是任何可以轉換爲bool類型的表達式，包括但不僅限於**bool***、**int**、指針。與**if**和**while**的條件部分稍顯不同的是，這裏不能定義變量，否則會導致語法錯誤。

另外，條件語句會切實地控制執行流程，而不僅僅是控制返回值。也就是說，兩個返回值表達式中永遠只有一個會被求值，在表達式的執行順序很重要時，這點尤爲值得注意。比如：

int *pi=getInt();

int i=pi**?***pi**:**0;

這裏，只有當pi的值不爲0時，它纔會被提領(dereference)。這種語義保證了程序的正確性，因爲提領一個空指針將導致致命的運行期錯誤(通常是非法操作的警告)。同時，正因爲條件運算符控制運算流程的特點，使得它不能用類似iif的普通函數來模擬：

int iif(**int** con,**int** t,**int**f){**if**(c)**return** t;**return** f;}//試圖模擬?:

…//in some function

int *pi=getInt();

int i=iif(pi,*pi,0);//Error!

這段代碼會導致上文提到的致命運行期錯誤。C/C++標準規定，參數在被傳遞給函數之前求值，因此無論pi爲何值，都會被提領。又因爲函數傳回一個空指針的情況比較少見，所以這樣的錯誤在調試時很難被發現，一旦發生又勢必造成重大災難。這樣的代碼在實踐中應儘量避免。

有時，條件運算符控制流程的特點會不知不覺影響我們的代碼。在C時代，最大值MAX通常用宏實現：

#defineMAX(a,b) ((a)>(b)**?**(a)**:**(b))

需要用額外的括號將宏參數和宏本體保護起來，以免運算符優先級擾亂邏輯，這是宏醜陋的特點之一，這裏暫且不提。矛盾在於，用具有副作用的表達式調用宏時，會出現問題：

int i=5,j=6;//…

int a=MAX(++i,++j);

代碼的作者原意顯然是想先將i,j分別遞增，再將其中較大的一個賦給a。執行這段代碼，當i=5,j=6時，a=8，知道爲什麼嗎？通過宏展開，賦值語句成這樣：

int a=(++i)>(++j)**?**(++i)**:**(++j);//刪除了多餘括號

在判斷之前，i、j被分別自增一次，然後捨棄:之前的部分，j又被自增一次。執行之後，i=6,j=8。

MAX的更正確更安全的實現，是利用模板將類型參數化。STL標準算法中就有一個這樣的工具級模版函數std::max。

條件運算符是表達式而不是語句，這使得它可以出現在任何需要表達式的地方，這擴大了它的適用範圍。在那些語法上只能出現表達式而不能出現語句的地方（比如變量初始化），條件運算符有着不可替代的作用。

條件運算符優於**if**語句的另一個場合是“模板元編程”(TMP, Template MetaProgramming)。在TMP這個古怪奇異的編譯期運算編程技術中，一切舊有的技術和法則被全線擊破，我們所能仰仗的工具，只有模板特化(Specialization)、**typedef**s、函數聲明(無法調用它們)、以及編譯期常量運算。已經有人很深入地論證過，僅有以上這些，就已經形成了一個“圖靈完善”的計算機語言。我們可以用模板特化技術，來模擬條件分支，循環迭代等一系列複雜的語言結構。由於可以參與編譯期常量運算，條件運算符在TMP世界中很自然地扮演起重要角色。

比如，給與類型T的一個變量t，我們想聲明一個緩衝區存放t和一個int，緩衝區的大小不小於sizeof(T)也不小於sizeif(int)，我們可以這樣寫：

char buffer[sizeof(T)>sizeof(int)? sizeof(T): sizeof(int)];

我們不能用一個if語句替換這個運算：

```
int i;
if(sizeof(T)>sizeof(int))i=sizeof(T);
else i=sizeof(int);
char buffer[i];//語法錯誤!
```

原因在於數組聲明中的下標必須是一個編譯期常量，而不是一個運行期的值，條件表達式的運算可以在編譯期進行，if語句就只能在執行期執行。