C++ Primer 答客問 (27) 【標準與實作之間】
PC 環境上三種編譯器對 C++ Standard 的支援

侯捷 jjhou@ccca.nctu.edu.tw
1999.12.29 第一次發表於
清大.楓橋驛站（140.114.87.5）.電腦書訊版（Computer/CompBook）

C++ Standard 相容編譯器

我想很多人關心，目前市面上哪些廠牌的 C++ 編譯器，完全支援 C++ Standard。C/C++ User Journal, Nov. 1999 的【C/C++ Stanadrd FAQ】專欄中，P.J. Plauger 對此問題的回答是：目前還沒有完全支援 C++ Standard 的編譯器產品。P.J. Plauger 閱歷廣泛，他的文章提到不同平台上的多種 C++ 編譯器（但並沒有深入談論，都只淺淺帶過）。

回憶歷史，C Standard 定案後，市面上立刻出現一大堆宣稱與標準規格完全相容 的 C 編譯器。爲什麼符合 C++ 標準規格的編譯器卻是如此難產呢？我想因素之一是，C++ 遠比 C 複雜得多，後期導入的一些性質（如 member templates, template partial specialization...）在編譯器技術層面上更是高難度。因素之二是，C++ Standard library 是個浩大的工程，而編譯器通常是以 bundle 的方式搭配其他公司的 C++ Standard Library，所以彼此的進度、技術、相容性都需要更多時間來協調配合。因素之三是，C++ 編譯器的價值比較，已經不再只是單純地比誰對 C++ Standard 的支援程度高（或甚至也不是比較誰的編譯速度快），而是比較在特定平台上對客戶是否有更多的企業服務。拿 Windows 環境上的 C++ 編譯器來說，「誰提供更好的 Windows 應用軟體開發工具與開發資源」可能比「誰更貼近 C++ Standard」，對客戶而言更爲重要。

但是大家還是會很關心哪家編譯器最貼近 C++ Standard -- 即使這不影響你對 C++ 編譯器的選擇。

個人經驗

自從我將 L&L（Lippman & Lajoie）的《C++ Primer》譯出後，面對 C++ Standard 所規範的許多嶄新性質，就有一股躍躍欲試的衝動。其後由於個人興趣，也因爲課程需要，把《C++ Primer 中文版》整個重新檢閱一遍，並動手在三種不同的編譯器上進行測試（都

是 PC/Windows 平台）。以下整理我的個人經驗，供各位參考。

我儘量爲每一個主題列出一份簡短而完整的測試碼。這些或許不是太嚴謹的測試，但是如果這些符合 C++ Standard 的程式碼無法通過編譯，我們說這個編譯器未能奉行 C++ Standard，並不過份。但反過來說，通過我所列之簡易測試者，或許仍有可能在更複雜的情況中出錯（尤其是 template 相關主題）。如果您曾有經驗，歡迎提供出來造福大家。

沒有什麼好點子，可以對以下各個主題編號排序。所以我以它們出現在《C++ Primer 中文版》上的頁次爲序。只擁有英文版的讀者請注意，中文版和英文版頁次完全相同。

測試環境

我的測試環境是：Intel Pentium，Windows 98，三套 C++ 編譯器：

```
(1) Microsoft Visual C++ 6.0（以下以 VC6 代表）
(2) Inprise Borland C++Builder 4.0 （以下以 BCB4 代表）
(3) GNU C++ egcs-2.91.57（以下以 G++ 代表）
```

請注意，GNU C++ 有著各種作業平台上的各種版本。我只用手上的 egcs-2.91.57 for win32 來測試。

我在 Windows 98 文字模式（console mode）底下進行編譯。以下是三種編譯器的環境設定（其中路徑可能與你不同。如欲循此方式設定，請自行修改）：

★VC6 環境設定

```
@echo off
rem
set PATH=C:\MSDEV\VC98\BIN;C:\MSDEV\COMMON\MSDEV98\BIN
set INCLUDE=C:\MSDEV\VC98\INCLUDE
set LIB=C:\MSDEV\VC98\LIB
```

★BCB4 環境設定

```
@echo off
rem
set PATH=C:\inprise\CBuilder4\BIN
set INCLUDE=C:\inprise\CBuilder4\INCLUDE
set LIB=C:\inprise\CBuilder4\LIB
```

★G++ egcs-2.91.57 環境設定

```
@echo off
rem
set PATH=c:\CYGNUS\CYGWIN~1\H-I586~1\BIN
set INCLUDE=
set LIB=
cls
```

我的編譯選項（compile options）極其簡單，

```
VC6  : cl -GX test.cpp
BCB4 : bcc32 test.cpp
G++  : g++ -o test.exe test.cpp
```

歡迎以下的討論與指正

- 如果加上任何編譯選項（compile options）即可解決（避免）所列之任何一個錯誤的話，歡迎（敬請）告知。
- 如果我的文字或程式碼有任何問題，歡迎（敬請）告知。
- 如果您有其他（未列於本文）的編譯器問題，歡迎（敬請）告知。

歡迎於 BBS/News 上討論。如能同時轉寄一份給我，以免除我遺漏觀看的可能，最是感謝。如不欲公開討論，亦歡迎將意見直接 email 給我。如欲討論，請寫範例，不要只是臆測。

■C++ Primer p213 下, p.393 下
主題：for loop 的 init-statement 區域內，所有 objects 皆爲 local。
測試結果：VC6[x] BCB4[o] G++[o]
實例：

```
#001 void main()
#002 {
#003 for (int ival=0; ival< 10; ++ival);
#004 for (int ival=0; ival< 10; ++ival);
#005 }
```

注意：C/C++ User Journal, Oct.1999, p.94 曾提過在 VC 上的一個簡易閃避辦法，這並且也是明載於 MSDN News（Vol7, Num6, Dr. GUI column）上的作法。設計一個巨集如下即可解決：

```
#define for   if(0); else for
```

該文並說，在簡單情況下可以有效運作，但並未測試太過複雜的情況。

■C++ Primer p262 中
主題：STL list 建構時，直接給 list 的大小以及所有元素的相同初值
測試結果：VC6[o] BCB4[x] G++[o]
實例：

```
#001 #include <list>
#002 ...
#003 const int list_size = 64;
#004 list<int> ilist1(list_size);     // BCB4 error.  VC6 ok. G++ ok
#005 list<string> ilist2(list_size);   // BCB4 ok.    VC6 ok. G++ ok
#006 list<int> ilist3(list_size, -1);  // BCB4 ok.    VC6 ok. G++ ok
#007 list<int> ilist4(list_size, 0);   // BCB4 error   VC6 ok. G++ ok
#008 list<int> ilist5(list_size, 1);   // BCB4 error   VC6 ok. G++ ok
#009 list<int> ilist6(list_size, -6);  // BCB4 ok.    VC6 ok. G++ ok
```

歸納：看來似乎 BCB4 不允許讓 list<int> 只獲得一個引數，也不允許 list<int> 獲得正值初值。很奇怪的行爲。我疏忽了什麼嗎？

■C++ Primer p383 上
主題：透過指向「函式指標陣列」的指標，喚起該陣列中的函式指標，其型式有簡略式和明顯式兩種。
測試結果：VC6[x] BCB4[x] G++[x] （都不支援簡略型式）
實例：

```
#001 #include <iostream.h>
#002
#003 typedef int (*PFV)();
#004 int f1() { return 1; }
#005 int f2() { return 2; }
#006
#007 PFV fFuncs[2] = { f1, f2 };
#008 PFV (*pfFuncs)[2] = &fFuncs;
#009
#010 void main()
#011 {
#012   cout << fFuncs[0]() << endl;  // 1
#013   cout << fFuncs[1]() << endl;  // 2
#014
#015   cout << pfFuncs[0]() << endl;  // 簡略式，VC6[x]  BCB4[x]  G++[x]
#016   cout << ((*pfFuncs)[1])() << endl;  // 2 （明顯式）
#017 }
```

■C++ Primer p410 上

主題：各編譯器對 auto_ptr 的支援

測試結果：VC6[o] BCB4[o] G++[x]

實例：

```
#001 #include <memory>  // for auto_ptr
#002 using namespace std;
#003 int main()
#004 {
#005   auto_ptr<int> pi(new int(1024));  // G++ error: auto_ptr undeclared.
#006 }
```

■C++ Primer p411

主題：string* 的建構（直接指定以另一個 string*）

測試結果：VC6[x] BCB4[o] G++[o]

實例：

```
#001 #include <string>
#002 using namespace std;
#003 int main()
#004 {
#005   string *pstr_type = new string( "Brontosaurus" );
#006   string *pstr_type2( pstr_type );  // <== VC6 error.
#007   delete pstr_type;
#008   delete pstr_type2;
#009 }
```

■p.412 中下

主題：auto_ptr 的 reset() 動作

測試結果：VC6[x] BCB4[o] G++[x]

實例：

```
#001 #include <memory>  // for auto_ptr
#002 using namespace std;
#003 int main()
#004 {
#005   auto_ptr<int> p_auto_int;        // <== G++ error
#006   p_auto_int.reset(new int(1024));  // <== VC6 and G++ error
#007 }
```

■C++ Primer p461

主題：lvalue-to-rvalue 轉換，rvalue-to-lvalue 轉換。

討論：lvalue-to-rvalue 屬於完全吻合（exact match）轉換的一種。
但是 rvalue-to-lvalue 呢？例如以一個 literal constant 或 temporary object 指派給一個 reference？應該不行，除非是指派給一個 const reference。

測試結果：我的經驗是，各編譯器對此一主題寬緊不一，且無定法（至少我歸納不出）

例一：

```
#001 int main()
#002 {
#003  int &i = 3;            // (1) should be error
#004                         // rvalue assign to non-const reference
#005  const int &i2 = 3;     // (2) should be ok
#006                         // rvalue assign to const reference
#007  int &i3 = int(3);      // (3) should be error
#008                         // rvalue (temp obj) assign to non-const reference
#009  const int &i4 = int(3); // (4) should be ok
#010                         // rvalue (temp obj) assign to const reference
#011 }
#012
#013 // G++ : (1),(3) warning: initialization of non-const reference `int &'
#014 //                      from rvalue `int'
#015 // jjhou 使用 G++ 2.91.57。
#016 // 據 jyhuang 說，G++ 2.92.2 並不允許通過 (1),(3)。
#017 //
#018 // VC6 : (1),(3) error: 'initializing' : cannot convert from 'const int'
#019 //                   to 'int &'. A reference that is not to 'const'
#020 //                   cannot be bound to a non-lvalue
#021 //
#022 // BCB4: (1),(2),(3),(4) warning: Temporary used to initialize 'i'
#023 //                            in function main ()
```

例二：

```
#001 void func1(int  i) { };         // pass by value
#002 void func2(int& i) { };         // pass by reference
#003 void func3(int* i) { };         // pass by pointer
#004 void func4(const int& i) { };    // pass by reference
#005
#006 void main()
#007 {
#008 int i;
#009 const int ci = 5;
#010
```

```
#011   func1(i);     // lvalue-to-rvalue, always ok.
#012   func1(ci);
#013   func2(i);
#014   func2(ci);    // (15)
#015   func3(&i);
#016   func3(&ci);   // (17)
#017   func4(i);
#018   func4(ci);
#019
#020   func2(int(6)); // (21), rvalue-to-nonconst-reference.
#021   func4(int(6)); // rvalue-to-const-reference, always ok.
#022 }
#023
#024 /*
#025 VC6 :
#026 (15) : error C2664: 'func2' : cannot convert parameter 1 from
#027       'const int' to 'int &'. Conversion loses qualifiers
#028 (17) : error C2664: 'func3' : cannot convert parameter 1 from
#029       'const int *' to 'int *'. Conversion loses qualifiers
#030 (21) : error C2664: 'func2' : cannot convert parameter 1 from
#031       'int' to 'int &'.
#032       A reference that is not to 'const' cannot be bound to a non-lvalue
#033
#034 BCB4 :
#035 Warning (15): Temporary used for parameter 'i' in call to 'func2(int &)'
#036 Error (17): Cannot convert 'const int *' to 'int *'
#037 Error (17): Type mismatch in parameter 'i' in call to 'func3(int*)'
#038 Warning (21): Temporary used for parameter 'i' in call to 'func2(int &)'
#039
#040 G++ :
#041 (15): warning: conversion from `const int' to `int &' discards const
#042 (3) : warning: in passing argument 1 of `func2(int &)'
#043 (17): warning: passing `const int *' as argument 1 of `func3(int *)' discards const
#044 (21): warning: initialization of non-const reference `int &' from rvalue `int'
#045 (3) : warning: in passing argument 1 of `func2(int &)'
#046 */
```

例三：詳見「C++ Primer 答客問 (19) part-3」

■p.492, p.499, p.500

主題：以 template nontype parameter 做爲陣列尺度（dimension）

測試結果：VC6[x] BCB4[o] G++[o]

注意：G++ 對於型別的 const-ness 檢驗極嚴格。以下 (1) 必須改爲
const int ia[5] =...; 才能通過 G++。

實例：

```
#001 template <class Type, int size>
#002 Type min( const Type (&r_array)[size] ) // VC6 error C2057:
#003 { /* ... */ }                        //  expected constant expression
#004
#005 void main()
#006 {
#007  int ia[5]={40,20,49,17,28}; // (1) 注意，G++ 要求需爲 const int ia[5]。
#008  min(ia);
#009 }
```

■C++ Primer p500 中上

主題：利用轉型運算子，將 template function 在模稜兩可（ambiguous）的環境下
以某特定型別具現化（instantiated）。

測試結果：VC6[x] BCB4[x] G++[x]

實例：

```
#001 template <typename Type, int size>
#002 Type min( Type (&r_array)[size] ) { /*... */ }  // VC6 error C2057
#003
#004 typedef int (&rai)[10];    // rai："10 個 ints 組成之陣列" 的 reference.
#005 typedef double (&rad)[20]; // rad："20 個 doubles 組成之陣列" 的 reference
#006
#007 // overloaded functions
#008 void func( int (*)(rai) ) { };    // int(*)(rai) 是函式指標型別，
#009                                   // 該函式的參數型別是 rai。
#010 void func( double (*)(rad) ) { }; // double(*)(rad) 是函式指標型別，
#011                                   // 該函式的參數型別是 rad。
#012
#013 void main()
#014 {
#015   func(static_cast<double(*)(rad)>(&min));  // (1) 此行無法編譯
#016   // BCB4 E2335: Overloaded 'min' ambiguous in this context
#017   // G++: undefined reference to `func(double (*)(double (&)[19]))'
#018 }
```

解決之道：繞個道，就可以。將上述 (1)：

```
func(static_cast<double(*)(rad)>(&min));
```

改爲以下即可：

```
double(*fp)(rad) = &min; // instantiate 'min', using specified type.
func(fp);
```

■C++ Primer p503

主題：如果 template function 的函式參數型別是一個 class template，而實際引數是一個 class，此 class 有個 base class，係從「被指定做爲函式參數」之 class template 身上具現出來，那麼 template 的引數推導可以順利進行。

測試結果：VC6[x] BCB4[x] G++[x]

實例：

```
#001 template <class T>
#002 class Array {  };
#003
#004 template <class T>
#005 class ArrayRC : public Array<T>  {  };
#006
#007 template <class T>
#008 T min4(Array<T>& array) { return T(0); }
#009
#010 void main()
#011 {
#012   ArrayRC<int> ia_rc();
#013
#014   // min4() 的函式引數型別是 ArraryRC<int>，其 base class 爲 Array<int>，
#015   // 正是 function template min4() 的函式參數型別 Array<T> 的
#016   // 一個具現體（instantiation），所以 min4() 應該可以接受它（書上說可以）
#017   min4(ia_rc);   // error in VC6, BCB4, G++2.51.97
#018 }
```

■C++ Primer p507

主題：明白指定一部份 template 引數型別，另一部份由編譯器推導而得。

測試結果：VC6[x] BCB4[x] G++[o]

實例：

```
#001 template <class T1, class T2, class T3>
#002    T1 sum( T2 v2, T3 v3)
#003    { return T1(v2+v3); }
#004
#005 typedef unsigned int ui_type;
#006
```

```
#007 ui_type calc( char ch, ui_type ui )
#008 {
#009   // 明白指定 T1 爲 ui_type，
#010   // T2 則被推導爲 char，T3 被推導爲 ui_type。
#011   ui_type (*pf)( char, ui_type ) = &sum< ui_type >;
#012
#013   ui_type loc = (*pf)(ch, ui);
#014   return loc;
#015 }
#016
#017 void main()
#018 {
#019  calc('c', ui_type(1024));
#020 }
```

■C++ Primer p508

主題：明白指定 template 引數型別

測試結果：VC6[x] BCB4[x] G++[o]

實例：

```
#001 template <class T1, class T2, class T3>
#002   T1 sum( T2 op1, T3 op2 ) { /* ... */ return T1(10); }
#003
#004 void manipulate( int (*pf)( int,char ) ) { };
#005 void manipulate( double (*pf)( float,float ) ) { };
#006
#007 void main( )
#008 {
#009   manipulate( &sum< double, float, float > );
#010 }
```

■C++ Primer p511

主題：separate compilation model for function template

測試結果：VC6[x] BCB4[x] G++[x]

VC6 不支援 export template

BCB4 支援關鍵字 export，但 linking 時找不到
temlate instantiation 在哪裡（unresolved external...）

G++ 不支援 export template

■C++ Primer p514

主題：funtion template explicit specialization

注意：書中以 max 爲自定之 function template 的名稱。然而有些編譯器已內附 max 函式（有的是屬於 runtime function，有的是屬於 generic algorithms），切莫以爲沒有含入相應的 header file，就不會喚起那些編譯器內附的東西，因爲有的 header files 會再含入其他 header files，那是你從表面看不出來的。所以，自己的碼千萬不要命名爲 max/min，才不會混淆自己。

測試結果：VC6, BCB4, G++ 都支援 funtion template explicit specialization。然而在 char*, const char*, const char[], text literal 之間，相當混淆而令人迷亂。

■C++ Primer p516

主題：funtion template explicit specialization + argument deduction

測試結果：VC6 表現太寬鬆，不嚴謹。

實例：

```
#001 #include <iostream>
#002 using namespace std;
#003
#004 template <class T1, class T2, class T3>
#005 T1 sum(T2 op1, T3 op2)
#006 {
#007   cout << "generic form" << endl;
#008   return static_cast<T1>(op1+op2);
#009 }
#010
#011 template<> double sum(float, float);
#012 //上一行在 VC6 竟然可以通過，差勁。
#013 //上一行在 bcb4 出現 e2423: explicit specialization or instantiation
#014 //                      of non-existing template 'sum'
#015 //上一行在 G++ 出現 : template-id `sum<>' for `sum<>(float, float)'
#016 //                 does not match any template decaration
#017
#018 // T1 明白指定爲 double, T2 推導爲 float, T3 推導爲 float
#019 template<> double sum<double>(float op1, float op2)
#020 {
#021   cout << "specialization form1" << endl;
#022   return static_cast<double>(op1+op2);
#023 }
#024
#025 // T1, T2, T3 明白指定爲 int, char, char
#026 template<> int sum<int, char, char>(char op1, char op2)
#027 {
```

```
#028  cout << "specialization form2" << endl;
#029  return static_cast<int>(op1+op2);
#030 }
#031
#032 void main()
#033 {
#034 int i=5;
#035 char c='a';
#036 float f=4.5;
#037 double d=6.5;
#038
#039  cout << sum<int>(i, i) << endl;      // generic form  10
#040  cout << sum<double>(f, f) << endl;    // specialization form1  9
#041  cout << sum<int>(c, c) << endl;      // specialization form2  194
#042 }
```

■C++ Primer p554

主題：function try block

測試結果：VC6[x]　BCB4[x]　G++[o]

實例：

```
#001 #include <iostream>
#002 using namespace std;
#003
#004 class popOnEmpty { /* ... */ };
#005 class pushOnFull { /* ... */ };
#006
#007 int main()
#008 try {
#009    throw popOnEmpty();
#010    throw pushOnFull();
#011    return 0;
#012 }
#013 catch ( pushOnFull ) {
#014    cout << "catch pushOnFull" << endl;
#015 }
#016 catch ( popOnEmpty ) {
#017    cout << "catch popOnEmpty" << endl;   // <-- 執行結果：此行。
#018 }
```

■C++ Primer　p564

主題：exception specification

測試結果：BCB4 表現佳，G++ 尚可，VC6 粗糙

實例：

```
#001 #include <iostream>
#002 using namespace std;
#003
#004 class popOnEmpty { /* ... */ };
#005 class pushOnFull { /* ... */ };
#006
#007 void func1() throw (pushOnFull);
#008
#009 void func1() throw (pushOnFull)
#010 {
#011  throw popOnEmpty();  // BCB4 warning: Throw expression violates
#012                       //  exception specification in function
#013                       //  func1() throw(pushOnFull)
#014                       // VC6 : no error, no warning
#015                       // G++ : no error, no warning
#016
#017  throw pushOnFull();  // BCB4 Warning : Unreachable code in function
#018                       // func1() throw(pushOnFull)
#019                       // VC6 : no error, no warning
#020                       // G++ : no error, no warning
#021 }
#022
#023 int main()
#024 {
#025  try {
#026      func1();
#027      return 0;
#028  }
#029  catch ( pushOnFull ) {
#030      cout << "catch pushOnFull" << endl;
#031  }
#032  catch ( popOnEmpty ) {
#033      cout << "catch popOnEmpty" << endl;
#034  }
#035 }
#036 // 執行結果：
#037 // BCB4: Abnormal program termination
#038 // G++ : none（我想是喚起了 C++ standard library function unexpected()，
#039 //       後者喚起 terminate()，其內喚起 abort()。
#040 // VC6 : catch popOnEmpty（我認爲 VC6 對於 exception spec. 的處理太粗糙）
```

■C++ Primer　p643 中

主題：直接在 class 內針對 static const integral data member 給予初值

（所謂 in-class initialization）

測試結果：VC6[x]　BCB4[o]　G++[o]

實例：

```
#001 #include <iostream.h>
#002
#003 class Account {
#004 public:
#005   static const int namesize = 16;   // <== in-class initialization
#006 };
#007
#008 const int Account::namesize;
#009
#010 void main()
#011 {
#012   cout << Account::namesize << endl;
#013 }
```

■C++ Primer p834

主題：class templates 內的 friend function

測試結果：VC6[x] BCB4[x] G++[o]

參考：請見稍後 ■C++ Primer p1090 對於 "VC6 的 friend functions" 的深入說明。

實例：

```
#001 #include <iostream>
#002 using namespace std;
#003
#004 // 以下的 forward declaration 非常重要，見 p834 L-8
#005 template <typename T> class A;
#006 template <typename T> ostream& operator<< (ostream& os, const A<T>& a);
#007
#008 template <typename T> class B;
#009 template <typename T> ostream& operator<< (ostream& os, const B<T>& b);
#010
#011 // 以下以 class A 和 class B 模擬 class Queue 和 class QueueItem 之間的關係
#012
#013 template <typename T>
#014 class A
#015 {
#016   // 以下的 <T> 非常重要，見 p834 L-6
#017   friend ostream& operator<< <T>(ostream& os, const A<T>& a);
#018 public:
#019   A (T i) : _i(i) {
#020           front = new B<T>(i);
#021           back  = new B<T>(i);
#022   }
#023   // 爲求完整，應再設計 dtor 以避免 memory leak.
#024
#025 private:
```

```
#026  T _i;
#027  B<T>* front;
#028  B<T>* back;
#029 };
#030
#031 template <typename T>
#032 ostream& operator<< (ostream& os, const A<T>& a)
#033 {
#034  os << '<' ;
#035  os << *(a.front) << ' ';
#036  os << *(a.back) << ' ' ;
#037  os << '>' << endl;
#038  return os;
#039 }
#040
#041 template <typename T>
#042 class B
#043 {
#044  friend ostream& operator<< <T>(ostream& os, const B<T>& b);
#045 public:
#046  B (T i) : _item(i) { }
#047 private:
#048  T _item;
#049 };
#050
#051 template <typename T>
#052 ostream& operator<< (ostream& os, const B<T>& b)
#053 {
#054  os << b._item;  // BCB4 error: _item is not accessible. why?
#055  return os;
#056 }
#057
#058 void main()
#059 {
#060  A<int>   a1(5);
#061  A<float> a2(5.4);
#062  A<char>  a3('a');
#063
#064  cout << a1 << a2 << a3 << endl;
#065  /* output :
#066     <5 5 >
#067     <5.4 5.4 >
#068     <a a >
#069  */
#070 }
```

■C++ Primer p842

主題：class templates（內含 nest types）的 friend functions。

測試結果：VC6[x] BCB4[x] G++[o]

實例：

```
#001 #include <iostream>
#002 using namespace std;
#003
#004 // 以下的 forward declaration 非常重要，見 p834 L-8
#005 template <typename T> class A;
#006 template <typename T> ostream& operator<< (ostream& os, const A<T>& a);
#007
#008 // 以下以 class A 和 class B 模擬 class Queue 和 class QueueItem 之間的關係
#009
#010 template <typename T>
#011 class A
#012 {
#013   // 以下的 <T> 非常重要，見 p834 L-6
#014   friend ostream& operator<< <T>(ostream& os, const A<T>& a);
#015
#016 private:
#017   class B   // nested class
#018   {
#019   public:
#020     B (T i) : _item(i) { }
#021     T _item;
#022   };
#023
#024 public:
#025   A (T i) : _i(i) {
#026           front = new B<T>(i);
#027           back  = new B<T>(i);
#028   }
#029   // 爲求完整，應再設計 dtor 以避免 memory leak.
#030
#031 private:
#032   T _i;
#033   B<T>* front;
#034   B<T>* back;
#035 };
#036
#037 template <typename T>
#038 ostream& operator<< (ostream& os, const A<T>& a)
#039 {
#040   os << '<' ;
#041   os << (a.front)->_item << ' ';
#042   os << (a.back)->_item  << ' ' ;
#043   os << '>' << endl;
```

```
#044
#045  return os;
#046 }
#047
#048 void main()
#049 {
#050  A<int>   a1(5);
#051  A<float> a2(5.4);
#052  A<char>  a3('a');
#053
#054  cout << a1 << a2 << a3 << endl;
#055  /* output :
#056     <5 5 >
#057     <5.4 5.4 >
#058     <a a >
#059  */
#060 }
```

■C++ Primer p844

主題：member templates

測試結果：VC6[o]　BCB4[o]　G++[o]

實例：

```
#001 #include <iostream>
#002 #include <string>
#003 #include <vector>
#004 using namespace std;
#005
#006 template <class T>
#007 class Queue {
#008 public:
#009       // class member template
#010       template <class Type>
#011       class CL
#012       {
#013          Type member;
#014          T mem;
#015       };
#016 public:
#017       // function member template
#018       template <class Iter>
#019       void assign( Iter first, Iter last )
#020       {
#021         cout << "Queue<T>::assign()" << endl;
#022       }
#023 };
#024
```

```
#025 void main()
#026 {
#027   // nested types
#028   Queue<int>::CL<char> c;
#029   Queue<int>::CL<string> s;
#030
#031   // instantiation of Queue<int>
#032   Queue<int> qi;
#033
#034   // instantiation of Queue<int>::assign( int *, int * )
#035   int ai[4] = { 0, 3, 6, 9 };
#036   qi.assign(ai, ai+4);        // output: Queue<T>::assign()
#037
#038   // instantiation of Queue<int>::assign( vector<int>::iterator,
#039   //                                     vector<int>::iterator)
#040   vector<int> vi(ai, ai+4);
#041   qi.assign(vi.begin(), vi.end());  // output: Queue<T>::assign()
#042 }
```

■C++ Primer p853

主題：separate compilation model for class template

推論：既然 BCB4 and G++ and VC6 都未能支援 separate compilation model for function templates，我想它們也一定都沒有支援 separate compilation model for class templates。但我未做測試（挺煩人 :)）

■C++ Primer p856

主題：class template specializations

測試結果：VC6[o] BCB4[o] G++[o]

■C++ Primer p861

主題：class template partial specializations

測試結果：VC6[x] BCB4[o] G++[o]

實例：

```
#001 #include <iostream>
#002 using namespace std;
#003
#004 // form 1
#005 template <class T, int hi, int wid>
#006 class Screen {
#007 public:
```

```
#008   void print() { cout << hi << ' ' << wid << " form1" << endl; }
#009 };
#010
#011 // form 2 (template partial specialization)
#012 template <class T, int hi>
#013 class Screen <T, hi, 80> {
#014 public:
#015   void print() { cout << hi << " form2" << endl; }
#016 };  // VC6 error C2989
#017
#018 // form 3
#019 template <class T, int hi>
#020 class Screen <T*, hi, 25> {
#021 public:
#022   void print() { cout << hi << ' ' << sizeof(T*) << ' '
#023                    << sizeof(T) << " form3" << endl; }
#024 };
#025
#026 int main()
#027 {
#028   Screen<int, 100, 40>  s1;
#029   Screen<int, 100, 80>  s2;
#030   Screen<int, 500, 25>  s3;
#031   Screen<char*, 300, 25> s4;
#032   Screen<double*, 400, 25> s5;
#033
#034   s1.print(); // output: 100 40 form1
#035   s2.print(); // output: 100 form2
#036   s3.print(); // output: 500 25 form1
#037   s4.print(); // output: 300 4 1 form3
#038   s5.print(); // output: 400 4 8 form3
#039   return 0;
#040 }
```

■C++ Primer p904

主題：using declaration 可將 base class 內任何一個具名的 member
（條件是 accessible）放進 derived class scope 內。

測試結果：VC6[x] BCB4[o] G++[x]
但如果將下例的 using declaration 移到 Shy::mumble() 之前，則 VC6[o]

實例：

```
#001 #include <iostream>
#002 #include <string>
#003 using namespace std;
#004
#005 class Diffident {
```

```
#006 public:
#007    void mumble (int softnes)
#008    { cout << "Diffident::mumble" << endl; };
#009 };
#010
#011 class Shy : public Diffident {
#012 public:
#013    void mumble(string whatYaSay)
#014    { cout << "Shy::mumble" << endl; };
#015    using Diffident::mumble;
#016 };
#017
#018 void main()
#019 {
#020 Diffident d;
#021 Shy s;
#022 string str("jjhou");
#023
#024  d.mumble(5);    // Diffident::mumble
#025  s.mumble(5);    // should be "Diffident::mumble"
#026  s.mumble(str);   // Shy::mumble
#027 }
```

■C++ Primer p940

主題：設計 "virtual" new operator（亦即 clone）時所需的一個技術：如果虛擬函式的 base instance 傳回 'A' class type（或爲指標，或爲 reference），那麼虛擬函式的 base instance 可以傳回 'A' type 或 'A' 的 publicly derived class（或爲指標，或爲 reference）

測試結果：VC6[x] BCB4[o] G++[o]

實例：

```
#001 class Query {
#002 public:
#003  virtual Query* clone() = 0;
#004 };
#005
#006 class NameQuery : public Query {
#007 public:
#008  virtual NameQuery* clone() { return new NameQuery(*this); }
#009  // VC6 error C2555: 'NameQuery::clone' : overriding virtual function
#010  //   differs from 'Query::clone' only by return type or calling convention
#011  //   see declaration of 'Query'
#012 };
#013
```

```
#014 void main()
#015 {
#016  Query* pq  = new NameQuery();
#017  Query* pq2 = pq->clone();
#018
#019  NameQuery* pnq  = new NameQuery();
#020  NameQuery* pnq2 = pnq->clone();
#021 }
```

■C++ Primer p1090

主題：friend operator<<

測試結果：VC6[x]　BCB4[o]　G++[o]

注意：如果使用 <iostream.h> 而不是 <iostream>，在 VC6 中
運用 friend 就比較沒有問題。但如果這麼做的話，由於下例
用到 <string>，一定得 using namespace std; 而這在 VC6 中似乎
導至暗中含入 <iostream>，於是與 <iostream.h> 起衝突。總之，
挖東補西，很煩。VC6 在這主題上表現不佳。

實例：

```
#001 #include <iostream>
#002 #include <string>
#003 using namespace std;
#004
#005 class WordCount {
#006    friend ostream& operator<<(ostream&, const WordCount&);
#007 public:
#008    WordCount( string& word, int cnt=1 )
#009    : _word(word), _occurs(cnt)
#010    { };
#011 private:
#012    string _word;
#013    int _occurs;
#014 };
#015
#016 ostream& operator <<( ostream& os, const WordCount& wd )
#017 {   // format: <occurs> word
#018     os << "< " << wd._occurs << " > " << wd._word;  // VC error!
#019     return os;
#020 }
#021
#022 void main()
#023 {
#024  string s1("Hello");
#025  string s2("jjhou");
#026  WordCount w1(s1, 5);
```

```
#027   WordCount w2(s2, 7);
#028   cout << w1 << endl;   // < 5 > Hello
#029   cout << w2 << endl;   // < 7 > jjhou
#030 }
```

■C++ Primer p1126 下

主題：list<T> 的 object initialization

測試結果：VC6[o] BCB4[x](有瑕疵) G++[o]

實例：

以下這行可通過 VC6 和 G++

```
list<int> ilist_result(ilist.size());
```

爲了在 BCB4 中通過，需改爲：

```
int i = ilist.size();
list<int> ilist_result(i, -1);
```

■C++ Primer p1156~p1157

主題：generic algorithms max() 和 min()

測試結果：VC6 所附的 STL 竟未支援 max 和 min 這兩個 generic algorithms。

聯想：試看這個程式：

```
(01) #include <iostream>
(02) #include <string>
(03) using namespace std;
(04)
(05) template <typename T>
(06) T min( T v1, T v2)      // <-- note
(07) {
(08)   return (v1 < v2 ? v1 : v2);
(09) }
(10)
(11) class Rect {
(12)  friend ostream& operator<<(ostream& os, Rect& r);
(13) public:
(14)  Rect(int i) : _i(i) {  }
(15)  bool operator<(Rect& rhs)
(16)    { return (this->_i < rhs._i ? true : false); }
(17) private:
(18)  int _i;
(19) };
(20)
(21) ostream& operator<<(ostream& os, Rect& r)
(22) {
(23)   os << r._i;
```

```
(24)   return os;
(25) }
(26)
(27) void main()
(28) {
(29)   cout << min( 17, 15) << endl;             // 15
(30)   cout << min(13.5, 13.57) << endl;         // 13.5
(31)   cout << min('a', 'e') << endl;            // a
(32)   cout << min("jjhou", "allan") << endl;    // allan
(33)
(34)   Rect r1(6), r2(3), r3(9);
(35)   cout << r1 << r2 << r3 << endl;           // 639
(36)   cout << min(r1, r2) << endl;              // 3
(37) }
```

檢討：

一 此程式在 VC6 中失敗，但如果改爲 #include<iostream.h> 並移除 using namespace std; 則成功，這是 VC6 的 bug！（見先前對 friend 的討論，C++ Primer p1090）

二 此程式在 BCB4 中失敗，錯誤訊息如下：

```
Error E2094 T1.CPP 36: 'operator<<' not implemented in type
'ostream' for arguments of type 'Rect' in function main()
```

這是因爲我們的 template function min() 和 BCB4 提供之 STL 中的 gemeric algorithm min() 名稱一樣。程式中喚起的其實是 STL 的 gemeric algorithm min()，而其 parameter list 內對於 parameters 的 constness 的要求，導至編譯器在進行 template argument deduction 時，不接受我所提供的 Rect。只要將程式中的 min() 改爲 mymin()，並將(12)和 (21) 的最後一個參數型別改爲 const Rect&，即可。

注意：有時候雖然你並未明白含入某些 header files，或是你並未明白使用 using namespace std; 編譯器卻會暗自加上。前者是由於 header files 一個含入一個…導至，後者請看 BCB4 的每一個 header files 的最後面，幾乎都有這一行：

```
#ifndef __USING_STD_NAMES__
  using namespace std;
#endif
```

三 此程式在 G++ 中獲得警告訊息如下：

```
t1.cpp:36: warning: initialization of non-const reference
'class Rect &' from rvalue 'Rect'
t1.cpp:22: warning: in passing argument 2 of
'operator <<(ostream &, Rect &)'
```

其道理與修改方法，和上述二相同：將 min() 改名爲 mymin() 即可。

心得：程式中不要出現任何命名與 C++ standard library 內的 components 同名。

■C++ Primer p1169
主題：generic algorithms random_shuffle()
測試結果：G++ 所附的 STL 對 random_shuffle() 的支援有問題

■Effective C++ 2/e p70
主題：如果 base class 的 operator= 係由編譯器產生，亦即所謂
default operator=，某些編譯器會拒絕讓你明白喚起 base class operator=，
這不是好現象。
測試結果：VC6[o] BCB4[x] G++[o]（BCB4 表現不佳，拒絕我們喚起上述 operator=）
實例：

```
#001 class Base {
#002 ...
#003   // no defined operator=
#004 };
#005
#006 class Derived : public Base {
#007 ...
#008   Derived& operator=(const Derived& rhs);
#009 };
#010
#011 Derived& Derived::operator=(const Derived& rhs)
#012 {
#013   if (this == &rhs) return *this;
#014
#015   // explicitly invoke Base operator=。VC6[o]  BCB4[x]  G++[o]
#016   Base::operator=(rhs);
#017   ...
#018 }

--- the end
```