Appendix B

C++ 保留字

C++ 保留了一些字彙給自己，以及 C++ 函式庫使用。在程式裡的宣告，我們不可以使用保留字作為變數名稱。關於保留字主要可以分三個範疇：關鍵字（keyword）、替代性標記（alternative token），以及 C++ 函式庫保留名稱（reserved name）。

C++ 關鍵字

關鍵字是識別字，它是組成程式語言的字彙。關鍵字不得用於如變數等其它用途。表 B.1 為 C++ 的關鍵字。以粗體字表示者，也是 ANSI C99 標準的關鍵字。其中的斜體字是 C++11 的關鍵字。

表 B.1 C++ 關鍵字

asm	**auto**	bool	**break**	**case**
catch	**char**	class	**const**	const_cast
continue	**default**	delete	**do**	**double**
dynamic_cast	**else**	**enum**	explicit	export
extern	false	**float**	**for**	friend
goto	**if**	**inline**	**int**	**long**
mutable	namespace	new	operator	private
protected	public	**register**	reinterpret_cast	**return**
short	**signed**	**sizeof**	**static**	static_cast
struct	**switch**	template	this	throw
true	try	**typedef**	typeid	typename
union	**unsigned**	using	virtual	**void**
volatile	wchar_t	**while**		

替代性標記

除了關鍵字，C++ 對於一些運算子，也可以使用英文字的方式呈現，稱為替代性標記。這些標記也被 C++ 保留起來了。表 B.2 列出英文字替代性標記，以及其所表示的運算子。

表 B.2 C++ 保留的替代性標記及其意義

標記	意義
and	&&
and_eq	&=
bitand	&
bitor	|
compl	~
not	!
not_e	!=
or	||
or_eq	|=
xor	^
xor_eq	^=

C++ 函式庫保留名稱

編譯程式不會讓我們使用關鍵字或替代性標記作為使用的名稱。此外，這裡還有另外一個被禁止使用之名稱的類別，但是不會作絕對的使用保護，此稱為保留名稱（reserved name）。這些名稱會保留給 C++函式庫使用。如果你使用其中的一個名稱作為識別字，會發生什麼事情是未知的。這也就是說，它可能會造成編譯程式錯誤；它可能會產生一個警訊；它可能會造成程式不正確地執行，或者它也有可能完全不會造成任何問題。

C++ 語言保留函式庫標頭檔中所使用的巨集（macro）名稱。如果程式引入某一個標頭檔，那麼我們就不可以使用標頭檔內（或者是標頭檔所引入之標頭檔，以此類推）定義的巨集名稱來作其它用途。舉例來說，如果我們直接或間接地引入標

頭檔 <climits>，就不應該使用 CHAR_BIT 作為識別字，因為這個名稱已經在那個標頭檔中使用了。

C++ 語言保留名稱的開頭會使用兩個底線或者是單一底線，後面接著一個大寫字母；此外，C++ 保留開頭為單一底線的名稱作為全域變數使用。因此，在任何情況，請不要建立如 __gink 或 __Lynx 的名稱，或者是在全域名稱空間中使用如 _lynx 這樣的名稱。

C++ 語言保留名稱會在函式庫標頭檔案中，以外部連結方式加以宣告。對於函數而言，這會包含函數簽名（名稱與參數列表）。舉例來說，假設我們使用這段程式碼：

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

在這種情況下，函數簽名 tan(double) 會被保留起來。而這表示在我們的程式中，不能夠宣告具有此原型的函數：

```
int tan(double);  // don't do it
```

這個沒有與函式庫裡會回傳 double 型態的 tan() 原型匹對，不過它與函數簽名部分匹對了。然而，下面的原型是可以接受的：

```
char * tan(char *);  // ok
```

這是因為雖然它與 tan() 識別字匹對，但它與函數簽名並沒有匹對。

具有特殊意義的識別字

C++ 社群不喜歡再加入新的關鍵字，因為它們可能會與現存的程式碼衝突。這也是為什麼委員會重新調整 auto 關鍵字並提供更多的類似的項目，如 virtual 和 delete。C++11 為了避免加入新的關鍵字，以另一種方法實現之，那就是將讓識別字具有特殊的意義。這些識別字如 override 和 final 並不是關鍵字，是用來呈現程式語言的特性。編譯程式將使用上下文，以決定它們是用來當作一般的識別字，或是用來呈現語言的特性。

```
class F
{
    int final; // #1
public:
...
    virtual void unfold() {...} = final; // #2
};
```

在此處的#1 那一行被用來當作一般識別字，而在#2 那一行被用來當作語言的特性。這兩種用法彼此之間是沒有衝突的。

同時 C++ 有許多的識別字常出現在程式中，但它們不是保留字。這些包括標頭檔名稱、庫存函數名稱、main、以及程式開始執行時所需要的函數名稱。只要避免名稱空間的衝突，您可以使用這些識別字做為其它的用途，雖然沒有理由要這樣做。我們要避免以下的用法，因為它缺乏一般常識。

```
// allowable but silly
#include <iostream>
int iostream(int a);
int main ()
{
    std::
cout << iostream(5) << '\n';
    return 0;
}
int iostream(int a)
{
    int main = a + 1;
    int cout = a -1;
    return main*cout;
}
```