國立屏東商業技術學院 資訊工程系 物件導向程式設計

3. C++基礎語法

由於C++語言是奠基於C語言之上，因此我們不針對基礎的語法進行介紹，僅就C++不同於C語言之處，做一要簡要的介紹。在開始之前，先讓我們瞭解C++語言與C語言的第一個差異：標準輸入與輸出。

3.1 標準輸入與輸出

標準輸入與輸出(standard input/output)大概是許多學過C語言的同學，在學習C++時最不能適應的地方。其實C++的做法讓輸出與輸出變得更為簡單，只要用習慣就會喜歡上這種寫法。在使用C語言時，由於printf在輸出資料時必須自行指定型態(透過format specifier)可能因程式設計師的指定錯誤，而導致錯誤的輸出結果。但在C++中的cout會自動判斷變數型態，因此較為安全與方便。

原本在C語言中，以#include <stdio.h>所載入的標準輸出入標頭檔，在C++中改成使用#include <iostream>與using namespace std並且以cout進行輸出，以cin取得使用者的資料。

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

int main()
{
   int a;
   char s [100];

   cout << "This is a sample program." << endl;

   cout << endl;

   cout << "Type your age : ";
   cin >> a;

   cout << "Type your name: ";
   cin >> s;

   cout << endl;

   cout << "Hello " << s << " you're " << a << " old." << endl;
   cout << endl << endl << "Bye!" << endl;

   return 0;
}
```

最後，要提醒同學，原本在C語言中的printf()與scanf()函式，在C++中仍可以使用。不過在我們的例子中，將儘量使用新的cout與cin來代替。

3.2 命名空間與函式標頭檔

C++語言使用命名空間(namespace)來管理在其函式/類別庫中眾多的識別字名稱，其中std是我們已經使用過的一個namespace你可以試著將程式中using namespace std;這行移除，再編譯程式看看，是不是會發現許多編譯的錯誤？其原因是包含endl, cout等，皆命名於std這個命名空間中，如果不先行載入此命名空間，則每次使用時都必須以std::endl, std::cout等方式清楚說明欲使用的識別字位於哪個命名空間中。除此之外，我們也可以宣告自己的namespace:

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

namespace first
{
   int a;
   int b;
}

namespace second
{
   double a;
   double b;
}

int main ()
{
   first::a = 2;
   first::b = 5;

   second::a = 6.453;
   second::b = 4.1e4;

   cout << first::a + second::a << endl;
   cout << first::b + second::b << endl;

   return 0;
}
```

若是要使用原本C語言的各種函式，其對應的標頭檔案當然也必須載入，不過要注意以下兩點：

1. 其它需要載入的標頭檔，仍使用#include載入，但.h的副檔名已不再使用
2. 傳統的C語言函式標頭檔，則以字元c開頭

```
using namespace std; // 載入標準函式庫
#include <iostream>  // 載入與輸出輸入相關
#include <cmath>     // 載入傳統的C語言函式庫標頭檔

int main ()
```

```
{

  double a;
  a = 1.2;
  a = sin (a);

  cout << a << endl;
  return 0;
}
```

3.3 變數與常數

3.3.1 變數

- C++語言變數命名具備以下規定：
  - 只能使用英文大小寫字母、數字與底線(_)
  - 不能使用數字開頭
  - 大寫與小寫字元將視為不同字元
  - 不能與C++語言的保留字相同
  - 在C++的實作中，以一個底線開頭後接一個大寫字母，或是以兩個底線開頭的變數，被保留做為C++的實作用途(供編譯器及其相關資源使用)。因此，使用這樣的命名並不會造成compiler的錯誤，但有可能造成執行上的錯誤。

C++語言共有以下84個保留字:

alignas	alignof	and	and_eq	asm	auto	bitand
bitor	bool	break	case	catch	char	char16_t
char32_t	class	compl	const	constexpr	const_cast	continue
decltype	default	delete	do	double	dynamic_cast	else
enum	explicit	export	extern	false	float	for
friend	goto	if	inline	int	long	mutable
namespace	new	noexcept	not	not_eq	nullptr	operator
or	or_eq	private	protected	public	register	reinterpret_cast
return	short	signed	sizeof	static	static_assert	static_cast
struct	switch	template	this	thread_local	throw	true
try	typedef	typeid	typename	union	unsigned	
using	virtual	void	volatile	wchar_t	while	xor
xor_eq						

變數可以在任何地方加以宣告，只要在第一次使用前完成宣告即可。

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

int main ()
```

```
{
   double a;

   cout << "Hello, this is a test program." << endl;

   cout << "Type parameter a: ";
   cin >> a;

   a = (a + 1) / 2;

   double c;

   c = a * 5 + 1;

   cout << "c contains      : " << c << endl;

   int i, j;

   i = 0;
   j = i + 1;

   cout << "j contains      : " << j << endl;

   return 0;
}
```

我們可以利用這個C++的特性，將程式寫的更有彈性。請參考下面這個程式：

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

int main ()
{
   double a;

   cout << "Type a number: ";
   cin >> a;

   {
      int a = 1;
      a = a * 10 + 4;
      cout << "Local number: " << a << endl;
   }

   cout << "You typed: " << a << endl;

   return 0;
}
```

此外C++還允許我們在宣告變數時，使用別的變數做為初始值設定的一部份，請參考下面這個程式：

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

int main ()
{
   double a = 12 * 3.25;
   double b = a + 1.112;

   cout << "a contains: " << a << endl;
   cout << "b contains: " << b << endl;

   a = a * 2 + b;

   double c = a + b * a;

   cout << "c contains: " << c << endl;

   return 0;
}
```

我們也可以在迴圈中宣告變數，其生命週期就僅限於迴圈內，請參考下面的程式：

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

int main ()
{
   int i;                       // Simple declaration of i
   i = 487;

   for (int i = 0; i < 4; i++)  // Local declaration of i
   {
      cout << i << endl;        // This outputs 0, 1, 2 and 3
   }

   cout << i << endl;           // This outputs 487

   return 0;
}
```

若在區域內的變數與某個全域變數名稱相同，只要加上 “ :: “ 就可以在區域內使用，請參考下面的程式：

```
using namespace std;
```

```
#include <iostream>

double a = 128;

int main ()
{
   double a = 256;

   cout << "Local a:  " << a   << endl;
   cout << "Global a: " << ::a << endl;

   return 0;
}
```

3.3.2 常數

- 以const宣告之變數，其值不能被變更，稱為常數。例如：

```
const double radius = 5.5;
```

- 也可以用#define這一個preprocessor directive來定義常數。例如:

```
#define PI 3.1415926
```

3.3.3 新的變數初始值給定方式

C++提供了新的變數初始值給定的方法,

```
type valName = value;
type valName(value);
type valName = {value};
type valName{value};
```

其中最後一種宣告的方法，是在C++11的標準才開始提供，所以在編譯時必須要使用-std=gnu++11的選項才能順利的編譯。請參考下面的例子：

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

int main()
{
  int i = 3;
  int j (4);
  int k = {5};
```

```
  int l {6};

  cout << "i=" << i << endl;
  cout << "j=" << j << endl;
  cout << "k=" << k << endl;
  cout << "l=" << l << endl;

  return 0;
}
```

上述這個程式的編譯與執行畫面如下：

```
[03:39 user@ws ch3]$ g++ -std=gnu++11 newVarInitial.cpp
[03:39 user@ws ch3]$ ./a.out
i=3
j=4
k=5
l=6
[03:39 user@ws ch3]$
```

新的宣告方式，還提供了型態安全的檢查，我們將上面的程式修改如下：

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

int main()
{
  int i = 3.5;
  int j (4.5);
  int k = {5.5};
  int l {6.5};
  char x {332};

  cout << "i=" << i << endl;
  cout << "j=" << j << endl;
  cout << "k=" << k << endl;
  cout << "l=" << l << endl;

  return 0;
}
```

其編譯結果如下：

```
[03:52 user@ws ch3]$ g++ -std=gnu++11 newVarInitial2.cpp
newVarInitial2.cpp: In function ‘int main()’:
```

```
newVarInitial2.cpp:8:15: warning: narrowing conversion of ‘5.5e+0’ from
‘double’ to ‘int’ inside { } [-Wnarrowing]
newVarInitial2.cpp:9:13: warning: narrowing conversion of ‘6.5e+0’ from
‘double’ to ‘int’ inside { } [-Wnarrowing]
newVarInitial2.cpp:10:14: warning: narrowing conversion of ‘332’ from ‘int’
to ‘char’ inside { } [-Wnarrowing]
newVarInitial2.cpp:10:14: warning: overflow in implicit constant conversion
[-Woverflow]
[03:53 user@ws ch3]$ ./a.out
i=3
j=4
k=5
l=6
[03:53 user@ws ch3]$
```

注意到了嗎？新的「{}」方式還會進行型態安全的檢查。我們將「{}」這種宣告初始值的方法稱為List Initialization。當「{}」內沒有提供初始值時，編譯器會以0做為初始值。

最後C++還提供了一種新的宣告方式：

```
auto valName = value;
auto valName (value);
auto valName = {value};
auto valName {value};
```

使用auto是讓編譯器視變數的初始值，自動決定適切的資料型態。

3.4 資料型態

C++語言提供多種資料型態，包含基本型態(fundamental type)與複合資料型態(compound type)兩類。本章僅就基本型態做一說明，複合型態請參閱後續章節。

3.4.1 整數型態

C++語言一共有以下8種整數型態：

- Standard signed integer types (標準符號整數型態)
  - short int
  - int
  - long int
  - long long int
- Standard unsigned integer types (標準無符號整數型態)
  - unsigned short int
  - unsigned int
  - unsigned long int
  - unsigned long long int

<note important>

型態也可以縮寫?

我們在宣告整數型態的變數時，可以將int省略，例如:可以short來代表short int；以unsigned代表unsigned int；以long代表long int；long long代表long long int；以及unsigned short代表unsigned short int；unsigned long代表unsigned long int；unsigned long long代表unsigned long long int；

</note> 在整數型態的數值範圍方面，都是取決於其所使用的記憶體空間。由於不同平台上可能會有差異，C++語言僅提供規範，實際情形由各平台上的實作決定。因此，在一個平台上撰寫程式時，我們通常會使用以下的程式，先行瞭解各型態所佔的空間：

```
#include <iostream>
#include <climits> // use limits.h

int main()
{
  using namespace std;
  int n_int = INT_MAX;       // initialize n_int to max int value
  short int n_short = SHRT_MAX;  // symbols defined in climits file
  long int n_long = LONG_MAX;
  long long int n_llong = LLONG_MAX;

  // sizeof operator yields size of type or of variable
  cout << "int is " << sizeof (int) << " bytes." << endl;
  cout << "short is " << sizeof n_short << " bytes." << endl;
  cout << "long is " << sizeof n_long << " bytes." << endl;
  cout << "long long is " << sizeof n_llong << " bytes." << endl;
  cout << endl;

  cout << "Maximum values:" << endl;
  cout << "int: " << n_int << endl;
  cout << "short: " << n_short << endl;
  cout << "long: " << n_long << endl;
  cout << "long long: " << n_llong << endl << endl;

  cout << "Minimum int value = " << INT_MIN << endl;
  cout << "Bits per byte = " << CHAR_BIT << endl;

  return 0;
}
```

以我們的ws.csie.npic.edu.tw工作站為例，limits.cpp的執行結果如下：

```
[09:52 junwu@ws ch4]$ g++ limits.cpp
[09:52 junwu@ws ch4]$ ./a.out
int is 4 bytes.
```

```
short is 2 bytes.
long is 8 bytes.
long long is 8 bytes.

Maximum values:
int: 2147483647
short: 32767
long: 9223372036854775807
long long: 9223372036854775807

Minimum int value = -2147483648
Bits per byte = 8
[09:52 junwu@ws ch4]$
```

上述程式，主要使用了sizeof()函式以及位在climits標頭檔中的定義，有關climits標頭檔，可至/usr/include/c++目錄下查詢。下表為climits中所定義的部份常數：

Symbolic Constant	**Represents**
CHAR_BIT	Number of bits in a char
CHAR_MAX	Maximum char value
CHAR_MIN	Minimum char value
SCHAR_MAX	Maximum signed char value
SCHAR_MIN	Minimum signed char value
UCHAR_MAX	Maximum unsigned char value
SHRT_MAX	Maximum short value
SHRT_MIN	Minimum short value
USHRT_MAX	Maximum unsigned short value
INT_MAX	Maximum int value
INT_MIN	Minimum int value
UINT_MAX	Maximum unsigned int value
LONG_MAX	Maximum long value
LONG_MIN	Minimum long value
ULONG_MAX	Maximum unsigned long value
LLONG_MAX	Maximum long long value
LLONG_MIN	Minimum long long value
ULLONG_MAX	Maximum unsigned long long value

Tab. 1: 在climits中定義的常數

3.4.2 浮點數

C++語言中有3種符點數的型態：float, double與long double，分別實作了IEEE 754當中的單精確度、倍精確度與擴充精確度：

- float: 單精確度浮點數(single-precision floating-point)
- double: 倍精確度浮點數(double-precision floating-point)
- long double: 擴充精確度符點數(extended-precision floating-point)

3.4.3 字元型態

所謂的字元型態就是用以表示文字、符號等資料，在C/C++語言中只有一種字元型態：

- char

在不同的系統中，字元的數值可能會代表不同意義，視其所採用的字元集(character set)而定。現行最常見的字元集為ASCII(American Standard Code for Information Interchange)，請參考Wikipedia關於ASCII的說明。

既然char型態就是整數，那可不可以再配合unsigned使用呢？因為char型態的整數數值是用以對應特定的字元集(如ASCII)，而每個字元集都有其可表達的字元個數要求，C++語言會自動將char定義為singed或unsigned以符合字元集的需求。因此我們通常不會特別在char前加上unsigned，但是，如果您有某些較小的整數資料要處理，就可以考慮使用char來代替int，因為int為32 bits，甚至short int也要使用到16 bits，若您只需要處理一些介於-128到127之間的數值，那您就可以考慮改用char來代替int，或是宣告為unsigned char來處理那些介於0到255的正整數資料。

3.4.4 布林型態

布林型態為C++所新增的資料型態，其名稱為bool，一個bool型態的資料只可能有true或false兩種可能的數值。與傳統的C語言一樣，若你要以整數來表達bool型態的值，則以0表示false，其它非0的值皆視為true。

```
bool isQuit = false;

int continueProcess = true; //將true轉換為1
```

3.4.5 資料型態轉換

如果在程式碼中，我們想要把某個數值之型態加以轉換，可以使用顯示型態轉換(explicit conversion)來對數值進行強制的轉型(casting)。使用的方法很簡單，只要在想要轉型的數值前加上一組()其中指定欲轉換的型態即可，例如:

```
int x;
long int y;

y=(long)x;
y = (long)(x+837);
x = (int)sizeof(int);
```

3.5 運算子

3.5.1 算術運算子

算術運算子就如同我們一般在數學式子中，所使用的運算符號，例如加減乘除等。table 2為C/C++語言支援的算術運算子：

operator	意義	unary/binary
+	正	unary
-	負	unary
+	加法	binary
-	減法	binary
*	乘法	binary
/	除法	binary
%	餘除	binary

Tab. 2: Arithmetic Operators

3.5.2 指定運算子

等號「=」被稱為C/C++語言的指定運算子(assignment operator)，用以將等號右方的值指定(assign)給等號左方，我們將其稱為是右關聯(right associativity)的運算子。例如：

- i = 5;
- j = i;
- k = 10 * i + j;

要注意的是，若等號左右兩邊的資料型態不一致時，C/C++語言會進行自動的型態轉換，例如:

假設宣告有：

```
int i;
float j;

i = 83.34f; // i = 83

j = 136;    // j=136.0
```

在一個運算式中，有時可以出現一個以上的等號，例如:

- i = j = k = 0;

等同於

1. k=0;
2. j=(k=0);
3. i=(j=(k=0));

請考慮以下的程式片段，想想看其輸出結果為何？

```
i = 1;
k = 1 + (j=i);
```

```
cout << i << ", " << j << ", " << k << endl;
```

3.5.3 複合指定運算子

假設i=3，考慮以下的運算式：

- i = i + 2;

其結果是先進行等號右邊的運算，得到結果為5後，將數值5給定到等號左邊的變數i，因此，最後i的值等於5。針對這種情形，C/C++語言提供複合指定(compound assignment)運算子，例如「+=」，上面的運算式可重寫為:

- i += 2;

其它常見的複合指定運算子，還有 -=, *=, /=與 %=。這些複合指定運算子為右關聯，請考慮下列的運算式：

- i += j += k;

等同於

- i += (j += k);

3.5.4 遞增與遞減運算子

當我們需要將某個變數的值遞增時，可以寫做：

i = i + 1; 或 i += 1;

但是C/C++語言還提供++與--這兩個運算子，分別是

- ++，遞增(increment)運算子
- --，遞減(decrement)運算子

我們可以把i=i+1或i+=1改寫為：

i++;

同理，還有i- -可以遞減i的數值。但是++與- -可以選擇為prefix operator或postfix operator，視其寫在變數的前面或後面而定。放在前面，例如++i，會先遞增i的數值，然後再傳回新的i的數值；但寫在後面，例如i++則會先傳回i現有的數值，然後才將i的值遞增。

考慮以下的程式碼，想想看輸出的結果為何？

```
i=1;
cout << ++i << endl;
cout << i << endl;
cout << i++ << endl;
cout << i << endl;
```

3.5.5 sizeof運算子

sizeof運算子可以計算型態或變數等的記憶體空間，其用法有二：在sizeof後接一組括號並在其中放置要計算空間的對象，或者直接在sizeof後放置欲計算空間的對象(無須括號)，請參考下例：

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

int main()
{
  char *month[]= {"January", "February", "March", "April", "May", "June",
      "July", "August", "September", "October", "November", "December" };

  cout << "Size of char type = " << sizeof(char) << " byte." << endl;
  cout << "Size of month array = " << sizeof month << " bytes." << endl;

  return 0;
}
```

3.5.6 關係運算子

關係運算子是一個二元的運算子，用以判斷兩個運算元(數值、函式、變數或運算式)之間的關係，其可能的關係有：大於、小於、等於、或不等於。C/C++語言提供以下的關係運算子，如table 3。

符號	範例	意義
>	a > b	a 是否大於 b
<	a < b	a 是否小於 b
>=	a >= b	a 是否大於或等於 b
<=	a <= b	a 是否小於或等於 b

Tab. 3: Relational Operators

雖然我們已經提過。C/C++語言將數值0視為false。並將其它所有非0的數值視為true。但關係運算子會以數值0代表運算結果為false。以數值1代表true。還要注意關係運算子較算術運算子的優先順序低，所以像是 x + y < i - j 等同於 (x + y) < (i - j)。在C/C++語言中。x < y < z等同於 (x < y) < z。因為關係運算子為左關聯。假設x=1, y=3, z=5。 x<y<z ⇒ (x < y) <z ⇒ 1 < z ⇒ 1。要注意的是C++已提供了bool型態，因此也可以直接使用true或false來代表運算的結果。

3.5.7 相等運算子

相等運算子是一個binary運算子，用以判斷兩個運算元(數值、函式、變數或運算式)之值是否相等。C/C++語言提供以下的關係運算子，如table 4。

符號	範例	意義
==	a == b	a 是否等於 b
!=	a != b	a 是否不等於 b

Tab. 4: Equality Operators

同樣地，相等運算子會以數值0代表運算結果為false，以數值1代表true，還要注意相等運算子與關係運算子一樣，其優先順序都較算術運算子來得低。

<note important>是 ==， 不是 = 。 千萬不要將比較兩數是否相等的==寫成=，這實在是一個常常會遇到的錯誤!建議您以後如果遇到程式執行結果錯誤，但找不出任何問題時，試試檢查一下所有的 = 與 ==，有很高的機會可以改正您的程式。</note>

3.5.8 邏輯運算子/Logical Operator

邏輯運算子共有以下三種，如table 5。

符號	意義	unary or binary
!	NOT	unary
&&	AND	binary
||	OR	binary

Tab. 5: Logical Operators

其運算結果請參考table 6的真值表：

X	Y	NOT X	X AND Y	X OR Y
0	0	1	0	0
0	1		0	1
1	0	0	0	1
1	1		1	1

Tab. 6: Truth Table

假設變數score代表c語言的修課成績，以下的邏輯運算即為檢查成績是否介於0~100：

```
((score >= 0) && (score <=100))
```

3.5.9 條件運算子

C/C++語言還有提供一種特別的運算子，稱為條件運算子(conditional operator)，可依條件決定運算式的傳回值，其語法如下：

```
expression1 ? expression2 : expression3
```

運算式的運算結果expression1的值為true(非0的數值)或false(數值0)而定，當expression1為true時，傳回expression2的值；否則當expression1為false時，傳回expression3的值。事實上，這等同於下面的if敘述：

```
if (expression1)
   result = expression2;
else
   result = expression3;
```

條件式敘述有可能是因為像「如果expression為真則…. 否則….」這樣的述敘，在程式中出現的機會很高的緣故吧！請參考以下的應用：

```
int x=1, y=2, z;
if(x>y)
   z=x;
else
   z=y;
```

上面這段程式碼是令z為x與y兩者中較大的值，如果以條件運算式改寫，則只要寫成

```
z= x>y ? x: y;
```

即可，是不是簡化很多？下面這行程式，假設score為學生成績，則可以簡單地檢查score是否大於100，若超過100則以100分計。

```
score = score > 100 ? 100 : score ;
```

3.5.10 優先順序與關聯性

我們將C/C++的運算子之優先順序與關聯性彙整於table 7

運算子	符號
一元運算子	+(正)、-(負)□++□--□!(NOT)□sizeof
算術運算子(乘除)	*□/□%
算術運算子(加減)	+□-
關係運算子	>=□<=□>□<
相等運算子	==□!=
邏輯運算子	&&□||
條件運算子	?:
指定運算子	=□*=□/=□%=□+=□-=

Tab. 7: 各運算子的優先順序(由高至低)

3.6 條件式敘述

C++與C語言一樣，提供了兩種條件式敘述□if與switch□其用法與C語言並無二致，在此不予贅述。

3.7 迴圈

C++與C語言一樣，提供了for、while、do while等迴圈敘述，其用法與C語言並無二致，在此不予贅述。

3.8 陣列

C++與C語言一樣，提供了陣列用以管理相同型態的資料，其用法與C語言並無二致，在此不予贅述。

3.9 函式

C++與C語言一樣，提供了函式(function)，其用法與C語言大致相同，但C++允許函式的引數可以有預設的數值:

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

double test (double a, double b = 7)
{
   return a - b;
}

int main ()
{
   cout << test (14, 5) << endl;    // Displays 14 - 5
   cout << test (14) << endl;       // Displays 14 - 7

   return 0;
}
```

3.10 指標

C++與C語言一樣，提供了指標以存取特定的記憶體位址，其用法與C語言並無二致，在此不予贅述。

3.11 參考(Reference)變數

C++讓我們可以為某個變數，建立一個副本。

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

```
int main ()
{
   double a = 3.1415927;

   double &b = a;                              // b is a

   b = 89;

   cout << "a contains: " << a << endl;     // Displays 89.

   return 0;
}
```

在上述範例中，b被稱為參考變數(reference variable)，我們在宣告b時，使用&b=a來讓b成為a的分身。要注意的是，一但宣告後，不可以再改變其參考的對象。

參考變數也可以用在函式的引數宣告，例如下面的程式碼：

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

void change (double &r, double s)
{
   r = 100;
   s = 200;
}

int main ()
{
   double k, m;

   k = 3;
   m = 4;

   change (k, m);

   cout << k << ", " << m << endl;         // Displays 100, 4.

   return 0;
}
```

下面則是以指標的方式，將前述程式再實作一次：

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

void change (double *r, double s)
```

```
{
   *r = 100;
   s = 200;
}

int main ()
{
   double k, m;

   k = 3;
   m = 4;

   change (&k, m);

   cout << k << ", " << m << endl;        // Displays 100, 4.

   return 0;
}
```

下面的程式則又複雜了一點：

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

double &biggest (double &r, double &s)
{
   if (r > s) return r;
   else       return s;
}

int main ()
{
   double k = 3;
   double m = 7;

   cout << "k: " << k << endl;       // Displays  3
   cout << "m: " << m << endl;       // Displays  7
   cout << endl;

   biggest (k, m) = 10;

   cout << "k: " << k << endl;       // Displays  3
   cout << "m: " << m << endl;       // Displays 10
   cout << endl;

   biggest (k, m) ++;

   cout << "k: " << k << endl;       // Displays  3
```

```
   cout << "m: " << m << endl;       // Displays 11
   cout << endl;

   return 0;
}
```

這個程式蠻有趣的吧！

3.12 try catch

try與catch的機制，可以讓我們在try的程式區塊中throw結果給catch區塊處理，請參考下面的程式：

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

int main ()
{
   int a, b;

   cout << "Type a number: ";
   cin >> a;
   cout << endl;

   try
   {
      if (a > 100) throw 100;
      if (a < 10)  throw 10;
      throw a / 3;
   }
   catch (int result)
   {
      cout << "Result is: " << result << endl;
      b = result + 1;
   }

   cout << "b contains: " << b << endl;

   cout << endl;

   // another example of exception use:

   char zero []     = "zero";
   char pair []     = "pair";
   char notprime [] = "not prime";
   char prime []    = "prime";

   try
```

```
   {
      if (a == 0) throw zero;
      if ((a / 2) * 2 == a) throw pair;
      for (int i = 3; i <= sqrt (a); i++)
      {
         if ((a / i) * i == a) throw notprime;
      }
      throw prime;
   }
   catch (char *conclusion)
   {
      cout << "The number you typed is "<< conclusion << endl;
   }

   cout << endl;

   return 0;
}
```