C++ 中的 Lambda 表達式寫法

上課老師：莊啟宏

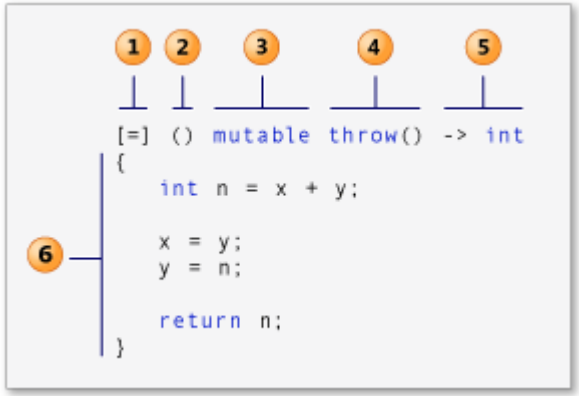
支援 Lambda Expression 的 C++ 編譯器

- 要開始學習 lambda expression 之前，要先準備支援 lambda expression 的編譯器，由於 lambda expression 是在 C++ 11 才加入的新語法，所以不見得每一種編譯器都有支援，以下這些是有支援 lambda expression 的編譯器版本：
- GCC 4.5：需要指定 -std= C++11 參數。
- Intel C++ Compiler 11：需要指定 /Qstd=c++0x 參數。
- Microsoft Visual C++ 2010（包含在 Visual Studio 2010 之中）

什麼是Lambda表達式

- MSDN上對lambda表達式的解釋：
  - 在 C++ 11 中，lambda 表達式（通常稱為 “lambda”）是一種在被調用的位置或作為參數傳遞給函數的位置定義匿名函數對象的簡便方法。 Lambda 通常用於封裝傳遞給算法或異步方法的少量代碼行。
- 看了這個解釋，相信大家已經理解 lambda 表達式是什麼。簡而言之， lambda 表達式就是一種定義函數的簡單的方法。

Lambda 的組件



- (1)擷取子句(也稱為lambda introducer在C++規格。)
- (2)參數清單選擇性。 (也稱為lambda declarator)
- (3)可變動規格選擇性。
- (4)例外狀況規格選擇性。
- (5)尾端傳回型別選擇性。
- (6)lambda 主體。

- Lambda expression 基本的用法如下：

```
[=] (int x) mutable throw() -> int
{
  // 函數內容
  int n = x + y;
  return n;
}
```

- [=]：lambda-introducer，也稱為 capture clause。所有的 lambda expression 都是以它來作為開頭，不可以省略，它除了用來作為 lambda expression 開頭的關鍵字之外，也有抓取（capture）變數的功能，指定該如何將目前 scope 範圍之變數抓取至 lambda expression 中使用，而抓取變數的方式則分為傳值（by value）與傳參考（by reference）兩種，跟一般函數參數的傳入方式類似，不過其語法有些不同，以下我們以範例解釋：

- [ ]：只有兩個中括號，完全不抓取外部的變數。
- [=]：所有的變數都以傳值（by value）的方式抓取。
- [&]：所有的變數都以傳參考（by reference）的方式抓取。
- [x, &y]：x 變數使用傳值、y 變數使用傳參考。
- [=, &y]：除了 y 變數使用傳參考之外，其餘的變數皆使用傳值的方式。
- [&, x]：除了 x 變數使用傳值之外，其餘的變數皆使用傳參考的方式。
- **這裡要注意一點，預設的抓取選項（capture-default，亦即 = 或是 &）要放在所有的項目之前，也就是放在第一個位置。**

- (int x)：lambda declarator，也稱為參數清單（parameter list）。定義此匿名函數的傳入參數列表，基本的用法跟一般函數的傳入參數列表一樣，不過多了一些限制條件：
- 不可指定參數的預設值。
- 不可使用可變長度的參數列表。
- 參數列表不可以包含沒有命名的參數。

- 參數清單在 lambda expression 中並不是一個必要的項目，如果不需要傳入任何參數的話，可以連同小括號都一起省略。

- mutable：mutable specification。加入此關鍵字可以讓 lambda expression 直接修改以傳值方式抓取進來的外部變數，若不需要此功能，則可以將其省略。

```
#include <iostream>
using namespace std; int main() {
    int m = 0;
    int n = 0;
    [&, n] (int a) mutable { m = ++n + a;}(4);
    cout << m << endl << n << endl;
}
```

結果：
5
0

- 因為變數 n 是以傳值方式擷取，其值在 Lambda 運算式呼叫之後會保持為 0。 可變規格(mutable) 允許n 在 lambda 內修改。

範例

- 值捕獲：直接獲取外部變量的值

```
1
2       int a = 456;
3       auto f = [a]() {cout << a << endl; };
4       f();
```

- 引用捕獲：捕獲外部變量的引用

```
1
2       auto z = [&a]() {cout << a << endl; };
3       z();
```

- 隱式捕獲：根據 lambda 函數體內部的變量情況,自動去捕獲外部變量,括號內用一個等號表示

```
1 auto c = [=]() {cout << a << endl; };
2       c();
```

- 使用mutable關鍵字修改被捕獲的變量

```
auto h = [=]()mutable { cout << ++a << endl; };
        cout << a << endl;   //輸出456,外部的值沒有被修改
        h();                 //輸出457,外部的值被修改
```

- **throw()**：例外狀況規格（exception specification）。指定該函數會丟出的例外，其使用的方法跟一般函數的例外指定方式相同。如果該函數沒有使用到例外的功能，則可以直接省略掉。

- **-> int**：傳回值型別（return type）。指定 lambda expression 傳回值的型別，這個範例是指定傳回值型別為整數（int），其他的型別則以此類推。如果 lambda expression 所定義的函數很單純，只有包含一個傳回陳述式（statement）或是根本沒有傳回值的話，這部分就可以直接省略，讓編譯器自行判斷傳回值的型別。

- { }：compound-statement，亦稱為 Lambda 主體（lambda body）。這個就是匿名函數的內容，就跟一般的函數內容一樣。

- 這是一個最簡單的 Hello World 範例。

```
#include <iostream>
using namespace std;
int main() {
  auto lambda = []() { cout << "Hello, Lambda" << endl; };
  lambda();
}
```

由於這裡的 lambda expression 並沒有需要傳入任何參數，所以可以連同小括號一起省略，改寫成這樣：

```
auto lambda = [] { cout << "Hello, Lambda" << endl; };
```

也可以在參數列加上 void，明確標示沒有傳入參數：

```
auto lambda = [](void) { cout << "Hello, Lambda" << endl; };
```

或是將傳回值的類型設為 void，明確標示這個函數沒有傳回值：

```
auto lambda = [](void) -> void { cout << "Hello, Lambda" << endl; };
```

作業一

```
int   factorial(int n) {
      int fact = 1;
      for (int i = 1; i <= n; ++ i)
          fact *= i; return fact;
}
```

- 請將上式改為 Lambda 表示式

作業二

- 請用 Lambda 表示式寫出當傳入兩個整數時可以傳出最大數是誰。