# 6-1 函數

隨著寫程式經驗愈來愈多，你會發現有些程式碼會不斷重複出現，就像例行性工作一樣，例如：求平方根、將資料排序、驗證帳號密碼.....等等。一次又一次的輸入這些程式碼會讓人很不耐煩。對於這些經常出現的程式碼片段，我們可以使用函數來把它們包裝起來。C/C++裡面的函數就像數學裡面的函數，例如：

$f(x)=2x^2+3x+4$

它有一個輸入：x，有一個輸出：f(x)。你給它一個輸入 3，它在運算後會給你一個輸出31；你給它另一個輸入 2，它會給你另一個相應的輸出18。不管你給的輸入是什麼，它都會很忠實的去完成該做的事 $2x^2 + 3x + 4$ ，並把結果輸出給你。

## 定義函數

以上面那個 f(x) 為例，我們可以這樣在 C++ 裡定義它。

```
int f(int x)
{
  int result = 2*x*x +3*x + 4;
  return result;
}
```

其架構如下：

```
回傳值型別 函數名稱(參數1型別 參數1名稱, 參數2型別 參數2名稱, ...)
{
    // 實作程式碼
    return 回傳值;
}
```

接下來我們就可以使用這個函數了。

```
#include <iostream>

using namespace std;

int f(int x)
{
  int result = 2*x*x +3*x + 4;
  return result;
}

int main()
{
    int n;
    int ans;

    n = 2;
    ans = f(n);
    cout << ans << endl;   // 18

    ans = f(3);
    cout << ans << endl;   // 31

    n = 5;
    cout << f(n) << endl;  // 69

    return 0;
}
```

請注意我們把 函數f 定義在 main() 的前面。如同變數在使用前要先宣告，函數也是一樣。

我們在第 17 行首次使用到 函數f，所以在這之前必須先知道 函數f 長什麼樣子。

如果把 函數f 定義在後面，在編譯時就會發生錯誤。

```
#include <iostream>

using namespace std;

int main()
{
    int n;
    int ans;

    n = 2;
    ans = f(n);  // f( ) 是什麼？往前看不到有人告訴我 f( ) 是什麼。
    cout << ans << endl;   // 18

    ans = f(3);
    cout << ans << endl;   // 31

    n = 5;
    cout << f(n) << endl;  // 69

    return 0;
}

// 定義在後面
int f(int x)
{
  int result = 2*x*x +3*x + 4;
  return result;
}
```

```
main.cpp: In function 'int main()':
main.cpp:11:11: error: 'f' was not declared in this scope
    ans = f(n);
          ^
```

# 宣告函數

有沒有注意到，前面我們一直說定義函數，而不是宣告函數(declare)。

我們以同一個 函數f 為例，宣告這個函數的作法為：

```
int f(int x);
```

或

```
int f(x);
```

宣告函數只要講清楚這幾個重點即可：

1. 函數名稱
2. 參數列 (每個參數的型別，可以沒有名字)
3. 回傳值型別

我們把上面的範例程式改成只有宣告試試。

```
#include <iostream>

using namespace std;

int f(int x);  // 宣告在這裡

int main()
{
```

```
    int n;
    int ans;

    n = 2;
    ans = f(n);    // 使用到 函數f
    cout << ans << endl;   // 18

    ans = f(3);    // 使用到 函數f
    cout << ans << endl;   // 31

    n = 5;
    cout << f(n) << endl;  // 69,  使用到 函數f

    return 0;
}
```

建置(build)這個程式時，出現了沒看過的錯誤。

```
main.cpp:13: undefined reference to `f(int)'
main.cpp:16: undefined reference to `f(int)'
main.cpp:20: undefined referenc
```

這個 `undefined reference to 'f(int)'` 是什麼意思呢？

我們的程式碼要經過「編譯(compile)」、「連結(link)」兩個步驟，才能生成最終的可執行檔。

在編譯階段，編譯器看到叫用(call)函數時，只會確認之前宣告過的函數

1. 名稱是否相符
2. 參數列的數量和型別是否相符
3. 回傳值型別是否相符

如果都符合，會在叫用函數的地方留個「空位」，然後編譯將會成功完成，進入連結階段。

在連結階段必須真的有一個函數被定義過，才能把這個函數「身體」所在的位置填入之前編譯階段留下的「空格」。

我們修改程式，在末端補上 函數f 的定義，即可順利建置。

```
#include <iostream>

using namespace std;

int f(int x);  // 宣告在這裡

int main()
{
    int n;
    int ans;

    n = 2;
    ans = f(n);    // 使用到 函數f
    cout << ans << endl;   // 18

    ans = f(3);    // 使用到 函數f
    cout << ans << endl;   // 31

    n = 5;
    cout << f(n) << endl;  // 69,  使用到 函數f

    return 0;
}

// 定義在後面
int f(int x)
{
  int result = 2*x*x +3*x + 4;
  return result;
```

```
}
```

或許有同學會覺得把它拆成兩段一個放前面、一個放後面，不是多此一舉嗎？

這個設計的考量是，我們在開發大專案時，不會把所有程式碼寫在同一個檔案裡，而是會分散在多個檔案裡。

如果有 10,000 行程式碼放在同一檔案裡，只要有一行修改，這 10,000 行都要重新編譯、連結、產出執行檔。

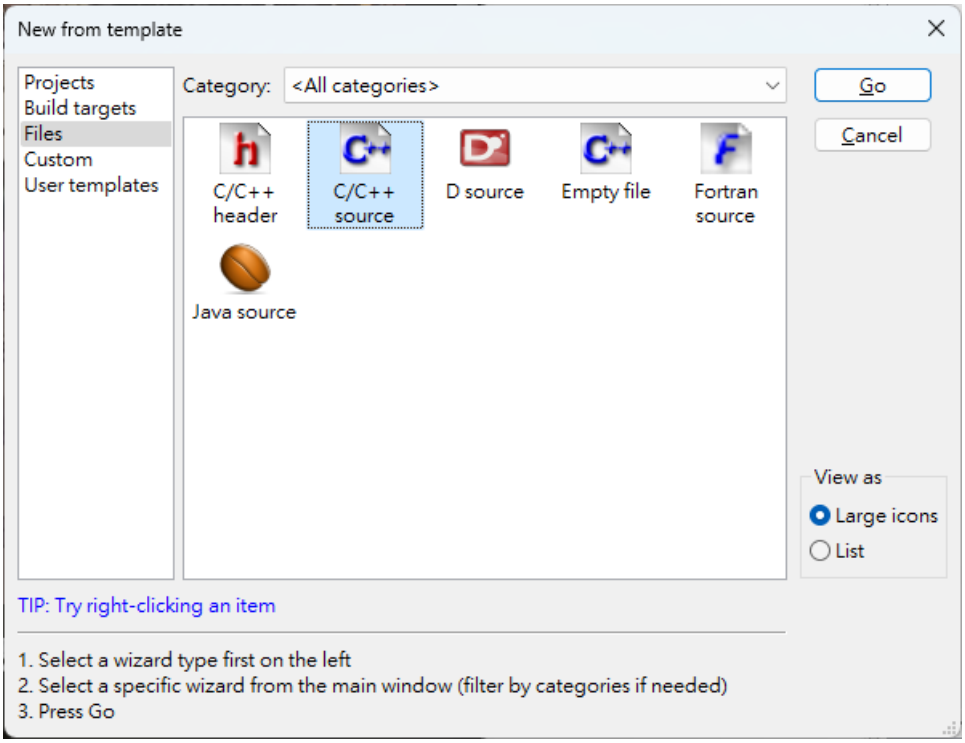
但若是把它拆成 10 個 1,000 行的檔案，當其中一行修改時，只有包含那行檔案 的 1,000 行需要重新編譯，然後把 10 個編譯後的檔案連結產出執行檔即可。

# 多檔案專案

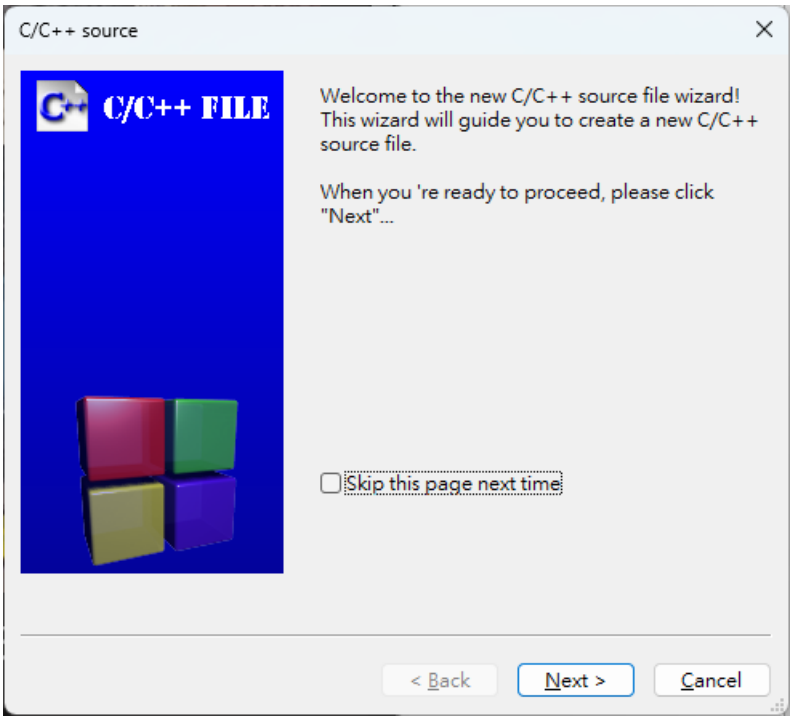
我們來實作一下把範例程式拆成兩個 .cpp 檔案。

目前我們有一個 main.cpp，接下來新增一個 myfuntion.cpp。

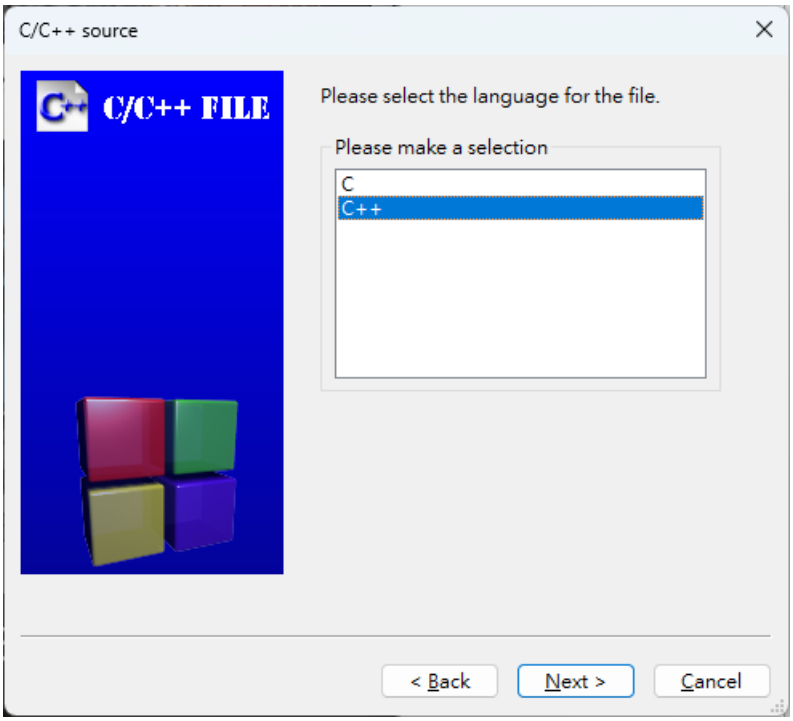
1. 首先依序點選 Code::Blocks 選單 [File]->[New]->[file...]
2. 選擇 [C/C++ source]->[Go]



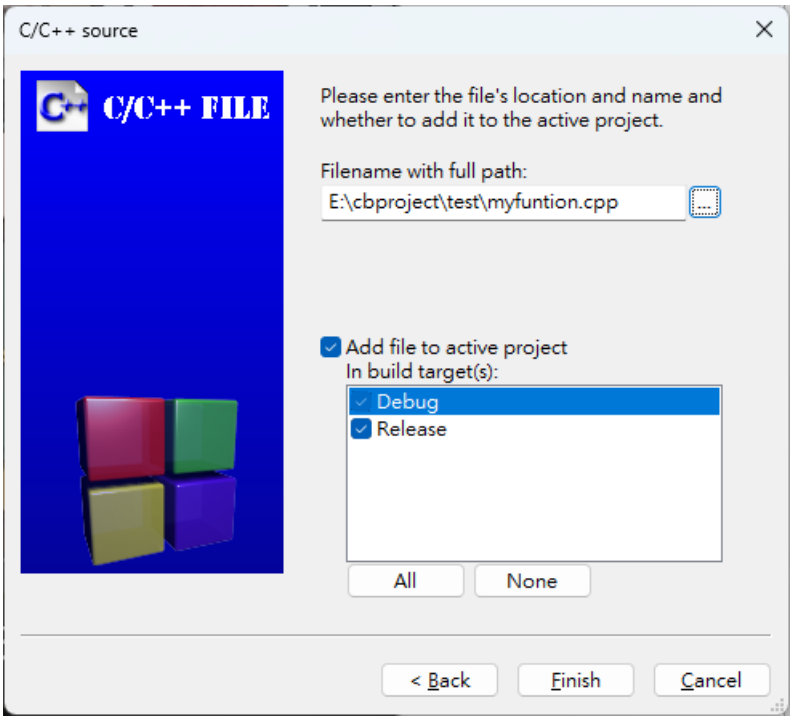

[Next]



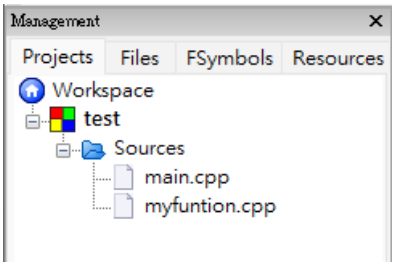

[Next]




點選 [...] 檔名輸入 "myfunction.cpp"，接著點選 [All]->[Finish]




3. 現在專案裡就可以多一個 function.cpp 檔了。




在 [function.cpp] 裡定義好函數f。

```
int f(int x)
{
  int result = 2*x*x +3*x + 4;
  return result;
}
```

在 [main.cpp] 裡宣告函數f 並使用它。

```
#include <iostream>

using namespace std;

int f(int x);  // 宣告在這裡

int main()
{
    int n;
    int ans;

    n = 2;
    ans = f(n);
    cout << ans << endl;   // 18

    ans = f(3);
    cout << ans << endl;   // 31

    n = 5;
    cout << f(n) << endl;  // 69
```

```
    return 0;
}
```

試著建置並執行，應該可以順利完成。

### [練習] 增加一個 $g(x) = x(x-1)$

在 [myfunction.cpp] 裡定義 g(x) 函數

```
int f(int x)
{
  int result = 2*x*x +3*x + 4;
  return result;
}

int g(int x)
{
  return x*(x-1);
}
```

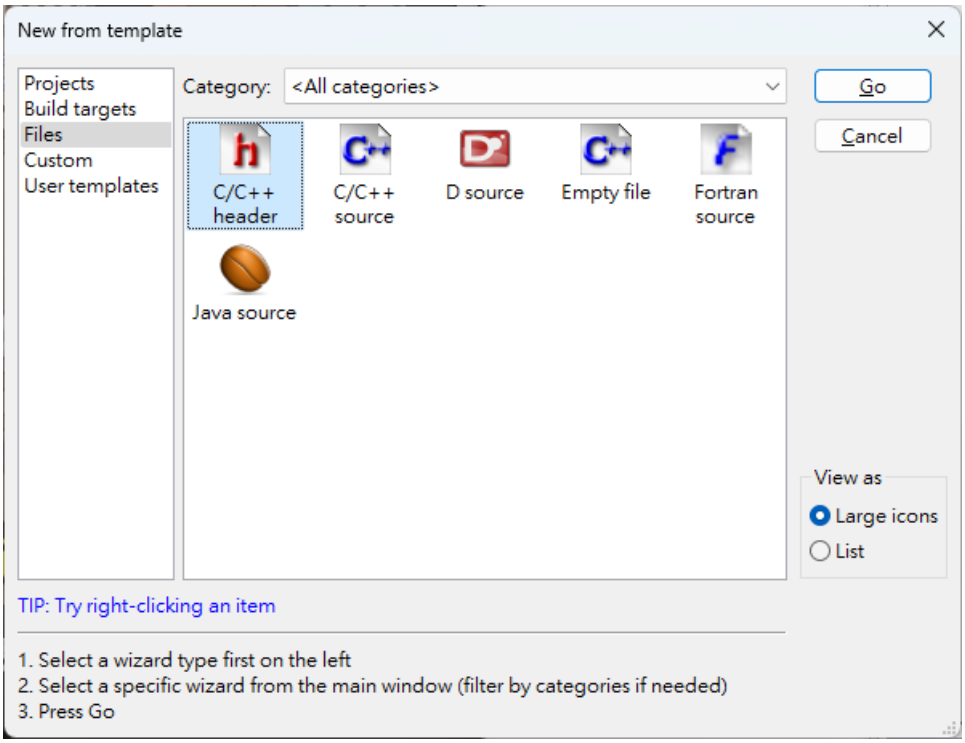
在 [main.cpp] 裡宣告並使用 g(x) 函數

```
#include <iostream>

using namespace std;

int f(int x);
int g(int x); // 宣告 g(x)

int main()
{
    int n;
    int ans;

    n = 2;
    ans = f(n);
    cout << ans << endl;

    cout << g(3) << endl;  // 6, 使用 g(x)

    return 0;
}
```

## 標頭檔(header file)

隨著自己定義的函數愈來愈多，[main.cpp] 前面的宣告會愈來愈多行。我們可以把這些宣告移到另一個檔案裡。

類似之前我們新增 [C/C++ source]檔 的方式，這次我們新增一個 **[C/C++ header]** 檔，並命名為 "myfunction.h"。




把 [main.cpp] 裡的宣告移到 [myfunction.h] 裡。

```
int f(int x);
int g(int x);
```

在 [main.cpp] 裡引入(include)標頭檔 [myfunction.h] 。在編譯時，編譯器會把 myfunction.h 檔案的內容抄到這個引入的地方。

```
#include <iostream>
#include "myfunction.h"  // 引入標頭檔 myfunction.h

using namespace std;

int main()
{
    int n;
    int ans;

    n = 2;
    ans = f(n);
    cout << ans << endl;

    cout << g(3) << endl;

    return 0;
}
```

建置並執行後，程式應該可以順利運行。

我們從一開始學 C++ 就在程式的開頭有一行 `#include <iostream>`。現在你應該可以了解它的作用了，它裡面放的就是和輸入、輸出相關的宣告。

至於為什麼它用角括號 `< >`，我們自己寫的用雙引號 `" "` 呢？

這跟標頭檔所在的位置有關，用角括號 `< >` 編譯器會去內建函式庫的資料夾找標頭檔，用雙引號 `" "` 編譯器會去目前這個專案的資料夾去找標頭檔。

---

Revision #21
★Created 5 May 2024 01:34:21 by huihui
Updated 1 August 2024 12:53:28 by huihui