# Chapter 2

# C++程式架構

# 簡單 C++ 程式範例

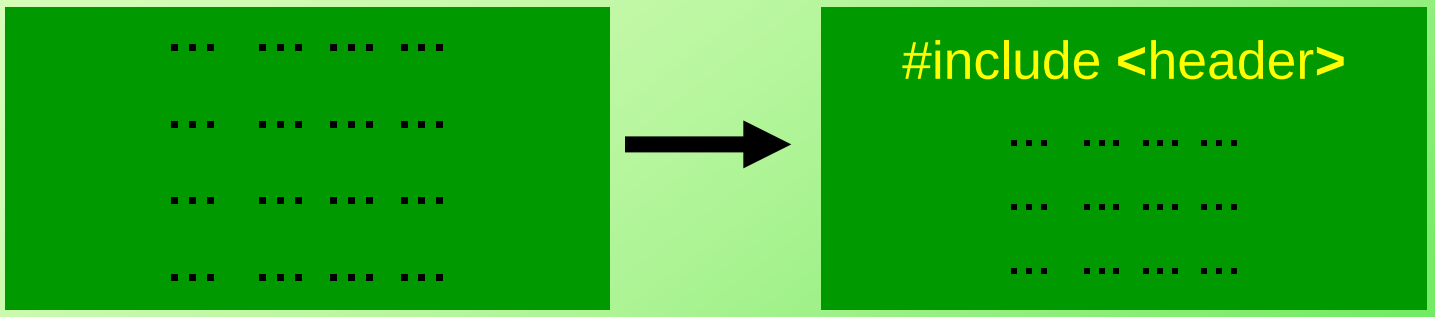


```
#include <iostream>

using namespace std ;

int main(){
    int c , f ;          // 定義兩個整數變數
    cin >> c ;           // 輸入攝氏溫度後存到整數 c
    f =  9 * c / 5 + 32 ;
    // 主函式輸出
    cout << "華氏溫度 = " << f << '\n' ;
    return 0 ;
}
```

# 標頭檔



- 使用 **#include** 將另一程式片段加入程式內




header

```
#include <header.h>
#include <header>
#include ”header”
```

- **#include** 後不加分號

header file

# 名稱空間



- 每一筆資料名稱都有其所屬的名稱空間

  **data → foo::data**

  **data** 為在 **foo** 名稱空間 內的名稱

- 於不同名稱空間的相同名稱仍視為不同

  **foo::data**　　**bar::data**

- 使用 **using namespace + 名稱空間** 可節省名稱空間的一再撰寫

```
using namespace std ;
std::cin  ...  →  cin  ...
std::cout ...  →  cout ...
```

namespace

# 主函式



- C++ 程式的執行由主函式開始

```
int main( void ) {
    double a = 3.14 ;
    ... ...
    ... ...
    return 0 ;
}
```

| | |
|---|---|
| int | 函式回傳型別 |
| main | 主函式名稱 |
| void | 主函式的輸入參數 |
| return | 主函數的回傳數值 |

main function

# 變數宣告與定義 (一)



- 變數宣告 (declaration)
  告知編譯器某變數資料的存在

- 變數定義 (definition)
  要求編譯器分配適當大小的記憶空間給變數使用

```
int dollar ;          // dollar 為整數型別

char ch1 , ch2 ;      // ch1 、 ch2 皆為字元型別

double d1 , d2 ;      // d1 、 d2
                      // 皆為雙精確度浮點數型別
```

declaration , definition

# 變數宣告與定義（二）



- 變數型別不同，其儲存空間大小與儲存方式也有差異

```
char   : 1 位元組 }
int    : 4 位元組 }   整數類型

float  : 4 位元組 }
double : 8 位元組 }   浮點數類型
```

- 變數在定義時，可直接使用 = 或是括號設定初始值

```
int    foo = 3 ;
float  bar(2.2) ;
float  c(bar) ;
int    a = 5 , b(4) ;
```

❖ 不能使用 `int s = t = 3 ;` 錯誤！ 因 `t` 沒有定義

# 變數宣告與定義（三）



- 若 b.cc 檔內須使用到 a.cc 檔內所定義的變數時，則 b.cc 檔僅能以宣告的方式使用此變數

```
… … … …
int foo;
… … … …
… … … …
… … … …
```

a.cc

```
#include <header>
… … … …
extern int foo
… … … …
… … … …
```

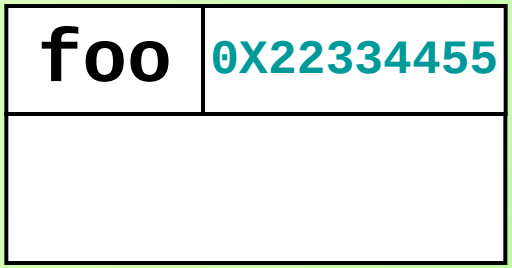
b.cc

- 此時 b.cc 檔的 extern int foo 代表 foo 整數變數已在其他檔案內定義

# 變數的指定

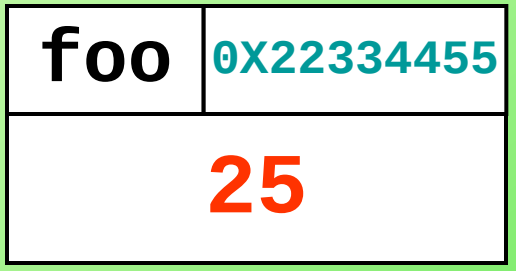


```
int foo ;
```




4 個位元組

```
foo = 10 + 3 * 5 ;
```




- 「＝」運算子將右邊數值存入左邊 **foo** 變數所在位址
- 「＝」稱為指定運算子，不是「等於」
- 指定運算子的左邊一定有一個空間 (location) 用來儲存運算子右邊的資料 (read value)
  - **lvalue**(左值)：代表指定運算子左邊變數位址
  - **rvalue**(右值)：代表指定運算子右邊的資料值

assignment

# 敘述

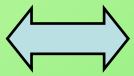


- 程式碼是以敘述為一單位，敘述間以分號隔開

```
int i , j = 5 ;
j = i + j ;
```

- 敘述間的換行字元('\n')，對齊字元('\t')與空白字元(' ')視為相同

```
i = i +
    j ;
```



```
i = i + j ;
```

- 複合敘述 (compound statement)

將若干的敘述用大括號框住，可以視為單一敘述使用

```
{ int i , j = 5 ;
  j = i + j ;
}
```

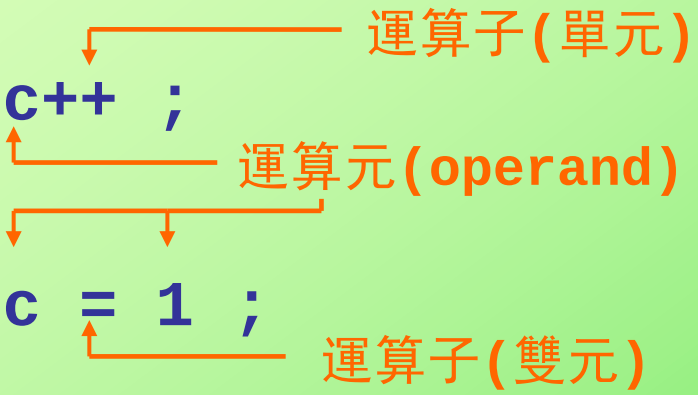
statement

# 基本數值運算 (一)



- 單獨運算式：

  包含一個運算子 (operator) 與 1 到 2 個運算元 (operand)




- 複合運算式：

  幾個運算式依照運算子處理的優先順序依序執行

```
c = a + b ;
    ②   ①
a = b * c + d / e ;
    ④   ①   ③   ②
```

operator

# 基本數值運算（二）



- C++ 的運算子種類繁多，各有其不同的優先順序，為避免認知錯誤，儘可能使用小括號來確定各運算子的處理順序

```
a = b – c + d * e ;
  ④   ②   ③   ①

a = b – ( c + d ) * e ;
  ④   ③     ①     ②
```

# 基本數值運算（三）



- 同型別運算元之運算式
  運算結果與兩運算元同型別

- 不同型別運算元之運算式
  運算結果與兩運算元中型別較大的運算元同型別

```
3 / 2 = 1      → 整數與整數之間運算為整數
3 / 2. = 1.5   → 整數與浮點數之間的運算為浮點數
```

❖ 運算過程中的暫時數值也可能產生溢位

```
100000 * 100000 / 100000 ≠ 100000 (整數溢位)
```

但是

```
1. *  100000 * 100000 / 100000 = 100000
```

# 註解



- 單行註解 ： 由 // 到末尾為註解範圍

```
int r ;                          //r 為半徑
double PI = 3.14 ;               //PI 為圓周率
```

- 多行註解：在 /* */ 之間的範圍為註解範圍

```
/* 圓面積 = π x r²
   圓周長 = 2πr      */
double area = PI * r * r ;
double perimeter = 2 * PI * r ;
```

comment

# 輸出與輸入 (一)



- 輸入：使用輸入運算子 >> 與預設的 cin 變數

```
int    foo ;          // 定義一整數 foo
cin >> foo ;          // 將輸入的資料存到 foo 所在的
                      // 整數空間內

int a , b ;           // 定義兩整數 foo ， bar
cin >> a >> b ;       // 將資料依序存到 foo 與 bar
                      // 所在的整數空間
```

- 輸出：使用輸出運算子 << 與預設的 cout 變數

```
int foo  =  3 ;
cout << foo ;         // 將 foo 整數的值送到螢幕

int bar = 5 ;
cout << foo << ' ' << bar << endl ;
```

input/output

# 輸出與輸入 (二)



- 輸出與輸入的式子可跨越多行撰寫

```
cin >> a >> b ;      ⟺    cin >> a
                           >> b ;
```

```
        cout << foo << ' ' << bar << endl ;
⟺      cout << foo << ' '
             << bar << endl ;
```

- **cin** 與 **cout** 都定義在 **iostream** 標頭檔內，且隸屬於 **std** 名稱空間。若程式內不使用敘述 **using namespace std**，則 **cin** 與 **cout** 要寫成 **std::cin** 與 **std::cout**

# 巨集指令

■ 巨集指令：

由 # 號開始的程式碼，用來控制正式進入編譯之前的前處理單元之執行步驟

常用的巨集指令

**#include** ：加入某程式碼

**#define** ：定義巨集變數

**#ifdef 與 #ifndef** ：設定前處理單元的執行區塊

❖ 巨集指令不是 C++ 語法的一部份，使用時末尾不須加上分號

macro

# #include

- #include <header1>
  header1 檔案在系統所預設的目錄內

- #include "header2"
  header2 檔案與程式碼同在一個目錄下

```
...
int a ;
...
```

f2

```
#include  <f1>
#include  "f2"
```

foo.cc

foo.cc 與 f2 在相同目錄
f1 在系統所預設的目錄內

# #define (一)



- 巨集變數與巨集函式

```
#define PI  3.14                  // 定義巨集變數 PI 為 3.14
#define negative(x) -(x)          // 定義巨集函式 negative(x) 為 -(x)
using namespace std ;
int main(){
  int a = 3;
  cout << PI << '\n';                  //  cout << 3.14 << '\n';
  cout << negative(a) << endl;  //  cout << -(a) << endl;
  return 0;
}
```

# #define（二）



- 奇怪的結果

```
#define  one        3-2
#define  sqr(x)     x*x
#define  cubic(x) (x)*(x)*(x)
```

```
one*one                    =  3 - 2 * 3 - 2              = -5
sqr(1)*sqr(1)              =  1 * 1 * 1 * 1              = 1
sqr(1+2)*sqr(1+2)          =  1+2 * 1+2 * 1+2 * 1+2      = 9
cubic(1+2)*cubic(1+2)      = (1+2)*(1+2)*(1+2) *
                             (1+2)*(1+2)*(1+2)           = 3^6
```

# #ifdef 與 #ifndef (一)



- 簡單條件巨集指令

```
#ifdef FOO          // 如果巨集變數 FOO 已被定義
  part A            // 則編譯 part A 的程式碼
#endif


#ifndef FOO         // 如果巨集變數 FOO 尚未定義
  part B            // 則編譯 part B 的程式碼
#endif
```

# #ifdef 與 #ifndef (二)



```
#include <iostream>
using namespace std ;
int main(){
    int i = 5;
#ifndef MY_PC
#define MY_PC
    i = 7;
#endif
    cout << i << '\n' ;
#ifdef MY_PC
    i = 3;
#endif
    cout << i << endl ;
    return 0;
}
```

左邊各有兩個巨集條件指令分別為

**#ifndef 與 #ifdef**

由於巨集變數變數 **MY_PC** 在 **#ifndef** 條件式之前尚未定義 **#ifdef** 條件式之前已被定義

編譯器所編譯到的主程式碼為：

```
int main(){
    int i = 5;
    i = 7;
    cout << i << '\n' ;
    i = 3;
    cout << i << endl ;
    return 0;
}
```

# #ifdef 與 #ifndef (三)



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

int main(){
    int i = 5;
#ifndef MY_PC
#define MY_PC
    i = 7;
#endif
    cout << i << '\n' ;
#ifndef MY_PC
    i = 3;
#endif
    cout << i << endl ;
    return 0;
}
```

將之前的 ifdef 改成 ifndef

編譯器所編譯到的主程式碼：

```
int main(){
    int i = 5;
    i = 7;
    cout<< i << '\n';
    Cout<< i << endl;
    return 0;
}
```

# #ifdef 與 #ifndef (四)



- 兩種條件巨集指令

```
#ifdef  FOO
    Part A
#else
    Part B
#endif
```

```
#ifndef FOO
    Part C
#else
    Part D
#endif
```

# #ifdef 與 #ifndef (五)



- 使用條件巨集指令

```
#ifndef FOO
#define FOO
    敘述 1
#endif
    敘述 2
#ifndef FOO
#define FOO
    敘述 3
#endif
```

```
#define FOO
#ifndef FOO
    敘述 1
#else
    敘述 2
#endif
#ifdef FOO
    敘述 3
#endif
```

```
#define FOO
#ifdef  FOO
    敘述 1
#else
    敘述 2
#endif
#ifndef FOO
    敘述 3
#endif
```