C++程式設計簡介 2

C++程式設計藝術(第七版)(國際版)

學習目標

在本章中，你將學到：

- 用 C++撰寫簡單的電腦程式。
- 撰寫簡單的輸入與輸出敘述。
- 使用基本型別。
- 基本電腦記憶體觀念。
- 使用算術運算子。
- 算術運算子的優先順序。
- 撰寫簡單的判斷敘述。

本章綱要

2.1 簡介

- 現在介紹C++程式設計，可幫助我們以有條不紊的方式設計程式。
- 本書大部分的C++程式都會處理資訊，並顯示處理結果。

2.2 第一個 C++ 程式：列印一行文字

- 可印出一行文字的簡單程式 (圖2.1)。

```
1  // Fig. 2.1: fig02_01.cpp
2  // Text-printing program.
3  #include <iostream> // allows program to output data to the screen
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome to C++!\n"; // display message
9
10    return 0; // indicate that program ended successfully
11 } // end function main
```

```
Welcome to C++!
```

圖 2.1　列印文字的程式

2.2 第一個 C++ 程式：列印一行文字

```
1  // Fig. 2.1: fig02_01.cpp
2  // Text-printing program.
```

- 以//開始的每一行表示//右方的部分是**註解(comment)**。
  - 你可以插入註解，說明你的程式，並幫助其他人閱讀、了解該程式。
  - C++編譯器會忽略註解，不會產生任何機器語言目的碼。
- 以//開頭的註解稱為**單行註解 (single-line comment)**，因為註解只有這一行。
- 你也可以使用C的風格撰寫多行註解。它以/*開頭並以*/結尾，中間包住的部分就是註解。

```
/*
  hhhh
  gggg
*/
```

良好的程式設計習慣 2.1

每個程式開頭都要寫個註解，說明程式目的。

```
1   // Fig. 2.1: fig02_01.cpp
2   // Text-printing program.
3   #include <iostream> // allows program to c
4
5   // function main begins program execution
6   int main()
7   {
8      std::cout << "Welcome to C++!\n"; // di
9
10     return 0; // indicate that program ende
11  } // end function main
```

2.2 第一個 C++ 程式：列印一行文字

```
3  #include <iostream>
```

- **前置處理指令 (preprocessor directive)**，是給C++前置處理器的訊息。
- 在程式編譯之前，前置處理器會處理以#開頭的每一行。
- `#include <iostream>`程式碼會通知前置處理器，將**輸入/輸出串流標頭檔<iostream>**的內容含入程式以供使用。
  - 任何程式想要使用C++的串流，以將資料輸出到螢幕或從鍵盤輸入資料，就必須載入此標頭檔。

常見的程式設計錯誤 2.1

若從鍵盤輸入資料 (或輸出資料至螢幕) 的程式忘記載入 <iostream> 標頭檔，編譯器會產生錯誤訊息，因爲編譯器認不得串流元件 (如 cout) 的參照。

```
1   // Fig. 2.1: fig02_01.cpp
2   // Text-printing program.
3   #include <iostream> // allows program to c
4
5   // function main begins program execution
6   int main()
7   {
8      std::cout << "Welcome to C++!\n"; // di
9
10     return 0; // indicate that program ende
11  } // end function main
```

2.2 第一個 C++ 程式：列印一行文字

- 程式設計者可用空白行、空白字元與tab字元 (也就是「tab」) 來增加程式可讀性。
  - 這些字元合稱**空格 (white space)**。
  - 編譯器通常會忽略空格字元。

良好的程式設計習慣 2.2

空白行和空白字元可增加程式的可讀性。

```
1   // Fig. 2.1: fig02_01.cpp
2   // Text-printing program.
3   #include <iostream> // allows program to c
4
5   // function main begins program execution
6   int main()
7   {
8      std::cout << "Welcome to C++!\n"; // di
9
10     return 0; // indicate that program ende
11  } // end function main
```

2.2 第一個 C++ 程式：列印一行文字

```
6  int main()
```

- main是每個C++程式都有的部分。
- 在main之後的小括號表示main是程式的一個建構區塊，稱為**函式 (function)**。
- C++程式通常由一或多個函式與類別組成。
- 每個程式只能有一個稱為main的函式。
- C++程式一開始會先執行main函式，就算main函式不是程式的第一個函式也一樣。
- 位於main左邊的關鍵字int，表示main函式會「傳回」一個整數值。
  - **關鍵字 (keyword)** 是C++在程式中的保留字，以供特殊用途。
  - 目前我們只要在每個程式的main左邊擺個int關鍵字就行了。

函式定義長這樣

```
傳回值型別 函式名稱( …)
{
    …

}
```

```
 6   int main()
 7   {
 8       std::cout << "Welcome to C++!\n"; //
 9
10       return 0; // indicate that program e
11   } // end function main
```

2.2 第一個 C++ 程式：列印一行文字

```
6  int main()
7  {
```

- 每個函式主體的起始處必須是左大括弧{。
- 而對應的右大括號 } (right brace) 表示每個函式主體的結束之處。

```
1  // Fig. 2.1: fig02_01.cpp
2  // Text-printing program.
3  #include <iostream> // allows program to c
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome to C++!\n"; // di
9
10    return 0; // indicate that program ende
11 } // end function main
```

2.2 第一個 C++ 程式：列印一行文字

```
8       std::cout << "Welcome to C++!\n";
```

- 敘述句會指示電腦**執行某個動作 (perform an action)**。
- 字串有時稱做**字元字串 (character string)**、或是**字面字串 (string literal)**。
- 我們將雙引號之間的字元簡稱為**字串 (string)**。
  - 編譯器不會忽略字串內的空白字元。
- 每行C++敘述須以分號; (也叫**敘述結束符號，statement terminator**) 做結束。
  - 前置處理指令 (如#include) 則不以分號做結。

```
1  // Fig. 2.1: fig02_01.cpp
2  // Text-printing program.
3  #include <iostream> // allows program to o
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome to C++!\n"; // di
9
10    return 0; // indicate that program ende
11 } // end function main
```

2.2 第一個 C++ 程式：列印一行文字

```
8       std::cout << "Welcome to C++!\n";
```

- C++利用字元串流來輸入及輸出
- 執行cout敘述後，程式就會將字元串流送到**標準輸出串流物件std::cout（standard output stream object）**輸出，它通常「連結」到螢幕。
- 用到前置處理指令#include <iostream>所帶入的名稱時，必須在cout前加上std::。
  - 而std::cout符號代表我們所用的cout是屬於std這個「命名空間」。
  - cin (標準輸入串流) 與cerr (標準錯誤串流) 亦屬於命名空間std。
- 運算子<<稱為**串流插入運算子 (stream insertion operator)**。
  - 運算子右方的值，也就是右**運算元 (operand)** 會插入輸出串流中。

2.2 第一個 C++ 程式：列印一行文字

```
8       std::cout << "Welcome to C++!\n";
```

- 字元\n不會顯示在螢幕上。
- 反斜線符號 (\) 稱為**跳脫字元 (escape character)**。
  - 它表示要輸出一個「特殊」字元。
- 當字串中出現反斜線時，下一個字元就會與反斜線結合成跳脫序列 (escape sequence)。
- 跳脫序列\n表示新增一行 (newline)。
  - 會讓游標移到螢幕的下一行開端。

跳脫序列	說明
\n	換行。將游標移到下一行起始處。
\t	水平 tab。將游標移到下一個 tab 定位點。
\r	移回起始處。將游標移到目前這一行的起始位置；不要移到下一行。
\a	警告。讓系統發出警告聲。
\\	反斜線符號可用來列印反斜線字元。
\'	單引號。可用來列印單引號字元。
\"	雙引號。可用來列印雙引號字元。

圖 2.2　　跳脫序列

```
1  // Fig. 2.1: fig02_01.cpp
2  // Text-printing program.
3  #include <iostream> // allows program to c
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome to C++!\n"; // di
9
10    return 0; // indicate that program ende
11 } // end function main
```

2.2 第一個 C++ 程式：列印一行文字

```
return 0; //
```

- 在main函式的結束處使用return敘述（return statement）時，若值為0，表示該程式成功結束。
- 依據C++標準，假如程式到達main的最後而沒有碰到return敘述句，就表示程式成功結束了，跟在main的最後放上return 0;是相同的。

常見的程式設計錯誤 2.2

省略 C++敘述結尾的分號是一種語法錯誤。(再說一遍，前置處理指令不須以分號做結)。程式語言的**語法 (syntax)** 說明了以該語言建立正確程式的規則。當編譯器碰到違反 C++語言規則 (也就是語法) 的程式碼時，便發生**語法錯誤 (syntax error)**。編譯器通常會發出錯誤訊息，幫助你找出並修改錯誤的程式碼。語法錯誤又稱爲**編譯器錯誤 (compiler error)**、**編譯時期錯誤 (compile-time error)** 或是**編譯錯誤 (compilation error)**，因爲這些錯誤是編譯器在編譯階段偵測出來的。修正完所有語法錯誤之前，程式是無法執行的。某些編譯器錯誤並非語法錯誤，後面將會看到。

良好的程式設計習慣 2.3

將函式的整個主體內容在劃分函式主體的大括號間縮排一層，可讓程式的功能結構更加凸顯，提升程式可讀性。

```
 6  int main()
 7  {
 8     std::cout << "Welcome to C++!\n"; //
 9
10     return 0; // indicate that program e
11  } // end function main
```

良好的程式設計習慣 2.4

依您的喜好設定慣用的縮排量，然後統一使用此縮排量。Tab 鍵可用來建立縮排，但 tab 定位點可能有所差異。我們建議每隔 1/4 英吋放個 tab 定位點，或最好以三個空白作爲一層的縮排量。

2.3 修改第一個 C++ 程式

- 我們可用多種方式印出Welcome to C++!。

```
1  // Fig. 2.3: fig02_03.cpp
2  // Printing a line of text with multiple statements.
3  #include <iostream> // allows program to output data to the screen
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome ";
9     std::cout << "to C++!\n";
10 } // end function main
```

```
Welcome to C++!
```

圖 2.3　使用多行敘述印出一行文字

2.3 修改第一個 C++ 程式

- 單一敘述也可使用換行字元印出多行。
- 每次在輸出串流中出現\n (換行) 跳脫序列時，螢幕游標就會移到下一行的起始位置。
- 若想輸出一行空白，只要連續使用兩個換行字元即可。

```
6  int main()
7  {
8     std::cout << "Welcome\nto\n\nC++!\n";
9  } // end function main
```

```
1  // Fig. 2.4: fig02_04.cpp
2  // Printing multiple lines of text with a single statement.
3  #include <iostream> // allows program to output data to the screen
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome\nto\n\nC++!\n";
9  } // end function main
```

```
Welcome
to

C++!
```

圖 2.4　使用一行敘述列印多行文字

2.4 另一個 C++ 程式：整數加法

- 使用輸入串流物件**std::cin**，以及**串流擷取運算子>>（stream extraction operator）**，從鍵盤取得數值。

Input stream object

- `std::cin` from `<iostream>`
  標準輸入串流物件
  - Usually connected to keyboard
  - Stream extraction operator `>>` 串流擷取運算子
    - Waits for user to input value, press *Enter* (Return) key
    - Stores value in variable to right of operator
    - Converts value to variable data type
  - Example

    ```
    std::cin >> number1;
    ```

    Reads an integer typed at the keyboard

    Stores the integer in variable `number1`

```
1  // Fig. 2.5: fig02_05.cpp
2  // Addition program that displays the sum of two integers.
3  #include <iostream> // allows program to perform input and output
4
5  // function main begins program execution
6  int main()
7  {
8     // variable declarations
9     int number1; // first integer to add
10    int number2; // second integer to add
11    int sum; // sum of number1 and number2
12
13    std::cout << "Enter first integer: "; // prompt user for data
14    std::cin >> number1; // read first integer from user into number1
15
16    std::cout << "Enter second integer: "; // prompt user for data
17    std::cin >> number2; // read second integer from user into number2
18
19    sum = number1 + number2; // add the numbers; store result in sum
20
21    std::cout << "Sum is " << sum << std::endl; // display sum; end line
22 } // end function main
```

圖 2.5　加法程式，顯示鍵盤輸入的兩個數的和

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

2.4 另一個 C++ 程式：整數加法

- 宣告（ declarations）是用來讓程式認識識別字。
- 所有變數都需宣告名稱與資料型別，程式才能使用它。
- 識別字number1、number2和sum是**變數 (variable)** 名稱。
- 變數就是電腦記憶體中的某個位置，它可以存放數值供程式使用。
- 變數number1、number2與sum的資料型別是int，表示這些變數是存放**整數**數值。

```
3  #include <iostream> // allows program to perfo
4
5  // function main begins program execution
6  int main()
7  {
8     // variable declarations
9     int number1; // first integer to add
10    int number2; // second integer to add
11    int sum; // sum of number1 and number2
12
13    std::cout << "Enter first integer: "; // pr
14    std::cin >> number1; // read first integer
15
16    std::cout << "Enter second integer: "; // p
17    std::cin >> number2; // read second integer
18
19    sum = number1 + number2; // add the numbers
20
21    std::cout << "Sum is " << sum << std::endl;
22 } // end function main
```

2.4 另一個 C++ 程式：整數加法

- 若在一行中宣告多個變數名稱 (如前述)，這些名稱可用逗號 (,) 分隔，這稱為**以逗號分隔的列表 (comma-separated list)**。

```
int a, b, c;
```

良好的程式設計習慣 2.5

在每個逗號 (,) 後面放一個空白字元，提高程式可讀性。

2.4 另一個 C++ 程式：整數加法

- double：實數資料型別
  - 實數就是帶小數點的數，如3.4、0.0和－11.19。
- char：字元資料型別。
  - 一個char變數只能儲存一個小寫字母、大寫字母、阿拉伯數字或特殊字元 (如$或*)。
- int、double與char等型別通常稱作**基礎型別 (fundamental type)**。
- 基礎型別的名稱都是保留字，因此都須以小寫字母表示。
- 附錄C含完整的基礎型別清單。

2.4 另一個 C++ 程式：整數加法

- 變數名稱就是任何非關鍵字的有效**識別字 (identifier)**。
- 識別字是由字母、數字和底線 (_) 組成的一連串字元，但第一個字元不可是數字。
- C++會**區分大小寫（case sensitive）**，所以a1與A1是不同的識別字。
- 識別字可以用來做為變數名稱、函式名稱或類別名稱

以下有多少合法的識別字?

- a 01
- b
- 2c
- D3
- _E
- f-3
- hhh
- 3q

可攜性的小技巧 2.1

C++的識別字長度不限，但您的 C++實作可能會對識別字的長度加上某些限制。請使用 31 個字元以下的識別字，以確保可攜性。

良好的程式設計習慣 2.6

請選擇有意義的識別字，以輔助程式「**自我文件化**」**(self-documenting)** —別人只要直接讀程式碼就懂了，不用再翻手冊或讀註解。

良好的程式設計習慣 2.7

識別字不要用縮寫，以提升程式可讀性。

良好的程式設計習慣 2.8

不要用開頭是底線和雙底線的識別字，因爲 C++ 編譯器可能會用這種識別字完成內部工作。這可以避免您的識別字與編譯器所用的識別字混淆。

測試和除錯的小技巧 2.1

C++這種語言是瞬息萬變的。每演進一次，關鍵字可能就變得更多。建議您別用像是「object」等有特殊涵義的字當作識別字。雖然「object」現在並不是 C++的關鍵字，但未來它可能會變成關鍵字，因此新的編譯器也許無法編譯現有的程式碼。

2.4 另一個 C++ 程式：整數加法

- 變數宣告可放在程式的任何位置，但宣告必須出現在此變數使用的地方之前。

```
3  #include <iostream> // allows program to perfo
4
5  // function main begins program execution
6  int main()
7  {
8     // variable declarations
9     int number1; // first integer to add
10    int number2; // second integer to add
11    int sum; // sum of number1 and number2
12
13    std::cout << "Enter first integer: "; // pr
14    std::cin >> number1; // read first integer
15
16    std::cout << "Enter second integer: "; // p
17    std::cin >> number2; // read second integer
18
19    sum = number1 + number2; // add the numbers
20
21    std::cout << "Sum is " << sum << std::endl;
22 } // end function main
```

良好的程式設計習慣 2.9

宣告和接下來的可執行敘述之間，一定要加一行空白。可讓該宣告在程式中更爲明顯，使程式更清楚。

```
3  #include <iostream> // allows program to perfo
4
5  // function main begins program execution
6  int main()
7  {
8     // variable declarations
9     int number1; // first integer to add
10    int number2; // second integer to add
11    int sum; // sum of number1 and number2
12
13    std::cout << "Enter first integer: "; // pr
14    std::cin >> number1; // read first integer
15
16    std::cout << "Enter second integer: "; // p
17    std::cin >> number2; // read second integer
18
19    sum = number1 + number2; // add the numbers
20
21    std::cout << "Sum is " << sum << std::endl;
22 } // end function main
```

2.4 另一個 C++ 程式：整數加法

- **提示 (prompt)**，指示使用者進行某特定動作。
- 使用**cin 輸入串流物件** cin 以及**串流擷取運算子>>**從標準輸入串流(鍵盤)取得數值。

```
3  #include <iostream> // allows program to perfo
4
5  // function main begins program execution
6  int main()
7  {
8     // variable declarations
9     int number1; // first integer to add
10    int number2; // second integer to add
11    int sum; // sum of number1 and number2
12
13    std::cout << "Enter first integer: "; // pr
14    std::cin >> number1; // read first integer
15
16    std::cout << "Enter second integer: "; // p
17    std::cin >> number2; // read second integer
18
19    sum = number1 + number2; // add the numbers
20
21    std::cout << "Sum is " << sum << std::endl;
22 } // end function main
```

測試和除錯的小技巧 2.2

程式應確認所有輸入值的正確性，避免錯誤資訊影響程式計算結果。

2.4 另一個 C++ 程式：整數加法

- 當電腦執行輸入敘述時，它會等待使用者輸入變數**number1**的值。
- 使用者鍵入一個整數 (以字元形式) 並按**Enter**鍵將字元傳給電腦。
- 電腦會將輸入字元轉換成整數，並將此數字 (或值，value) 指定 (複製) 給變數**number1**。

```
3  #include <iostream> // allows program to perfo
4
5  // function main begins program execution
6  int main()
7  {
8     // variable declarations
9     int number1; // first integer to add
10    int number2; // second integer to add
11    int sum; // sum of number1 and number2
12
13    std::cout << "Enter first integer: "; // pr
14    std::cin >> number1; // read first integer
15
16    std::cout << "Enter second integer: "; // p
17    std::cin >> number2; // read second integer
18
19    sum = number1 + number2; // add the numbers
20
21    std::cout << "Sum is " << sum << std::endl;
22 } // end function main
```

2.4 另一個 C++ 程式：整數加法

- 此後，程式中任何參照到**number1**的位置都會使用此相同的數值。
- 在本程式中，指定敘述會計算變數**number1**和**number2**的總和，然後將結果以指定運算子(assignment operator)「**=**」設定給變數**sum**。
  - 大部分的計算都是用指定敘述來執行。

```
3  #include <iostream> // allows program to perfo
4
5  // function main begins program execution
6  int main()
7  {
8     // variable declarations
9     int number1; // first integer to add
10    int number2; // second integer to add
11    int sum; // sum of number1 and number2
12
13    std::cout << "Enter first integer: "; // pr
14    std::cin >> number1; // read first integer
15
16    std::cout << "Enter second integer: "; // p
17    std::cin >> number2; // read second integer
18
19    sum = number1 + number2; // add the numbers
20
21    std::cout << "Sum is " << sum << std::endl;
22 } // end function main
```

2.4 另一個 C++ 程式：整數加法

- 運算子=和+稱為二元運算子 (binary operator)，因為它們具有二個運算元。

```
number1= 3 + 8;
```

良好的程式設計習慣 2.10

在二元運算子的兩邊都放空白字元。除了突顯運算子外，也可讓程式更具可讀性。

2.4 另一個 C++ 程式：整數加法

- **std::endl**是一種**串流操作子 (stream manipulator)**。
- **endl**是「end line」的簡寫，屬於命名空間**std**。
- 串流操作子**std::endl**會輸出一個換行字元，然後「將輸出緩衝區清除」。
  - 這表示在某些系統中，輸出的資料會累積起來，直到累積到夠多的資料量才在螢幕顯示出來，但是**std::endl**會強迫所有累積的資料立即顯示在螢幕上。

2.4 另一個 C++ 程式：整數加法

- 在單一敘述中使用多個串流插入運算子 (<<)，可視為**串接的 (concatenating)**、**連鎖的 (chaining)** 或是**接續的 (cascading) 串流插入運算（stream insertion operation）**。
- 輸出敘述亦可執行計算。

```
std::cout << a + b;
```

```
3  #include <iostream> // allows program to perfo
4
5  // function main begins program execution
6  int main()
7  {
8     // variable declarations
9     int number1; // first integer to add
10    int number2; // second integer to add
11    int sum; // sum of number1 and number2
12
13    std::cout << "Enter first integer: "; // pr
14    std::cin >> number1; // read first integer
15
16    std::cout << "Enter second integer: "; // p
17    std::cin >> number2; // read second integer
18
19    sum = number1 + number2; // add the numbers
20
21    std::cout << "Sum is " << sum << std::endl;
22 } // end function main
```

- 請撰寫一程式輸入長方體的長、寬、高，程式將輸出此長方體的體積。

2.5 記憶體觀念

- 像**number1**、**number2**和**sum**這些變數名稱，實際上是對應到電腦記憶體的某個**位置 (location)**。
- 每個變數，都有名稱、數值、型別和大小。
- 當數值放入記憶體位置時，會蓋掉該位置原本的值，因此，將新數值放到記憶體位置是有**破壞性的（destructive）**。
- 從記憶體讀取數值時，其過程**不具破壞性 (nondestructive)**。

圖 2.6　本記憶體位置顯示變數 number1 的名稱和數值

圖 2.7　儲存 number1 和 number2 的值後的記憶體位置

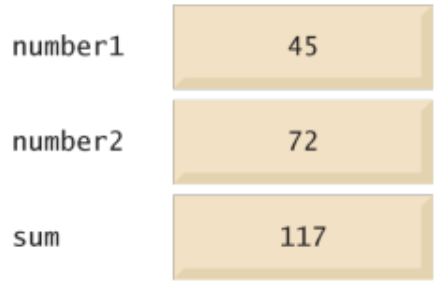


圖 2.8　計算並儲存 number1 和 number2 總和之後的記憶體位置

2.6 算術計算

- 大部分的程式都會執行算術計算。
- 圖2.9整理出C++的**算術運算子 (arithmetic operator)**。
- **星號「*」 (asterisk)** 代表乘法
- **百分比符號 「%」(percent sign)** 則是**模數（modulus）**運算子，我們很快會討論它們。
  - C++提供**模數運算子% (modulus operator)**，它會產生整數除法後的餘數。
  - 模數運算子（%）只能用於整數運算元。
- 圖2.9的算術運算子都是二元運算子。
- **整數除法 (Integer division**，就是被除數與除數都是整數) 會產生整數商數。
  - 整數除法中的分數部分都會**捨去（truncated）**，不會有小數出現。

C++ 運算	C++ 算術運算子	代數運算式	C++ 運算式
加法	+	$f + 7$	f + 7
減法	-	$p - c$	p - c
乘法	*	bm 或 b · m	b * m
除法	/	x / y 或 $\frac{x}{y}$ 或 x ÷ y	x / y
模數運算	%	$r \bmod s$	r % s

圖 2.9　算術運算子

常見的程式設計錯誤 2.3

將模數運算子 (%) 用在非整數運算元會產生編譯錯誤。。

2.6 算術計算

- C++的算術運算式須以**橫行形式（straight-line form）**輸入電腦。
- 因此，像是「a除以b」的運算式就必須寫成a/b，如此所有的常數、變數和運算子都會排成橫行。
- 在C++運算式中，小括號的用法跟代數運算的用法很像。
- 例如，要將a乘以b + c的值，就寫成a * (b + c)。

2.6 算術計算

- C++在算術運算式中的運算子用法，依照下述的**「運算子優先順序規則」(rules of operator precedence)** 決定運算的順序，一般而言與代數中的規則相同：

運算子	運算	計算的順序 (優先順序)
( )	小括號	優先計算。如果小括號是巢狀的，則會先計算最內層的。若有數個此類型的運算，則會從左到右運算。
*, /, %	乘法 除法 模數運算	第二個計算。若有數個此類型的運算，則會從左到右運算。
+ -	加法 減法	最後計算。若有數個此類型的運算，則會從左到右運算。

2.6 算術計算

- C++沒有表示指數的算術運算式，所以把x^2寫成x * x。
- 圖2.11是二次多項式中運算子的計算次序。
- 跟代數運算一樣，在運算式中加入非必要的小括號，可讓運算式的計算順序更清楚。
- 這些括號稱為**多餘括號（redundant parentheses）**。

常見的程式設計錯誤 2.4

某些程式語言使用**或^表示指數運算。C++不支援這些指數運算子，若使用它們會造成錯誤。

良好的程式設計習慣 2.11

在複雜的運算式中使用多餘括號，可讓運算式更清楚。

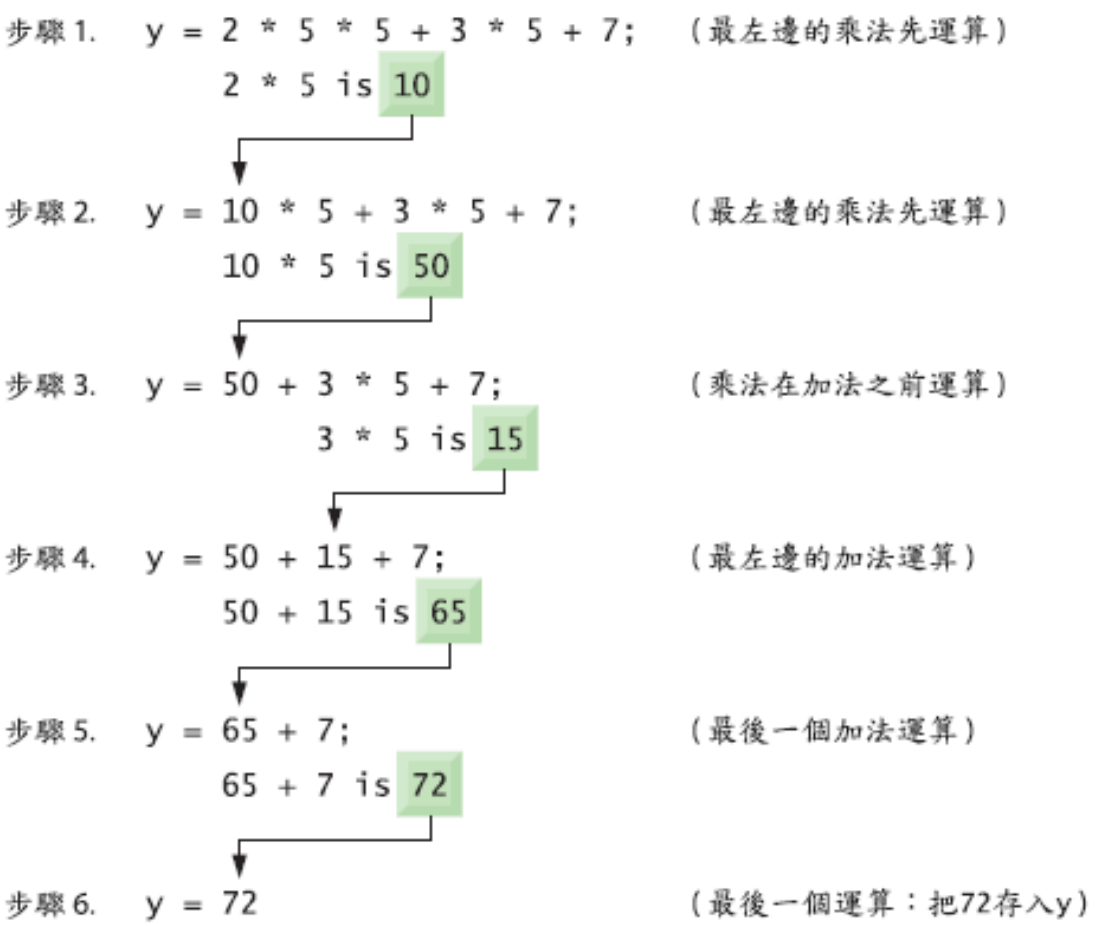


圖 2.11　二次多項式的計算順序

練習

- 請依照運算子優先順序計算以下結果

$$(13+2*5)/4-2*(6-4)$$

2.7 判斷：等號運算子和關係運算子

- **if敘述（if statement）**，它可讓程式按照某些**條件（condition）**的真假值來選擇要執行哪些動作。
- 如果條件為true，則會執行if敘述主體內的敘述。
- 若條件不吻合，也就是條件是偽 (false)，則程式不會執行if結構中的敘述。
- 在if結構中的條件式可利用**等號運算子（equality operators）**和**關係運算子（relational operators）**組成，圖2.12是這些運算子的整理。
- 所有關係運算子都擁有相同的優先權，並從左到右進行結合運算。
- 所有等號運算子都擁有相同的優先權，並從左到右進行結合運算。其優先權低於關係運算子。

if

```
語法:  if(條件式)
       {
          條件正確要執行的敘述
       }

if( x > 0 )
{
   printf("x is a positive number!");
}
```

- Condition 條件式
  - 計算結果是true或false
  - 若為關係運算或是相等運算，則運算結果為true或false
  - 若運算結果是數值，則
    - 0 is false, non-zero is true

常見的程式設計錯誤 2.5

若在運算子 ==、!=、>= 和 <= 的兩個符號中間出現空白，則會產生語法錯誤。

常見的程式設計錯誤 2.6

若將運算子 !=、>= 和 <= 的兩個符號顛倒 (寫成 =!、=> 和 =<)，通常會產生語法錯誤。在某些狀況下，將 != 寫成 =! 並不算是語法錯誤，但幾乎可以肯定是**邏輯錯誤 (logic error)**，在執行期間會造成影響。第 5 章講到邏輯運算子時，您便會了解理由。**致命的邏輯錯誤 (fatal logic error)** 會讓程式失敗且提早結束。**非致命的邏輯錯誤 (nonfatal logic error)** 則可讓程式繼續執行，但可能會產生不正確的結果。

標準代數等號或關係運算子	C++ 等號或關係運算子	C++ 條件範例	C++ 條件的意義
關係運算子			
>	>	x > y	x 大於 y
<	<	x < y	x 小於 y
≥	>=	x >= y	x 大於或等於 y
≤	<=	x <= y	x 小於或等於 y
等號運算子			
=	==	x == y	x 等於 y
≠	!=	x != y	x 不等於 y

圖 2.12　　等號運算子和關係運算子

常見的程式設計錯誤 2.7

將等號運算子 == 和指定運算子 = 搞混淆會產生邏輯錯誤。等號運算子應該唸成「等於」，而指定運算子應該唸成「取得」或「取值自」或「將_值指定給」。有些人喜歡將等號運算子讀成「雙等號」。如 5.9 節所述，將這些運算子搞混不一定會造成顯而易見的語法錯誤，但會造成很微妙的邏輯錯誤。

```
if ( x = 1 )
{
   std::cout << "x is equal to one";
} /* end if */
```

2.7 判斷：等號運算子和關係運算子

- 以下範例使用6個if敘述來比較使用者輸入的兩個數字。
- 若這些if敘述中有任何一個條件滿足，則會執行與該if相關的敘述。
- 圖2.13顯示程式以及三個執行範例的輸入/輸出。

```
1  // Fig. 2.13: fig02_13.cpp
2  // Comparing integers using if statements, relational operators
3  // and equality operators.
4  #include <iostream> // allows program to perform input and output
5
6  using std::cout; // program uses cout
7  using std::cin; // program uses cin
8  using std::endl; // program uses endl
9
10 // function main begins program execution
11 int main()
12 {
13    int number1; // first integer to compare
14    int number2; // second integer to compare
15
16    cout << "Enter two integers to compare: "; // prompt user for data
17    cin >> number1 >> number2; // read two integers from user
18
19    if ( number1 == number2 )
20       cout << number1 << " == " << number2 << endl;
21
```

圖 2.13　利用 if 敘述、關係運算子和等號運算子比較整數的大小

```
22      if ( number1 != number2 )
23         cout << number1 << " != " << number2 << endl;
24
25      if ( number1 < number2 )
26         cout << number1 << " < " << number2 << endl;
27
28      if ( number1 > number2 )
29         cout << number1 << " > " << number2 << endl;
30
31      if ( number1 <= number2 )
32         cout << number1 << " <= " << number2 << endl;
33
34      if ( number1 >= number2 )
35         cout << number1 << " >= " << number2 << endl;
36   } // end function main
```

```
Enter two integers to compare: 3 7
3 != 7
3 < 7
3 <= 7
```

圖 2.13　利用 if 敘述、關係運算子和等號運算子比較整數的大小

```
Enter two integers to compare: 22 12
22 != 12
22 > 12
22 >= 12
```

```
Enter two integers to compare: 7 7
7 == 7
7 <= 7
7 >= 7
```

圖 2.13　利用 `if` 敘述、關係運算子和等號運算子比較整數的大小

2.7 判斷：等號運算子和關係運算子

- **using宣告**，可讓我們不用像之前程式那般重複使用**std::**前置字。
- 一旦使用**using**敘述，之後就能直接寫**cout**取代**std::cout**、**cin**取代**std::cin**，用**endl**取代**std::endl**。
- 許多程式設計師會使用下列的宣告

```
using namespace std;
```

這樣程式就可以使用C++標頭檔（例如<iostream>）中任何可能被引入的名稱 。

- 我們在接下來的程式中，都會使用前述宣告。

```
4  #include <iostream> // allows program
5
6  using std::cout; // program uses cout
7  using std::cin; // program uses cin
8  using std::endl; // program uses endl
```

2.7 判斷：等號運算子和關係運算子

- 圖2.13中的每個if結構都包含一個敘述，因此都內縮一層。

```
if ( number1 == number2 )
   cout << number1 << " == " << number2 << endl;
```

- 第4章會介紹如何在if結構主體內擺進多行敘述 (將主體敘述用一對大括號{ }包起來，這個包起來的部分稱作**複合敘述 (compound statement)** 或**區塊 (block)**)。

良好的程式設計習慣 2.12

將 `if` 結構內的敘述都加以縮排，以增進可讀性。

良好的程式設計習慣 2.13

每一行只撰寫一個敘述，以增進可讀性。

常見的程式設計錯誤 2.8

若在 if 敘述中的條件式右方小括號之後加上一個分號，通常會造成邏輯錯誤 (雖然這不算是語法錯誤)。該分號會讓 if 結構的主體變成空的，所以該 if 結構不會執行任何動作，不管其條件是否爲 true。更糟糕的是，原來 if 結構內的敘述現在變成了循序敘述，就排在這個 if 結構之後，所以一定會執行到，常會讓程式產生不正確的結果。

```
if ( x > 0 );
{
   printf("x is a positive number!");
} /* end if */
```

2.7 判斷：等號運算子和關係運算子

- 你可依個人喜好，將敘述分成多行並用空白隔開。
- 但若將識別字、字串 (如「hello」) 和常數 (如數字 1000) 分成許多行，就是語法錯誤了。

常見的程式設計錯誤 2.9

在識別字的中間加入空白字元 (例如將 `main` 寫成 `ma in`) 是語法錯誤。

良好的程式設計習慣 2.14

長的敘述可以分成多行。若單一敘述必須分成幾行，則所選的斷句點必須有意義，例如在逗號分隔清單中的逗號後面斷句，或在一個長運算式中的運算子後面斷句。若一個敘述分成兩行或更多行，請從第二行開始縮排並靠左對齊。

2.7 判斷：等號運算子和關係運算子

- 圖2.14列出本章介紹的運算子優先權與結合性。
- 運算子的優先權是從上到下遞減。
- 除了指定運算子=以外，所有運算子都是從左到右進行運算。

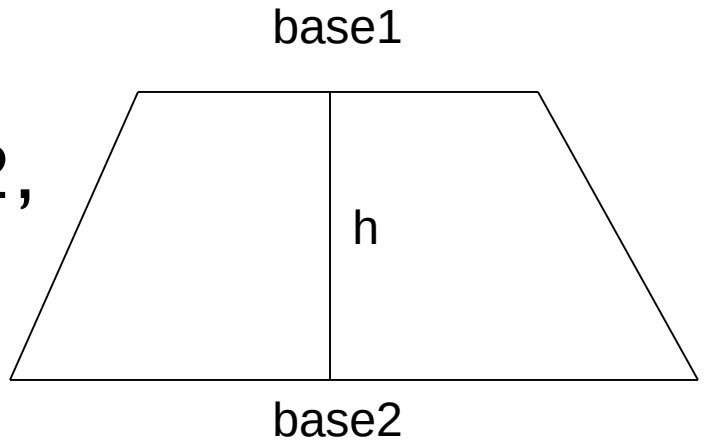
運算子				結合性	型別
()				從左到右	小括號
*	/	%		從左到右	乘、除法及模數運算子
+	-			從左到右	加、減法運算子
<<	>>			從左到右	串流插入 / 擷取運算子
<	<=	>	>=	從左到右	關係運算子
==	!=			從左到右	等號運算子
=				從右到左	指定運算子

圖 2.14　目前討論過的運算子優先順序和結合性

良好的程式設計習慣 2.15

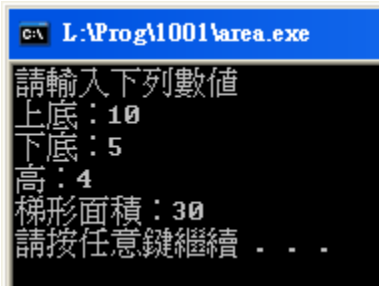
撰寫含有許多運算子的運算式時，請參考運算子的優先順序與結合性。確認運算式中的運算子是依期望的順序執行。若無法確認複雜運算式中的計算順序，則可將運算式分解成較小的敘述，或用小括號強制其計算順序，如同代數運算式的做法。請再三注意，有些運算子是從右到左運算，而不是從左到右，例如指定運算子 (=)。

計算梯形面積



假設上底 base1, 下底 base2,
高h, 算出的面積 area皆為
整數型態

程式輸入：三個整數上底、下底、高
程式輸出：梯形面積



判斷奇偶數

輸入一數，印出該數為奇數或是偶數

- 地球距離火星5千5百萬公里，無線電波傳遞的速度為每秒30萬公里，請問從地球發送無線電訊號至火星要多少秒？約合幾分鐘？

- 請撰寫一程式，輸入一三位數(100–999)，輸出其各位數字和(個位數字+十位數字+百位數字)。
- 例： 輸入 137
  輸出 11

- 輸入兩分數，判斷並印出此二數是否相等