# 2-變數與輸入、輸出

# 2.1 輸出

## 使用 cout 輸出資料

在 Code::Blocks 裡建立一個專案後，它會自動產生這樣一個程式架構。

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

## main function

其中的 `main()` 稱為主函數(main function)，它是程式的起點。程式在啟動後，會由 main 裡的第一行開始依序執行下去。

`return 0;` 表示帶著回傳值 0，返回呼叫 main() 的地方，以這個例子來說就是返回作業系統。

```
int main()
{
    // 要做的事情寫在這裡面
    return 0;
}
```

剩下的那行就是真正在做的事情，程式執行後會在畫面上看到。

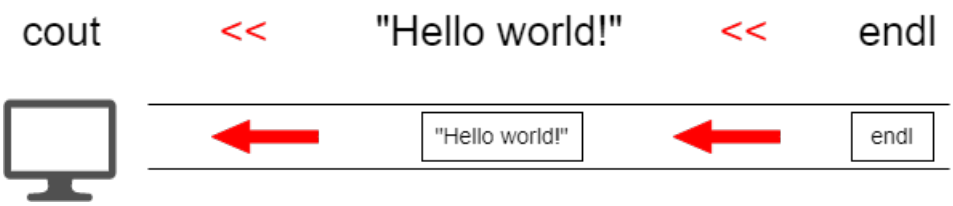
```
Hello world!
```

單行註解：雙斜線 // 開始到該行結尾都屬於「註解」，用來對程式碼做說明。這是給人類看的，對執行完全沒影響。

## 標準輸出

`cout` 是用來將資料輸出到標準輸出(standard output / stdout)，而一般的標準輸出指的是螢幕。

我們可以把它想像成，資料沿著 `<<` 的方向流到螢幕那邊。




至於 `endl` 則是 end of line，即「換行」的意思。如果把程式修改一下。

```
cout << "Hello" << endl << "world!";
```

那麼輸出就會變這樣。

```
Hello
world!
```

提醒：程式碼內容有更動後，都要[save]->[build]，才能[run]。 因為 [build] 會根據你目前的程式碼做出新的執行檔。若沒這麼做，你執行的還是之前舊的執行檔。

# 字串 和 運算式

比較一下這兩行程式碼有什麼不同。

```
cout << "2+3" << endl;
cout << 2+3 << endl;
```

看起來都是 2+3，執行結果卻大不相同。

```
2+3
5
```

第一行輸出的是 `"2+3"`，前後有雙引號。雙引號框起來的內容都會被視為文字，整個被當作字串(string)來處理，所以你寫什麼樣子，它就輸出什麼樣子。

第二行輸出的是 `2+3`，前後**沒有**雙引號。在這種情況下，cin 會等待 `2+3` 這個運算式的結果被計算出來，再將其結果輸出。因此我們會看到第二行的輸出結果是 5。

## 練習

請推測以下程式執行後的輸出結果為何？

自己寫下結果後，再實際輸入這個程式，觀察執行結果。

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    cout << 1 + 2 << endl;
    cout << 1 + 2 * 3 - 4 << endl;
    cout << (1 + 2) * (3 - 4) << endl;
    cout << "(1 + 2) * 5 / 3 = " << (1 + 2) * 5 / 3 << endl;

    return 0;
}
```

# 2.2 變數與輸入

像 `1`, `24`, `3.14` 這樣的數，我們稱為 **字面常數**(literal constant)，它的值是固定不變的。

另外像我們在數學代數中用到的 `x`, `y`, `z` 等，則稱為 **變數** ，它的值可以改變。

在電腦程式中，變數是很重要的。它可以用來儲存輸入的資料，計算中的數值，表示某個狀態等。

## 使用 cin 輸入資料

下面這段程式在執行之後，會先詢問你的年齡，在你輸入年齡並按下 [Enter] 後，輸出 `"You are xx years old."`。這個 `xx` 會是你輸入的值。

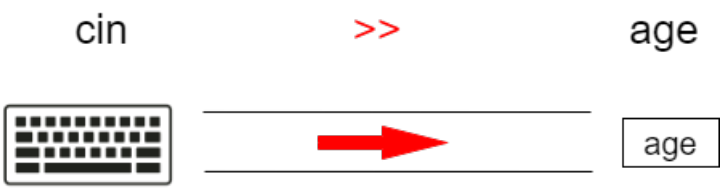
```
int age;

cout << "How old are you?";
cin >> age;
cout << "You are " << age << " years old." << endl;
```

### 標準輸入

在程式中出現的 `cin` 是用來由 **標準輸入(standard input)** 將資料讀入電腦，我們可以把它想像成和 `cout` 相反的流向。

一般來說標準輸入指的是鍵盤的輸入，而輸入的值必須被存放到電腦裡，供後續運算和使用。




### 變數

我們可以把變數想像成是一塊有名字的記憶體。但是它除了有名字之外還有 **型別** ，一個型別為整數(integer)的變數，裡面只能放整數；型別為字串(string)的變數，裡面只能放字串。

### 宣告

在上面程式碼的第一行 `int age;` ，是在 **宣告(declare)** 這個變數。每一個變數在使用前都必須先宣告，明確指出變數的型別和名字。

一個典型的變數宣告，長這個樣子。

**型別 變數名;**

例如：

**int age;**

如果不宣告就使用變數，會發生什麼事呢？

```
#include <iostream>

using namespace std;

int main()
{
    cout << "How old are you?";
    cin >> age;
```

```
    cout << "You are " << age << " years old." << endl;

    return 0;
}
```

在上面這個程式的第 8 行，我們想使用 age 這個變數。但是往前看卻沒有看到這個變數的宣告。

這個程式在編譯時，編譯器(compiler)會發出如下的錯誤訊息。

```
main.cpp: In function ‘int main()’:
main.cpp:8:12: error: ‘age’ was not declared in this scope
    8 |     cin >> age;
      |            ^~~
```

編譯器的錯誤訊息都是英文的，但是同學們一定要學會看錯誤訊息，看懂錯誤訊息可以讓你很快抓到重點，把錯誤修正。

在這段錯誤息裡

- 第一行是告訴我們，它抓到錯誤的位置在 `main.cpp` 這個檔案裡的 `int main()` 函數裡。
- 第二行可以看到精確的位置，第 8 列(row)，第 12 行(column)。所以到第 8 列第 12 個字元的位置，你就可以看到這個錯誤訊息描述的 'age'。
- 繼續往下看 `error` 表示這是個「錯誤」，不修正它程式就無法成功編譯執行。
- 之後則是錯誤的描述 `‘age’ was not declared in this scope`，它說「這個 'age' 沒有在這個範圍內宣告」(它有在下面把位置標記給你看)

> 稍後我們還會看到編譯器給出 `warring` 也就是「警告」的狀況，這是編譯器發現某處可能有問題，但程式依然可以完成編譯並執行。

看懂錯誤訊息後，下一步就是修正它。我們在前面補上宣告即可。

```
#include <iostream>

using namespace std;

int main()
{
    int age; // <-- 我們在這裡補上宣告
    cout << "How old are you?";
    cin >> age;
    cout << "You are " << age << " years old." << endl;

    return 0;
}
```

# 變數的資料型別和名字

前面提到了兩個重點

- 變數在使用前要 **宣告(declare)**
- 宣告時要明定其 **「型別」** 和 **「名字」**

## 變數名命規則

我們可以自行命名每一個變數，但是必須遵守以下規則：

1. 變數名稱的第一個字元必須是底線 `_` 或英文字母 `A~Z, a~z`
2. 除了第一個字元外，變數名稱只能由底線 `_` 、英文字母 `A~Z, a~z` 和數字 `0~9` 組成。

- 合法的變數名稱例子，如：`Age`, `age`, `length`, `_name`, `id1246`
- 不合法的變數名稱例子，如：`369city` (開頭不能是數字), `my#name`(變數名稱不能用 #)

**注意！** C++裡的變數名稱是有區分大小寫的，也就是 `Age` 和 `age`，會被視為 2 個不同的變數。

## 常用的基本資料型別

| 名稱 | 關鍵字 | 大小 | 範圍 | 備註 |
|---|---|---|---|---|
| 字元 | char | 1 Byte | $0 \sim 255$ | • ASCII<br>• 如：'a', '@' |
| 整數 | int | 4 Byte | $-2^{31} \sim 2^{31}-1$ | 如：12, -65 |
| 無號整數 | unsigned int | 4 Byte | $0 \sim 2^{32}-1$ | 如：23, 656372 |
| 單精確浮點數 | float | 4 Byte | | • IEEE 754<br>• 如：3.14, 5.0 |
| 雙精確浮點數 | double | 8 Byte | | • IEEE 754<br>• 如：3.14, 5.0 |
| 字串 | string | | | 如："Peter" |

資料型別牽涉到資料如何被儲存到記憶體裡，以及記憶體中的資料要如何被解讀。

## 避免誤用或使用不合適的型別

在下面這個用半徑計算圓周長的程式裡，我們宣告了一個名為 pi 的整數(int)型別變數，但是卻把一個浮點數 3.14 放入這個變數裡。

```
    int r;
    cout << "請輸入半徑 r:";
    cin >> r;

    int pi;
    pi=3.14;
    cout << "半徑為 " << r << " 的圓，其周長為 " << 2*pi*r << endl;
```

編譯時沒有出現任何錯誤訊息，執行結果如下。

```
請輸入半徑 r:1
半徑為 1 的圓，其周長為 6
```

如果把第 5 行的 pi 宣告成浮點數

```
    double pi;
```

執行結果則為

```
請輸入半徑 r:1
半徑為 1 的圓，其周長為 6.28
```

修改前因為 int 無法儲存小數的值，所以在 `pi=3.14;` 這裡發生了一些狀況。

3.14 本來是個浮點數，因為被指定(assign)到 int 型別的變數 pi，所以隱式轉型(implicit casting)為 int，喪失精確度變成了 3，過程中沒有出現任何錯誤訊息。如果我們在寫一個需要精確到小數以下 2 位的程式時，誤用 int 型別的變數來儲存數值，那麼結果可能會造成重大的損失。

如果是連喪失精確度轉型都做不到的狀況呢？

```
#include <iostream>

using namespace std;

int main()
{
    int name;

    name = "Peter";  // <--我們在這裡把字串放入整數型別的變數裡
    cout << "Hello, " << name << endl;

    return 0;
```

```
}
```

會出現錯誤訊息。

```
main.cpp: In function ‘int main()’:
main.cpp:9:12: error: invalid conversion from ‘const char*’ to ‘int’ [-fpermissive]
    9 |     name = "Peter";
      |            ^~~~~~~
      |            |
      |            const char*
```

重點在這句 `invalid conversion from ‘const char*’ to ‘int’`

不可以把 `const char*` 轉型為 `int`，因為「字串」沒辦法轉換成「整數」。

## 連續讀取多個值

cout 可以像這樣串接，一次輸出多組資料。

```
cout << "I am " << 16 << " years old.";
```

cin 也可以串接，讀取多個輸入值。

```
string name;
int age;

cin >> name >> age;
cout << name << " is " << age << " years old.";
```

原則上判斷輸入的斷句是在 `[空白]` 或 `[Enter]`

程式執行後，你可以輸入 `Peter` `[空格]` `16` `[Enter]`，或是輸入 `Peter` `[Enter]` `16` `[Enter]`

第一個輸入的值("Peter")會被放入變數 name 裡，第二個輸入的值(16)會被放入變數 age 裡。

```
Peter is 16 years old.
```

# 2.3 運算子與運算優先順序

在掌握了基本輸入、輸出之後，我們已經具備「將資料讀進電腦」，「將處理後資料送回外界」的能力，接下來重點就是中間的「處理」，也就是運算的部分。

首先我們要認識兩個名詞：

- **運算元(operands)**
- **運算子(operator)**

以 2+3 為例，`2` 和 `3` 都是運算元，`+` 是運算子。

我們可以把運算子想成是「運算符號」，運算元則是「運算的對象」。

## 指定(assign)運算子

在 C++ 中，`=` 不是 **等於(equal)** 而是 **指定(assign)**

`a = 3` 是「把 3 **指定** 給 a 這個變數」，而非表示「a 和 3 的值是相等的」

執行完這行後，變數 a 的值就會變成 3。

```
a = 3;
cout << a; // 3
```

運算子 `=` 會將其右側的運算結果，指定到左側的儲存空間。

例如 `v = 3+5` 是把 3+5 的運算結果指定給變數 v。

也就是說如果 `=` 的右側不是單純的值而是運算式，要先完成運算後，再將運算結果指定給其左側的變數。 所以，執行完這行後，變數 v 的值就會變成 8。

```
v = 3+5;
cout << v; // 8
```

想想看，以下這段程式執行後的輸出為何？

```
int a;
a = 3;
a = a + 2;

cout << a << endl;
```

其中第 2 行是把 3 指定給 a，所以執行後 a 的值為 3。

第 3 行因為 `=` 的右側是運算式 `a + 2`，因此要先完成這個運算，目前 a 的值是 3，所以 `a+2` 的運算結果是 5。

接著可以想像第3行變成 `a = 5`，所以整行執行後，a 的值為 5。

千萬不要用數學符號的角度把 `=` 當成「等於」去看待 `a = a + 2`，這行敘述，否則你會看不懂它。

### 練習

以下這段程式執行後，a 和 b 的值各為何？

```
int a = 3;
int b = 5;

a = a + b;
b = a - b;
a = a - b;
```

```
cout << "a = " << a << endl;
cout << "b = " << b << endl;
```

# 算術(arithmetic)運算子

「乘、除」 運算子的優先權高於「加、減」運算子，也就是在運算時會 **先乘除後加減**。

例如：`1+2*3-4;` 的運算結果是 `3`。

和數學運算式一樣，可以加上括號指定優先運算的部分。

例如：`(1+2)*(3-4);` 的運算結果是 `-3`。

有個需要留意的地方是，不同於我們在數學課中用小括號、中括號、大括號來一層層的指定優先運算的層次。C++裡只有小括號，不管是幾層都是用小括號來表示。中括號、大括號這兩個符號是用在其他地方。

數學課裡的 $(1+2)\times[(3-4)+2]$

在 C++ 裡是 `(1+2)*((3-4)+2)`

## 整數除法

除法運算子 `/` 有個需要注意的地方，如果它的左、右側運算元都是整數型別，那麼其運算結果也會是整數。

`cout << 5/2;` 的輸出不會是 `2.5`，而是`2`.

但 `cout << 5.0/2;`、`cout << 5/2.0;`、`cout << 5.0/2.0;` 的輸出，都是 `2.5`。

## 取餘數

另一個只能用在整數型別的 **模數(modulo)** 運算子 `%`，是用來計算兩整數相除後的餘數。

```
cout << 8%3 << endl; // 2
cout << 6%2 << endl; // 0
cout << 3%7 << endl; // 3
```

在許多演算法中，取餘數是很重要的運算，所以這是個很重要的運算子。

# 關係(Relational)運算子

下一個單元開始，我們的程式碼將不再只是單純按順序一行一行執行下去，它開始可以按照條件選擇接下來要執行哪一條分支路線。

例如：如果「a大於0」那麼.....否則.....。

那個條件在當下有兩種可能：

- 若成立，表示其為 **真(true)**
- 若不成立，表示其為 **偽(false)**

關係運算子的運算結果是 **布林值(Boolean)**。不同於整數型別的值有多種可能的值 `... -2, -1, 0, 1, 2, ...`，布林型別的值只有兩種 `true`, `false`。

以下是六個關係運算子的作用。

| 名稱 | 運算子 | 範例 | 範例運算結果 |
|---|---|---|---|
| 等於 | `==` | `2==3` | false |
| 不等於 | `!=` | `2!=3` | true |
| 大於 | `>` | `2>3` | false |
| 小於 | `<` | `2<3` | true |

| 名稱 | 運算子 | 範例 | 範例運算結果 |
| --- | --- | --- | --- |
| 大於等於 | `>=` | `2>=3` | false |
| 小於等於 | `<=` | `2<=3` | true |

# 邏輯(logical)運算子

前面提到的關係運算子，可以讓我們判斷一個條件是否成立，例如：判斷成績是否及格。

```
score>=60
```

但有時狀況比較複雜一點，例如：要產生補考名單時，我們要確認成績有沒有落在這個區間 $40\leq score<69$。也就是有2個條件要同時成立。

這時候我們可以用 **And 邏輯運算子** `&&`。

```
score>=40 && score<60
```

當 `score>=40` 為 true 而且 `score<60` 為 true，則整條式子的運算結果為 true。

**Or 邏輯運算子** `||` 則是只要其中一個運算是 true，則整條式子的運算結果就是 true。例如：只要國文或英文成績大於等於80分，就發給獎學金。

```
chinese>=80 || english>=80
```

**Not 邏輯運算子** `!` 可以把邏輯狀態反轉，就是把 true 變成 false，把 false 變成 true。

例如：fileIsOpen 若為 true，表示檔案已開啟。程式在讀取檔案內容前，要確定檔案已開啟。我們想在檔案未開啟時顯示錯誤訊息，只要在這個狀態為 true 時即可。

```
!fileIsOpen
```

因為當 fileIsOpen 為 false 時，把它用 Not 反轉後就是 true。

# 位元(bitwise)運算子

`&`, `|`, `^`, `~`, `>>`, `<<` 這幾個運算子可以在位元層級做運算，也就是對每一個 bit 做運算。我們會在之後對二進位系統有一定了解後再來討論。

# 其他的運算子

C++ 還有許多運算子，可以參考這張網上的 [運算子和優先順序圖表]。

# *2.4 C 語言的 printf( ) 格式化輸出函數

## 為什麼需要「格式化」輸出？

想像一下，如果你的程式計算出圓周率是 3.1415926，但你只想在螢幕上顯示 3.14；或者你希望輸出的成績單欄位能夠像表格一樣文字靠左對齊，數值靠右對齊。

這些都無法單純地將變數丟出來就辦到，我們需要「告訴」printf 應該用什麼「格式」來呈現資料，這就是格式化輸出的精髓。

### 標頭檔

要使用 printf，請務必在程式開頭引用標頭檔：

```
#include <cstdio>
```

`cstdio` 是指 C語言的標準輸入輸出(c standard input output)。

## printf 語法與核心：「格式化字串」

printf 的語法結構如下：

```
printf("格式化字串", 變數1, 變數2, ...);
```

它的靈魂就在於第一個參數——**格式化字串**。這個字串由兩種內容組成：

1. 一般文字：會被原封不動地輸出到螢幕上。
2. 格式指定符 (Format Specifier)：以 % 符號開頭，作為一個「佔位符」，它會被後面依序對應的變數值給取代。

這是最基本也最常用的幾種，我們先從它們開始。

| 指定符 | 對應資料類型 | 說明 |
| --- | --- | --- |
| Text | Text | Text |
| %d | int(整數) | 以十進位形式輸出整數。 |
| %f | float, double (浮點數) | 輸出浮點數 (小數)。 |
| %c | char | (字元) 輸出單一字元。 |
| %s | 字串 (char 陣列) | 輸出一整個字串。 |
| %% | 無 | 若你想顯示一個 % 符號，需使用 %%。 |

**範例 1：基本應用**

```
#include <cstdio>

int main() {
    int student_id = 101;
    float score = 85.5;
    char level = 'B';
    printf("學生學號：%d, 分數：%.1f, 等級：%c\n", student_id, score, level);
    // %.1f 的意思是浮點數只顯示到小數點後第一位
    return 0;
}
```

**執行結果：**

```
學生學號：101, 分數：85.5, 等級：B
```

# 格式化字串的各種變化與應用

接下來是今天的重頭戲。我們可以對 `%` 加上一些「修飾」，來精準控制輸出的樣式。

## 1. 控制輸出寬度 (Field Width)

我們可以指定一個數字來表示該欄位最少要佔用的寬度。如果實際內容比指定的寬度窄，預設會在**左邊**用空白補滿（也就是**靠右對齊**）。

**語法：** `%[寬度]d`、`%[寬度]f` ...

**範例 2：讓數字整齊排列**

```
#include <cstdio>

int main() {
    int num1 = 123;
    int num2 = 45;
    int num3 = 6789;

    printf("原始輸出：\n");
    printf("%d\n", num1);
    printf("%d\n", num2);
    printf("%d\n", num3);

    printf("\n指定寬度為 5 輸出 (靠右對齊)：\n");
    printf("%5d\n", num1); // 在 123 左邊補 2 個空白
    printf("%5d\n", num2); // 在 45 左邊補 3 個空白
    printf("%5d\n", num3); // 寬度不足 5，但數字不會被切斷，會完整顯示
    return 0;
}
```

**執行結果：**

```
原始輸出：
123
45
6789

指定寬度為 5 輸出 (靠右對齊)：
  123
   45
 6789
```

## 2. 控制對齊方式 (Alignment)

如果我們想靠左對齊，只要在寬度數字前加上一個負號 `-` 即可。

**語法：** `%-[寬度]d`、`%-[寬度]s` ...

**範例 3：文字的靠左與靠右**

```
#include <cstdio>

int main() {
    char product1[] = "Apple";
    char product2[] = "Banana";

    printf("--- 商品清單 (寬度 10) ---\n");
    printf("靠右對齊: |%10s|\n", product1);
    printf("靠左對齊: |%-10s|\n", product2);
    printf("-------------------------\n");
    return 0;
```

```
}
```

**執行結果：**

```
--- 商品清單 (寬度 10) ---
靠右對齊: |     Apple|
靠左對齊: |Banana    |
----------
```

---

對於浮點數 (%f)，我們可以用 .數字 來指定要顯示到小數點後幾位。注意：系統會自動進行四捨五入。

語法：`%.[位數]f`

**範例 4：計算圓面積並控制精度**

```
#include <cstdio>

int main() {
    double pi = 3.1415926535;

    printf("原始數值：%f\n", pi);
    printf("顯示到小數點後 2 位：%.2f\n", pi); // 輸出 3.14
    printf("顯示到小數點後 4 位：%.4f\n", pi); // 輸出 3.1416 (注意看，有四捨五入！)
    printf("不顯示小數：%.0f\n", pi);          // 輸出 3
    return 0;
}
```

執行結果：

```
原始數值：3.141593
顯示到小數點後 2 位：3.14
顯示到小數點後 4 位：3.1416
不顯示小數：3
```

---

我們也可以將寬度和精度結合起來，創造出更完美的排版。

語法：`%[總寬度].[小數位數]f`

**範例 5：顯示商品價格**

```
#include <cstdio>

int main() {
    float price1 = 5.99;
    float price2 = 123.5;

    // 總寬度為 8，小數點後 2 位
    printf("價格清單 (總寬度 8)：\n");
    printf("|%8.2f|\n", price1);
    printf("|%8.2f|\n", price2);
    return 0;
}
```

**執行結果：**

```
價格清單 (總寬度 8)：
|    5.99|
|  123.50|
```

**解說：**

總共佔 8 個字元寬，並且小數點後面固定顯示 2 位，不足的會補 0。

---

## 3. 補零 (Zero Padding)

如果希望在靠右對齊時，不是補空白而是補 `0`，只要在寬度數字前加上 `0` 即可。這常用於學號、時間等場合。

**語法：** `%0[寬度]d`

**範例 6：顯示學號**

```
#include <cstdio>

int main() {
    int id1 = 7;
    int id2 = 123;
    printf("三年一班 座號列表：\n");
    printf("補空白：%3d\n", id1);
    printf("補零：%03d\n", id1); // 寬度 3，不足處補 0
    printf("補零：%03d\n", id2);
    return 0;
}
```

**執行結果：**

```
三年一班 座號列表：
補空白：  7
補零：007
補零：123
```

---

# 總結與速查表

以下表格提供同學們需要時查詢使用：

| 分類 | 語法 | 說明與範例 |
| --- | --- | --- |
| 基礎類型 | %d | 輸出十進位整數 (int)。 `printf("%d", 100);` → `100` |
| | %f | 輸出浮點數 (float, double)。 `printf("%f", 12.34);` → `12.340000` |
| | %c | 輸出單一字元 (char)。 `printf("%c", 'A');` → `A` |
| | %s | 輸出字串 (char 陣列)。 `printf("%s", "Hi");` → `Hi` |
| | %% | 輸出 % 符號本身。 `printf("100%%");` → `100%` |
| 寬度控制 | %[n]d | 輸出寬度至少為 n 的整數，靠右對齊。 `printf("%4d",12);` → `  12` |
| | %- [n]d | 輸出寬度至少為 n 的整數，靠左對齊。 `printf("%-4d",12);` → `12  ` |
| 精度控制 | %.[n]f | 輸出浮點數，顯示到小數點後 n 位。 `printf("%.2f", 3.14159);` → `3.14` |
| 組合使用 | %[w].[p]f | 總寬度為 w，小數點精度為 p。 `printf("%6.2f", 3.14159);` → `3.14` |
| 特殊旗標 | %0[n]d | 輸出寬度為 n 的整數，不足處在左邊補 0。 `printf("%04d", 55);` → `0055` |

這些符號放在字串中會有特殊功能。

| 序列 | 名稱 | 功能 |
| --- | --- | --- |
| \n | 換行符 | 將游標移至下一行的開頭。 |
| \t | 定位符 | (Tab) 將游標移至下一個定位點，常用於對齊欄位。 |
| \ | 反斜線 | 顯示一個 \ 符號。 |
| " | 雙引號 | 在字串中顯示一個 " 符號。 |

# 練習題作業

請同學動手實作，加深對 printf 格式化的理解。

## 練習題 1：我的個人檔案

- **目標：** 練習 %s, %d, %.1f 的基本使用。
- **說明：** 請宣告變數來儲存姓名、年齡、身高(公尺)，並依照以下格式輸出。

**輸出範例：**

```
姓名：陳月光
年齡：17 歲
身高：1.7 公尺
```

---

## 練習題 2：商品價目表

- **目標：** 練習使用寬度、對齊與小數點精度，製作對齊的表格。
- **說明：** 有三樣商品及其價格如下：
  - "Milk": 65.5 元
  - "Bread": 42 元
  - "Juice": 51.25 元

請使用 printf 格式化功能，輸出如下對齊的價目表。商品名稱欄位寬度為 10 且靠左對齊，價格欄位總寬度為 8 且顯示到小數點後 2 位。

**輸出範例：**

```
+------------+----------+
| Item       | Price    |
+------------+----------+
| Milk       |    65.50 |
| Bread      |    42.00 |
| Juice      |    51.25 |
+------------+----------+
```

---

## 練習題 3：數位時鐘

- 目標：練習使用補零 0 的技巧。
- 說明：請宣告三個整數變數 h, m, s 分別代表時、分、秒，並賦值 (例如 h=8, m=5, s=30)。請使用 printf 輸出 HH:MM:SS 的格式，也就是不足兩位數時要補零。

**輸出範例：**

```
目前時間為：08:05:30
```

# *2.5 C 語言的 scanf( ) 格式化輸入函數

我們已經學會如何用 printf 讓程式輸出精美的訊息。但一個真正有用的程式，不僅要會「說」，更要會「聽」。它需要接收使用者的指令、數據，才能進行下一步的處理。

scanf (scan formatted) 就是這座溝通的橋樑。它會暫停程式的執行，靜靜地等待使用者從鍵盤輸入資料，然後依照我們指定的「格式」去解析這些輸入，並將它們存放到對應的變數中。

同樣地，要使用 scanf，請務必在程式開頭引用標頭檔：

```
#include <cstdio>
```

## 2.5.1- scanf 的核心語法與「&」的秘密

scanf 的語法看起來和 printf 有點像，但有一個關鍵且絕對不能忘記的區別。

```
scanf("格式化字串", &變數1, &變數2, ...);
```

- **格式化字串：** 由一或多個「格式指定符」組成，用來告訴 scanf 使用者將會輸入什麼類型的資料。例如，"%d" 代表使用者會輸入一個整數。
- **& (取址運算子)：** 這是 scanf 最重要、也最容易出錯的地方！

讓我們用一個生活化的比喻來理解：

想像一下，變數是一個「置物櫃」，裡面可以存放資料。

- 當你使用 `printf("%d", score);` 時，你是告訴 printf：「打開 score 這個櫃子，把裡面的東西（值）拿出來秀給大家看。」
- 當你使用 `scanf("%d", &score);` 時，你是告訴 scanf：「我給你 score 這個櫃子的地址（&score），請你把使用者輸入的東西，親自送到這個地址的櫃子裡放好。」

scanf 需要的是「存放資料的地址」，而不是「變數裡現有的值」。所以，除了字串陣列（我們稍後會提）以外，`幾乎所有變數在使用 scanf 時都必須在前面加上 & 符號！`

## 2.5.2- scanf 的各種應用與常見陷阱

常用的格式指定符：

| 指定符 | 對應資料類型 | 說明 |
| --- | --- | --- |
| %d | int | 讀取一個十進位整數。 |
| %f | float | 讀取一個浮點數。 |
| %lf | double | 注意！ 讀取 double 類型時要用 %lf (long float)，這是新手常犯的錯誤。 |
| %c | char | 讀取一個單一字元。 |
| %s | 字串 (char 陣列) | 讀取一個字串（但遇到空白、Tab或換行時會停止）。 |

**範例 1：讀取學生的基本資料**

```
#include <cstdio>

int main() {
    int age;
    double height;

    printf("請輸入你的年齡：");
    scanf("%d", &age);
```

```
    printf("請輸入你的身高 (公尺)：");
    scanf("%lf", &height); // 讀取 double，使用 %lf

    printf("好的，你 %d 歲，身高 %.2f 公尺。\n", age, height);
    return 0;
}
```

**執行過程：**

```
請輸入你的年齡：17 (使用者輸入後按 Enter)
請輸入你的身高 (公尺)：1.75 (使用者輸入後按 Enter)
好的，你 17 歲，身高 1.75 公尺。
```

你可以在格式化字串中放置多個指定符，scanf 會要求使用者一次輸入多個值，並用空白、Tab 或換行鍵來分隔它們。

## 範例 2：輸入座標

```
#include <cstdio>

int main() {
    int x, y;
    printf("請輸入一個二維座標 (例如: 15 30)：");
    scanf("%d %d", &x, &y);
    printf("你輸入的座標點為 (%d, %d)\n", x, y);
    return 0;
}
```

**執行過程：**

```
請輸入一個二維座標 (例如: 15 30)：15 30
你輸入的座標點為 (15, 30)
```

這是 scanf 最經典的陷阱！當你讀取完一個數字後，緊接著要讀取一個字元時，常常會出問題。

**範例 3 (錯誤示範)：**

```
#include <cstdio>

int main() {
    int choice;
    char confirm;
    printf("請選擇項目 (1-3): ");
    scanf("%d", &choice);
    printf("你確定嗎? (Y/N): ");
    scanf("%c", &confirm); // 這行會出問題
    printf("你的選擇是 %d, 確認字元是 '%c'\n", choice, confirm);
    return 0;
}
```

**執行過程：**

```
請選擇項目 (1-3): 2 (使用者輸入 2 後按 Enter)
你確定嗎? (Y/N): 你的選擇是 2, 確認字元是 '
'
```

**問題分析：**

當你輸入 2 並按下 Enter 鍵時，你其實輸入了兩個字元：`'2'` 和 `'\n'` (換行符號)。

`scanf("%d", ...)` 只讀走了 `'2'`，那個 `'\n'` 還留在輸入緩衝區中。輪到 `scanf("%c", ...)` 時，它立刻把還留著的 `'\n'` 讀走了，導致程式根本不等待你輸入 Y 或 N。

**解決方案：**

在 `%c` 前面加一個空格，`" %c"`。這個空格會告訴 scanf：「請忽略前面所有空白類的字元 ( 包含空格、Tab、換行

符），然後再讀取下一個真正的字元。」

**範例 3 (正確寫法)：**

```
#include <cstdio>
int main() {
    int choice;
    char confirm;
    printf("請選擇項目 (1-3): ");
    scanf("%d", &choice);
    printf("你確定嗎? (Y/N): ");
    scanf(" %c", &confirm); // 在 %c 前加一個空格
    printf("你的選擇是 %d, 確認字元是 '%c'\n", choice, confirm);
    return 0;
}
```

%s 雖然方便，但`它遇到任何空白字元（空格、Tab、換行）就會停止讀取`。

**範例 4：讀取姓名**

```
#include <cstdio>

int main() {
    char name[30];
    printf("請輸入你的英文全名 (例如: Peter Pan): ");
    scanf("%s", name); // 注意：字串陣列 name 本身就是位址，所以不用加 &
    printf("你好, %s！\n", name);
    return 0;
}
```

**執行過程：**

```
請輸入你的英文全名 (例如: Peter Pan): Peter Pan
你好, Peter！
```

**問題分析：**

scanf 只讀到了 `"Peter"` 就因為遇到空格而停止了，`"Pan"` 則被留在了後面。

這也是為什麼在 C++ 中，當需要讀取一整行含有空格的文字時，我們未來會學習更適合的 cin.getline() 或 fgets() 函式。

# 2.5.3- 總結與速查表

scanf 是程式接收輸入的基礎，雖然有些小陷阱，但只要小心使用，它依然非常強大。

| 指定符 | 對應資料類型 | 關鍵提醒 |
| --- | --- | --- |
| %d | int | 變數前記得加 &。例如 `scanf("%d", &num);` |
| %f | float | 變數前記得加 &。例如 `scanf("%f", &price);` |
| %lf | double | 務必使用 %lf，而非 %f。變數前記得加 &。例如 `scanf("%lf", &pi);` |
| %c | char | 變數前記得加 &。若前面有其他輸入，建議用 `" %c"` 來清除換行符。 |
| %s | char[] | 變數前不用加 &。只能讀取不含空白的字串。 |

# 2.5.4- 練習題作業

## 練習題 1：華氏溫度轉換

- **目標：** 練習讀取浮點數 (double) 並進行計算。
- **說明：** 請使用者輸入攝氏溫度，程式將其轉換為華氏溫度後輸出。
- **公式：** 華氏 = 攝氏 * 9 / 5 + 32

- 執行範例：

```
請輸入攝氏溫度：25.0
轉換後的華氏溫度為：77.0
```

## 練習題 2：簡易對話程式

- **目標：** 綜合練習 %d, %s。
- **說明：** 撰寫一個程式，詢問使用者的名字和學號，然後向他打招呼。
- **執行範例：**

```
你好！請問你叫什麼名字？ (請輸入英文名)
David
你的學號是幾號？
10123
你好, David (學號: 10123)，歡迎使用本系統！
```

## 練習題 3：解決字元輸入問題

- **目標：** 練習處理 %c 的換行符陷阱。
- **說明：** 撰寫一個程式，讓使用者輸入一個整數代表分數，然後再輸入一個字元代表評等 (A/B/C/D)。最後將兩者一起輸出。請務必確保程式能正確等待使用者輸入評等。
- **執行範例：**

```
請輸入你的分數：88
請輸入你的評等 (A-F)：A
你的成績是 88 分，評等為 A。
```