【第二講】

# 資料型態、運算子與表示式

講師: 李根逸 (Ken-Yi Lee), E-mail: feis.tw@gmail.com

★ ★ ★ ★

★ ★ ★ ★

# 課程大綱

# 資料型態 (Data type)

- 在高階語言中，為了能夠方便有效（省時省空間）的撰寫程式碼並做出各種複雜的運算，我們需要使用多種資料型態
  - 例如：整數, 小數和文字處理等 ...
- 電腦內部是使用位元 (Bit) 這個基本單位來表示資料並儲存於記憶單元 (記憶體) 或輔助記憶單元 (硬碟) 中。
  - 每個位元只可以表示 0 或 1 兩種值
- 任何資料型態的資料都可以轉換成由一串位元來表示
  - 換句話說，資料型態就是要告訴電腦要怎麼去解釋某一串位元資料，我們可以規定如何對不同的型態做運算

# C 常見的內建資料型態

| 資料型態 | 名稱 | 大小 (bytes) | 範例 |
|---|---|---|---|
| 短整數 (Short Integer) | **short int** | 2 | 32 |
| 整數 (Integer) | **int** | 4 | 32 |
| 長整數 (Long Integer) | **long int** | 4 | 32 |
| 字元 (Character) | **char** | 1 | '3' |
| 單精度浮點數 (Single Precision Floating Point) | **float** | 4 | 3.2 |
| 雙精度浮點數 (Double Precision Floating Point) | **double** | 8 | 3.2 |
| 無 | **void** | (無) | (無) |

《實作相依》：意指語言標準內容並沒有強制的規定，在使用不同編譯器或設定的情況下，可能會不一樣

大小是《實作相依》

# 使用 sizeof 看大小

- **語言標準內對資料型態沒有嚴格定義大小，隨著編譯器與設定的不同而可能不同。我們只知道在同樣的編譯器與設定中，同樣資料型態的大小是固定的。**
  - 例如 **int** 不一定要是 4 個位元組大，只是我們現在一般的電腦架構與作業系統通常是。而在 32-bit 編譯器內， **long int** 的大小可能是 4 個位元組， 但在 64-bit 編譯器中， **long int** 的大小可能是 8 個位元組
- **sizeof 是一個特殊的運算子，會得到某變數或資料型態在該平台編譯後佔有記憶體的大小。我們表示記憶體大小所使用的單位是位元組 (byte) ，而一個位元組 (byte) 通常等於八個位元 (bit)**
  - 開啟範例檔 **`sizeof.cpp`** 並執行看看

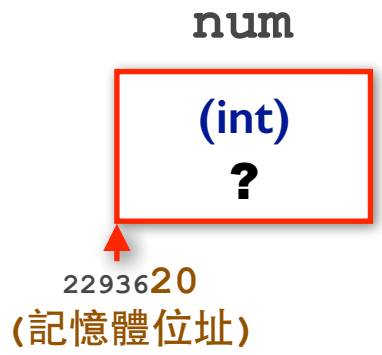
# 變數宣告

- 變數名稱在使用前，需要先進行宣告讓編譯器知道：
  - 資料型態 變數名稱;

```
int num;
```

  - 資料型態 變數名稱 = 初始值;

```
int num = 0;
```




在宣告時給予初始值這動作我們稱為初始化

名稱通常為英文字母大小寫、數字和底線構成，數字不能開頭，大小寫不同也代表不同的名稱

- 保留字 **(keywords)**：
  - C 語言中下列名稱 (保留字) 無法作為變數名稱
    - **auto,break,case,char,const,continue,default, do,double,else,enum,extern,float,for,goto,if, int,long,register,return,short,signed,sizeof, static,struct,switch,typedef,union,unsigned, void,volatile,while**

# 不同資料型態間的差異

- 表示的資料意涵不同：
  - 整數 (**int**) 與字元 (**char**)
- 表示的原理不同：
  - 整數 (**int**) 與浮點數 (**float**)
- 可表示的範圍大小不同：
  - 短整數 (**short int**) 與長整數 (**long int**)
- 可表示的精確度大小不同：
  - 單精度浮點數 (**float**) 與倍精度浮點數 (**double**)

與大小有關

- 有無正負數 **(有號與無號)**
  - 有號整數 (**int**) 與無號整數 (**unsigned int**)

# 整數型態的可表示範圍

- 資料型態可表示的範圍與他佔記憶體的『大小』有關
  - 每個位元可以表示兩種值 (0 或 1)
  - 每個位元組如果有 8 個位元：
    - 每個位元組可以表示 $2^8 = 256$ 種值
    - 在可表示正負數 (即有號 [**signed**]) 的情況下，可表示的整數範圍會是從 -128 到 127 (共 256 個數字)
    - 當無號 [**unsigned**] 時，可表示的整數範圍是從 0 到 255
  - 如果一個資料型態有 4 個位元組，則可以表示：
    - $2^{8x4} = 2^{32} = 4{,}294{,}967{,}296$ 種值
    - 4 個位元組大的 **int** 可表示範圍是從 −2,147,483,648 到 2,147,483,647 (約九位數有效數字)
    - 4 個位元組大的 **unsigned int** 的可表示範圍就是從 0 到 4,294,967,295 (約九位數有效數字)

運算結果超出可表示範圍稱為『溢位』，在溢位的情況下，值會變多少大部分情況是《未定義行為》

不要因為範圍可以大一點就用無號數

# 【補充】int 的位元表示法

32 bits (4 bytes)

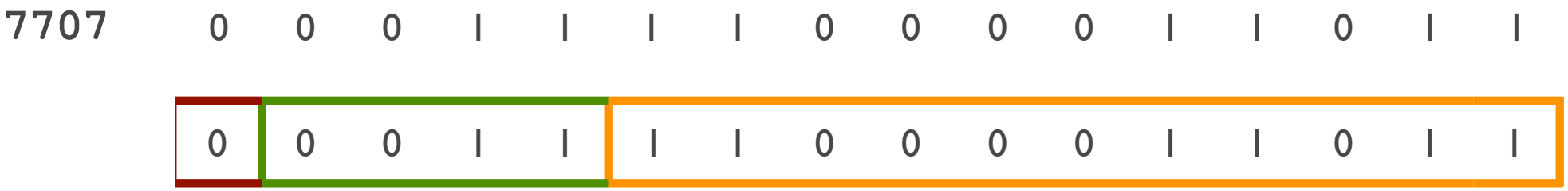
| 十進位表示法 | 二進位表示法 |
|---|---|
| 1 | 00000000 00000000 00000000 00000001 |
| 2 | 00000000 00000000 00000000 00000010 |
| 5 | 00000000 00000000 00000000 00000101 |
| 255 | 00000000 00000000 00000000 11111111 |
| 256 | 00000000 00000000 00000001 00000000 |
| 0 | 00000000 00000000 00000000 00000000 |
| 2147483647 | 01111111 11111111 11111111 11111111 |
| -1 | 11111111 11111111 11111111 11111111 |
| -2 | 11111111 11111111 11111111 11111110 |
| -255 | 11111111 11111111 11111111 00000001 |
| -2147483648 | 10000000 00000000 00000000 00000000 |

負數用 2 的補數表示法: 將正數表示法的 0 和 1 互換後再加 1

如何表示整數是《實作相依》行為，這裡只是列出一般的作法作為參考

# 浮點數表示法

- 浮點數 **(floating point)** 是用來將實數數位化表示的一種表示法
  - 我們現在所用的是由 IEEE 制定的浮點數表示標準
- 簡單來看，浮點數的表示法將位元分成三個區塊
  - 符號位元 (1 Bit), 指數部分, 小數部分




\+ 3

\+ $0.1563 \times 10^3$

1563

**float** 有效數字約6位，**double** 約15位

(實際上格式比較複雜，這裡只是個概念的說明。細節可參考 http://goo.gl/imXGf)

# printf 與 scanf 的格式字串

| 資料型態 | 名稱 | 格式符 |
|---|---|---|
| 短整數 (Short Integer) | **short int** | `%hd` |
| 整數 (Integer) | **int** | `%d` |
| 長整數 (Long Integer) | **long int** | `%ld` |
| 字元 (Character) | **char** | `%c` |
| 單精度浮點數 (Single Precision Floating Point) | **float** | `%f` |
| 雙精度浮點數 (Double Precision Floating Point) | **double** | `%f, %lf`<br>`printf  scanf` |

# 【範例】使用浮點數

- 請修改程式讓使用者分別輸入三個整數後，算出三個整數的和、平均值、乘積並顯示給使用者看 (四捨五入到小數點後三位)
  - 注意: 平均值可能具有小數而且使用者可能輸入的數值帶有小數
  - 變數宣告時須改為用 **float** 宣告
  - **scanf** 和 **printf** 須使用 `%f` 來讀入或輸出 **float**
  - **printf** 的格式字串可以加上數字表示位數
    - `%.3f` 表示印出浮點數並四捨五入到小數點後第三位

【範例】 calc.cpp

# 字面常數的型態

| 字面 | 資料型態 | 名稱 |
|---|---|---|
| `3` | 整數 (Integer) | **int**<br>**long int** |
| `3u` | 無號整數 (Unsigned Integer) | **unsigned int** |
| `3l` | 長整數 (Long Integer) | **long int** |
| `3.` | 雙精度浮點數<br>(Double Precision Floating Point) | **double** |
| `3.f` | 單精度浮點數<br>(Single Precision Floating point) | **float** |
| `'3'` | 字元 (Character) | **char** |

【補充】在能表示的情況下整數選前者

【範例】 `constant.cpp`

# 不同型態間的轉換

- **編譯器為了讓你的運算式合理，可能會試著幫你做型態的自動轉換 (隱性轉型)。**
  - 不同基本資料型態間的自動轉換 (隱性轉型) 通常以『可表示範圍大』的為準
    - 例如： `4 / 3` 與 `4 / 3.`
      - **4 / 3** 時，計算結果的資料型態會是 **int**
      - **4 / 3.** 時，**4** 是 **int** 而 **3.** 是 **double**。計算時會先將 **4** 轉換成 **double** 後再除以 **3.**，計算結果資料型態是 **double**。
  - 你也可以用強制的方式進行型態轉換 (顯性轉型)：
    - 例如： `(double) 4 / 3`
      - **4** 會先被強制轉換為 **double** 型態 (即 **4.** )，再試著去除以 **3**，此時 **3** 也被動的隱性轉型成 **double** 型態 (即 **3.**)。計算結果的資料型態會是 **double**

# 【範例】隱性轉型與格式

- 請開啟範例檔，並猜測執行結果
- 提示與解釋：
  - A = B 是指將 B 的值給 A，此時如果 B 的型態與 A 不同則可能會造成無法編譯或發生隱性轉型將 B 轉為 A 的型態
    - 轉型時，浮點數轉為整數是無條件捨去，通常值會變得不精確
  - 要小心的是，使用 **printf** 或 **scanf** 時，輸入的引數並不會自動的轉型
    - 例如: `printf("%d", 3.);`
      - 會因為 **3.** 是 **double** 卻當成 **int** 印而失敗，產生不易預期的結果
    - **printf** 跟 **scanf** 這算是特例中的特例，但是我們常常使用到。

【範例】 casting.cpp

# 【範例】大數計算

- 試寫一程式輸入兩個五位數整數後輸出他們的乘積
  - **int** 的資料型態一般情況下約可表示 $\log(2^{31}) \sim 9$ 位有效整數
- 【補充】使用範圍更大的整數型態
  - C++99 (新版 C++ 標準) 和 C99 (新版 C 標準) 中有一個 **long long int** 的資料型態至少可表示約 $\log(2^{63}) \sim 19$ 位有效整數
  - 在 Dev C++ 中，請在 **printf** 內使用 **`%I64d`** 來列印 **long long int** 型態數值
    - 在 GCC 編譯器中，**long long int** 是一個語言擴充功能

【範例】 bignum.cpp

| ASCII 值 十進位 | 十六進位 | 字元符號 | ASCII 值 十進位 | 十六進位 | 字元符號 | ASCII 值 十進位 | 十六進位 | 字元符號 | ASCII 值 十進位 | 十六進位 | 字元符號 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 000 | 00 | (null) | 032 | 20 | | 064 | 40 | @ | 096 | 60 | ` |
| 001 | 01 | ☺ | 033 | 21 | ! | 065 | 41 | A | 097 | 61 | a |
| 002 | 02 | ● | 034 | 22 | " | 066 | 42 | B | 098 | 62 | b |
| 003 | 03 | ♥ | 035 | 23 | # | 067 | 43 | C | 099 | 63 | c |
| 004 | 04 | ♦ | 036 | 24 | $ | 068 | 44 | D | 100 | 64 | d |
| 005 | 05 | ♣ | 037 | 25 | % | 069 | 45 | E | 101 | 65 | e |
| 006 | 06 | ♠ | 038 | 26 | & | 070 | 46 | F | 102 | 66 | f |
| 007 | 07 | • | 039 | 27 | ' | 071 | 47 | G | 103 | 67 | g |
| 008 | 08 | ◘ | 040 | 28 | ( | 072 | 48 | H | 104 | 68 | h |
| 009 | 09 | ○ | 041 | 29 | ) | 073 | 49 | I | 105 | 69 | i |
| 010 | 0A | ◙ | 042 | 2A | * | 074 | 4A | J | 106 | 6A | j |
| 011 | 0B | ♂ | 043 | 2B | + | 075 | 4B | K | 107 | 6B | k |
| 012 | 0C | ♀ | 044 | 2C | , | 076 | 4C | L | 108 | 6C | l |
| 013 | 0D | ♪ | 045 | 2D | - | 077 | 4D | M | 109 | 6D | m |
| 014 | 0E | ♫ | 046 | 2E | . | 078 | 4E | N | 110 | 6E | n |
| 015 | 0F | ☼ | 047 | 2F | / | 079 | 4F | O | 111 | 6F | o |
| 016 | 10 | ► | 048 | 30 | 0 | 080 | 50 | P | 112 | 70 | p |
| 017 | 11 | ◄ | 049 | 31 | 1 | 081 | 51 | Q | 113 | 71 | q |
| 018 | 12 | ↕ | 050 | 32 | 2 | 082 | 52 | R | 114 | 72 | r |
| 019 | 13 | ‼ | 051 | 33 | 3 | 083 | 53 | S | 115 | 73 | s |
| 020 | 14 | ¶ | 052 | 34 | 4 | 084 | 54 | T | 116 | 74 | t |
| 021 | 15 | § | 053 | 35 | 5 | 085 | 55 | U | 117 | 75 | u |
| 022 | 16 | ▬ | 054 | 36 | 6 | 086 | 56 | V | 118 | 76 | v |
| 023 | 17 | ↨ | 055 | 37 | 7 | 087 | 57 | W | 119 | 77 | w |
| 024 | 18 | ↑ | 056 | 38 | 8 | 088 | 58 | X | 120 | 78 | x |
| 025 | 19 | ↓ | 057 | 39 | 9 | 089 | 59 | Y | 121 | 79 | y |
| 026 | 1A | → | 058 | 3A | : | 090 | 5A | Z | 122 | 7A | z |
| 027 | 1B | ← | 059 | 3B | ; | 091 | 5B | [ | 123 | 7B | { |
| 028 | 1C | ∟ | 060 | 3C | < | 092 | 5C | \ | 124 | 7C | \| |
| 029 | 1D | ↔ | 061 | 3D | = | 093 | 5D | ] | 125 | 7D | } |
| 030 | 1E | ▲ | 062 | 3E | > | 094 | 5E | ^ | 126 | 7E | ~ |
| 031 | 1F | ▼ | 063 | 3F | ? | 095 | 5F | _ | 127 | 7F | ⌂ |

在輸入、儲存和運算時，char 都是使用整數(ASCII值)格式，只有在輸出(顯示或列印) 的時候會依照該 ASCII 值所對應的文字套用字型後輸出

**'A'+1 = ?**

**(char) (int)(???)**

**'A'-1 = ?**

**(char) (int)(???)**

**'A'+'1' = ?**

**(char) (char)(???)**

字元 (char) 是一種整數型態

【範例】 char.cpp

# 【範例】大小寫轉換

- 試寫一程式輸入一大寫英文字元，顯示相對應的小寫英文字元
  - 提示：

```
char input = ?;
char output = input - 'A' + 'a';
```

| 範例輸入一 ： | 範例輸入二 ： | 範例輸入三 ： |
|---|---|---|
| A | B | Z |
| 範例輸出一 ： | 範例輸出二 ： | 範例輸出三 ： |
| a | b | z |

【範例】 tolower.cpp

# 【補充】關於字元

- 就前面的解釋我們了解字元的處理方式是利用建表編號的方法來達成。但在 **C/C++** 語言標準中並沒有強制規定字元所使用的編號表格 **(**只規定要能表示其中 **95** 個字元**)**，像是一般電腦用的是 **ASCII** 或是 **ISO/IEC 646** 的編碼標準都是可能的。
  - 一個字元 (**char**) 原則上由一個位元組構成，但是一個位元組並不一定是有 8 位元 (某些古老或特別的電腦結構)
- 而也因為未嚴格規定編碼表格，**char** 資料型態不一定等價於 **signed char** 或 **unsigned char**，此為《實作相依行為》。

# 使用型態的選擇

- 那麼我們到底應該選擇怎樣的資料型態呢？

| 用途 | 資料類型 | 格式符 |
|---|---|---|
| 整數 (一般情況) | **int** | `%d` |
| 浮點數 (有小數或位數過大) | **double** | `%f` `%lf`<br>`printf` `scanf` |
| 字元 | **char** | `%c` |

  - 對於初學者只要熟悉使用這些資料型態就夠了
- **其他型態的用途？**
  - 最佳化記憶體的使用或效率

# 算術運算子: +-*/%

- 算術運算子運算的結果與運算元的值跟型態有關
  - 運算子是有優先順序的 (***/%** 優先於 **+-** )
  - 算術運算子優先順序相同時，在左邊的先
- 每次執行一個運算子時就會產生一個中間結果，我們稱為『暫時變數』，我們要了解這暫時變數的『值』與『型態』：

```
A = 3 * ( 2 + 1 ) + 7 / 2 + 9 * 3.;
A = 3 * 3         + 7 / 2 + 9 * 3.;
A = 9             + 7 / 2 + 9 * 3.;
A = 9             + 3     + 9 * 3.;
A = 9             + 3     + 27.   ;
A = 12                    + 27.   ;
A = 39.                             ;
```

# 【範例】賦值運算子：=

- = 為賦值運算子
  - 賦值運算子會將右方的值給**左方的**變數
    - 賦值運算子的左方一定要放置某個變數。
  - 賦值運算子的運算結果就是左方變數最後的值跟型態
- 運算優先順序
  - 賦值運算子 (=) 的運算優先順序是全部裡面最低的而且運算順序是由右至左 (特別！)

```
int A,C;
double B, D;
A = 3;
3 = A;
A = C = 3;
A = B = C = D = 3 + 7 / 2.;
```

這結果會是什麼？

【範例】 assign.cpp

# 是非真假

| 運算結果 | 意 | | |
|---|---|---|---|
| 不是 0 | 真的 | 正確 | 成立 |
| 0 | 假的 | 錯誤 | 不成立 |

- **在一般 C 語言標準 (C89/C90) 中，我們使用 int 型態來儲存是非真假對錯：**
  - 0 表示假的、錯誤和不成立的意思
  - 1 表示真的、正確和成立的意思
  - 2 表示真的、正確和成立的意思
  - -1 表示真的、正確和成立的意思

**【補充】在 C++ 中，也可以用 bool 資料類型來表示**

# 關係與等號運算子

- 在 C 語言標準中，關係與等號運算子的運算結果有不是 0 (成立) 和 0 (不成立) 兩種可能：

| 運算意義 | 運算符號 |
|---|---|
| 大於 | > |
| 小於 | < |
| 大於等於 (不小於) | >= |
| 小於等於 (不大於) | <= |
| 等於 | == |
| 不等於 | != |

```
4 > 3
4 < 3
4 == 3
4 != 3
4 > 3 > 2
```

【補充】在 C++ 中，運算結果是 true 與 false 兩種可能

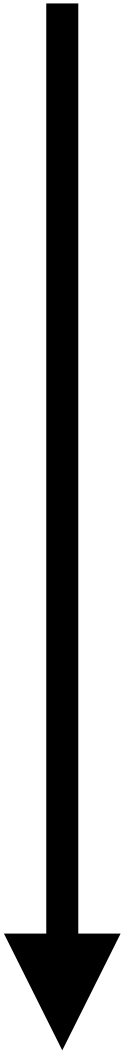
# 邏輯運算子

- 在 C 語言標準中，邏輯運算子的運算結果有不是 0 (成立) 與是 0 (不成立) 兩種可能：

| 運算意義 | 運算符號 | iso646.h C++ |
|---|---|---|
| 而且 (and) | && | and |
| 或者 (or) | \|\| | or |
| 非 (not) | ! | not |

```
3 > 2 && 1 > 2
    1 || 0
4 > 3 && 3 > 2
4 > 3 && 4 < 3
4 > 3 || 4 < 3
  !(3 > 2)
    !3
```

# 運算子優先順序表




| 運算符號 | 平手時運算順序 |
|---|---|
| ( ) | 由左至右 |
| ! | 由左至右 |
| * / % | 由左至右 |
| + - | 由左至右 |
| < > <= >= | 由左至右 |
| == != | 由左至右 |
| = | 由右至左 |
| && | 由左至右 |
| \|\| | 由左至右 |

# 【補充】邏輯運算子的特殊性

- **||** 與 **&&** 運算子都保證左邊運算元被算出後，才會開始算右邊運算元的值。
  - 其他運算子不是喔！之前我們只保證運算子的運算順序
- **||** 左邊運算元算出為非 **0** 時，就不會去算右邊運算元的值。反之當 **&&** 左邊運算元算出為 **0** 時，就不會去算右邊運算元的值。
  - 為什麼可以這樣？

# if 關鍵字

- if(表示式) { . . . }
  - 如果 表示式 為真就 ...
- 什麼是真或假？
  - 非0 或 0、成立或不成立

```
if (80 >= 60) {
   printf("PASSED !\n");
}

if (80 < 60) {
   printf("FAILED !\n");
}
```

```
int grade = 80;
if (grade >= 60) {
   printf("PASSED !\n");
}

if (grade < 60) {
   printf("FAILED !\n");
}
```

# 【範例】比較兩數大小

- 試寫一程式讓使用者輸入兩個數字後顯示其中比較大的給使用者看：

```
請輸入第一個整數 : 3
請輸入第二個整數 : 4
比較大的整數是   : 4
```

- 提示：(程式片段)

```
int max;
if (num1 >= num2) {
  max = num1;
}
if (num1 < num2) {
  max = num2;
}
printf("比較大的整數是 %d\n", max);
```

【範例】 cmp.cpp

# 【練習】簡易版猜數字

- 試寫一個程式，在程式內部預設一個整數作為猜數字遊戲的答案。當使用者執行程式後，需要輸入一個整數，如果該整數與程式預設的答案不同，請顯示是比較大或者比較小；如果該整數與程式預設的答案相同，請恭喜使用者：

```
請輸入你的猜測 ： 3
太小了！

請輸入你的猜測 ： 5
太大了！

請輸入你的猜測 ： 4
答對了！
```

  - 【思考】要如何讓使用者可以一直猜到答案正確？

# 【補充】逐位元運算子

| 運算意義 | 運算符號 |
| --- | --- |
| 逐位元 AND | & |
| 逐位元 OR | \| |
| 逐位元 XOR | ^ |
| 逐位元 NOT | ~ |
| 逐位元左移 | << |
| 逐位元右移 | >> |

# 習題 [1]

- **[E0201]** 試寫一程式，印出下面這個變數值：
  - **double** x = 3000000000000000.5;
- **[E0202]*** 試寫一程式，印出下面式子的計算結果：
  - 3000000000000000.5+0.05
- **[E0203]** 試寫一程式，輸入英哩換算後印出公里 **(四捨五入至小數點後一位) [公里 = 英哩 * 1.6]**
- **[E0204]** 試寫一程式印出 **129263*54628** 的結果
  - 與 [E0110] 相同
- **[E0205]*** 試寫一程式算出 **1292635428** 三次方的值 **(2159872744519190546127922752)**

# 習題 [2]

- **[E0206]** 試寫一程式，輸入一有號整數，顯示該整數是正整數 **(>=0)** 或負整數 **(< 0)**
- **[E0207]** 試寫一程式，輸入一字元，顯示該字元是數子 **(0-9)**、英文字元 **(a-zA-Z)** 或其他符號
  - 字元 (**char**) 請在 **scanf** 內用 **%c** 讀入，在 **printf** 內用 **%c** 印出
- **[E0208]** 試寫一程式，輸入一個英文小寫字元，將字元轉換為大寫印出
- **[E0209]** 試寫一程式，輸入一個英文字元，將字元的大寫印出 **(**不限制輸入的字元為大寫或小寫**)**

# 習題 [3]

- **[E0210]** 試寫一程式，輸入兩個整數，將兩個整數由小到大印出。

  範例輸入一： 3 6　　範例輸出一： 3 6

  範例輸入二： 6 3　　範例輸出二： 3 6

- **[E0211]** 試寫一程式，讓使用者輸入一八位整數然後將數字直排顯示。

```
請輸入一個八位數整數： 38603456
3
8
6
0
3
4
5
6
```

# 習題 [4]

- **[E0213]** 試寫一程式，讓使用者輸入身分證字號的前九碼後顯示該身分證字號的第十碼 (驗證碼) `%c%d`
  - 我們的檢查碼計算方式：(A: 10, B:11, C:12, ..., Z:36)

| A | | I | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| I | 0 | I | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| x1 | x9 | x8 | x7 | x6 | x5 | x4 | x3 | x2 | x1 |

  - 總和 = 1x1 + 0x9 + 1x8 + 2x7 + 3x6 + 4x5 + 5x4 + 6x3 + 7x2 + 8x1 = **121**
  - **121**%10 = **1**, (10-**1**)%10 = **9**