

# for 迴圈練習與四則運算

實習目標：

丁培毅

1. 瞭解輸出資料格式的變化
2. 練習 for 迴圈的語法, 遵循語法規定, 掌握語法允許的變化, 完全瞭解各部份執行的順序
3. 體會迴圈如何有效運用電腦運算能力
4. 設計程序化程式的基本方法 – 模擬執行
5. 運用 for 迴圈設計程式需要完全掌握迴圈變數的意義
6. 資料的強制型態轉換
7. (廣義)運算式的組成: 運算子的優先順序與結合律
8. 瞭解變數資料型態選擇運算子的方法

# for 迴圈練習與四則運算

請撰寫一個程式

- 要求使用者由鍵盤鍵入一正整數  $n$
- 計算並且在螢幕上印出下表的運算結果

請輸入正整數：8

$$1 / 2 = 0.500000$$

$$2 / 3 = 0.666667$$

$$3 / 4 = 0.750000$$

$$4 / 5 = 0.800000$$

$$5 / 6 = 0.833333$$

$$6 / 7 = 0.857143$$

$$7 / 8 = 0.875000$$

$$8 / 9 = 0.888889$$

分析: 想像你自己是執行這個程式的電腦, 你需要

1. 在螢幕上列印提示文字 “請輸入正整數 :”

```
printf("Hello world\n");
```

2. 當使用者在鍵盤上打入整數 (假設使用者是聽話的), 你需要讀進來, 放到一個整數變數  $n$  裡

```
int n;  
scanf("%d", &n);
```

3. 接下來需要根據變數  $n$  裡存放的數值, 計算及列印  $n$  次 (你需要由前面的執行範例裡看出來迴圈需要執行幾次, 例如在範例裡你看到每一列第一個數字是 1, 2, ..., 8, 所以可以確定是  $n$  次)

```
for (int i=1; i<=n; i++) {  
    ....  
}
```

請注意設計時通常會讓迴圈的控制變數  $i$  對照到重複執行的事項裡面某一個有意義的資料, 例如在這裡變數  $i$  的數值就是印出來每一列的第一個數字, 如此在構思迴圈的內容時可以有一個參考數值

4. 第  $i$  列須要列印在螢幕上的是：  
例如  $i = 4$  時  $4 / 5 = 0.800000$   
可以寫成

$$\begin{array}{l} \text{變數 } i \\ \text{裡存放的} \\ \text{整數資料} \end{array} / \begin{array}{l} \text{變數 } i \\ \text{裡存放的} \\ \text{整數資料} \\ + 1 \end{array} = \begin{array}{l} \text{(變數 } i \text{ 裡存放的整數資料轉換成} \\ \text{浮點數格式) 除以 ((變數 } i \text{ 裡存放的} \\ \text{整數資料} + 1) \text{ 轉換成浮點數格式)} \end{array}$$

用 C 語言表示為  $((\text{double}) i) / ((\text{double})(i+1))$

5. 列印的部份可以分開來寫成

```
printf("%d", i);  
printf(" / ");  
printf("%d", i+1);  
printf(" = ");  
printf("%f", ((double) i) / ((double)(i+1)));  
printf("\n");
```

或是合併在一個敘述裡

```
printf("%d / %d = %f\n", ((double) i) / ((double)(i+1)));
```

6. 有很多方法可以得到和  $((\text{double}) i) / ((\text{double})(i+1))$  一樣的結果，你可以自己嘗試一下，例如：

```
((double) i) / (i+1)
(double) i / (i+1)
i * 1.0 / (i+1)
i / (i + 1.0)
i / ((double) (i+1))
i / (double) (i+1)
i / ((double) i + 1)
i / (i + (double) 1)
```

所有上面的寫法中的 **double** 如果換成 **float** 也可以得到一樣的結果，這有點討厭吧！怎麼沒有標準答案呢？就像所有的語言一樣，要表示相同的意思常常有很多種方法，你需要知道關鍵的規則，才能夠自己變換。

另外你也可以嘗試一下

```
i / i + 1
i / (i+1)
```

這兩種寫法的答案你也需要能夠解釋，這樣才算了解運算式的規則

7. C 程式中 +, -, \*, / 運算式的基本規則：
- (a) 先乘除後加減, 小括號裡面先做
  - (b) 連續相同運算符號由左至右執行
- 其它運算符號請參考 [operator precedence/association 表格](#)
8. 整數和浮點數的表示方法不一樣，整數是用二的補數表示，浮點數是用 IEEE 754 的格式表示，可以用 (double) 或是 (int) 這樣的 強制型態轉換 語法互相轉換：例如
- (double) 3 和 3.0 會得到相同的結果，
  - (int) 4.6 和 4 會得到相同的結果；
- 很多時候編譯器會根據前後文幫你做資料型態的轉換：例如
- int x; x = 3.6; 編譯器會幫你改爲 x = (int) 3.6;
  - double y; y = 5; 編譯器會幫你改爲 y = (double) 5;
  - x = x + 3.5; 編譯器會幫你改爲 x = (int) ((double) x + 3.5);
- 編譯器遵循的規則是什麼？
9. = 的規則比較簡單，如果等號左手邊的變數型態是 double，右手邊運算的結果是 int，編譯器幫你加上 (double) 的轉換
- 例如： int x = 3; double y; y = 10 - 2 \* x;
- ⇒ y = (double) (10 - 2 \* x);

9. 如果等號左手邊的變數型態是 `int`，右手邊運算的結果是 `double`，編譯器幫你加上 `(int)` 的轉換，但是也會警告你資料的精確度會有損失 (會失去小數點右邊的資料)

例如：`int x; double y = 7.3; x = y / 2.0;`

$\Rightarrow y = (\text{int}) (y / 2.0);$

10. 四則運算 `+`, `-`, `*`, `/` 個別都有兩個版本：整數和浮點數的版本，你可以想像因為資料的格式都不一樣了，計算方法當然不一樣，所以需要有兩個版本，當然更進一步也許你需要知道浮點數是類似科學記號的表示方法，浮點數相加減時需要先把小數點的位置(指數部份)調整到相同，才能夠加減，相對地整數的加減法就不需要考慮這個

既然有兩個版本，當你在程式裡寫 `x / y` 時到底是用哪一個版本？這是授權由編譯器根據前後文來決定的，規則是

- a. 如果 `x` 或是 `y` 有一個是浮點數，`x / y` 的除法就是浮點數的除法，計算出來的結果也是浮點數的格式，如果參與浮點數除法的資料是整數格式的話，編譯器就自動加上 `(double)` 轉為浮點數，另外浮點數除法要求資料都是倍精準浮點數，7

10. 如果資料是 `float` 型態也會加上(`double`) 強制轉換為倍精  
準浮點數

b. 否則如果兩個參與運算的資料都是 `int/char/short/long`  
型態，`x / y` 的除法就是整數的除法，計算出來的結果也  
是整數的格式

所以前面提及的 `i / i + 1`，根據上面的規則會等效於  
`((i / i) + 1)` 的計算方法，同時 `'/'` 和 `'+'` 都是整數運算，所以  
只要 `i` 裡面存放的資料不是 `0`，結果就會是 `2`；

另外 `i / (i + 1)`，根據上面的規則會等效於 `(i / (i + 1))` 的計  
算方法，同時 `'/'` 和 `'+'` 都是整數運算，只要 `i` 裡面存放的資  
料不是 `-1`，結果會都是 `0`

11. 前面看到的寫法 `i / (i + 1.0)`，根據上面的規則會等效於  
`(i / (i + 1.0))` 的計算方法，其中 `'+'` 是浮點數的加法，所以  
編譯器會幫你加上型態轉換 `(i / (((double) i) + 1.0))`，加法  
的結果是浮點數，因此其中 `'/'` 也就會是浮點數的除法，編  
譯器會幫你加上型態轉換成為 `((double) i) / (((double) i) +  
1.0))`，`i` 存放的數值是 `4` 時，會先轉換為 `4.0`，然後加 `1.0`  
，再計算 `4.0 / 5.0 = 0.8`，請嘗試解釋其它的寫法