陣列 (Arrays) 使用方法簡介

我們在後面的課程會詳細介紹陣列。但是其實陣列的基本概念也不複雜，而且我們也用過字元陣列，所以我們就先簡介陣列的使用方法，這樣接下來的程式也有更多寫法可以運用。陣列就是一連串相同型別的資料存放在連續的空間中，譬如 15 個 **int** 型別的資料，或是十個 **char** 字元。整個陣列的內容只需要用一個名字來統稱，而其中每個元素可以藉由索引 index 方式取出。宣告陣列的方法如下

```
float nums[20];
```

這樣就表示 **nums** 是一個陣列，包含 20 個元素，每個元素可用來記錄一個型別為 **float** 的數值。陣列的第一個元素叫做 **nums[0]**，第二個元素是 **nums[1]**，以此類推，最後一個元素是 **nums[19]**。所以 C 的陣列編號方式是從 **0** 開始編號而不是從 **1** 開始。這一點要特別注意，因為這也是初學者常忘記的規則，用錯了會產生 bug。每個元素可以當作一個變數來使用，譬如

```
nums[3] = 3.4;
nums[9] = 18.5;
```

或是

```
scanf("%f", &nums[4]); /* 讀入一個小數存放在第五個元素裡 */
```

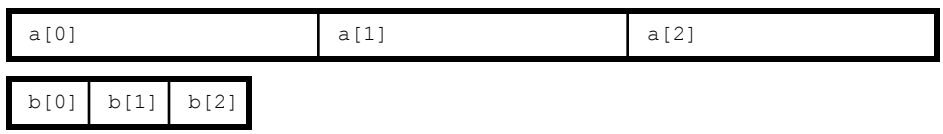
由於 C 並不會去檢查我們給的 index 是否超出當初宣告的陣列大小範圍，所以可能會寫出下面有 bug 的程式而沒有察覺，compiler 也不會跟我們說程式有 error

```
nums[20] = 2.5; /* A BUG */
nums[30] = 3.6; /* A BUG */
```

程式 compile 之後，執行到像是上面那兩行的時候，程式非常可能就會當掉，因為它試圖去讀取錯誤的記憶體位置中的東西。

我們還可以宣告其他類型的陣列，譬如

```
int a[3];
long c[1000];
char b[3];
```



這裡順便複習一下，字元陣列和字串的差別只在最後是否有 **'\0'** 結束符號，如果要當作字串來使用，字元陣列要加入 **'\0'** 當作結束字元。不同型別的陣列佔用的記憶體空間也不同，例如 **int** 陣列每個元素佔用四個 bytes，而 **char** 陣列每個元素只佔一個 byte。

搭配陣列來使用迴圈
[範例]

```
#include <stdio.h>
int main(void)
{
   int i;
   float hours[7], average;
   printf("Enter the hours of sleep per night last week\n");
   for(i = 0; i < 7; i++) {
      scanf("%f", &hours[i]);
   }
   printf("The numbers you entered are as follows:\n");
   for(i = 0; i < 7; i++) {
      printf("  %4.1f", hours[i]);
   }
   printf("\n");
   for(i = 0, average = 0; i < 7; i++) {
      average += hours[i];
   }
   average /= 7;
   printf("Your average hours of sleep per night were %.1f hours.\n", average);
   return 0;
}
```

輸出：

```
Enter the hours of sleep per night last week
9 10 4 12 13 14 3.5
The numbers you entered are as follows:
   9.0  10.0   4.0  12.0  13.0  14.0   3.5
Your average hours of sleep per night were 9.4 hours.
```

前面的範例先用迴圈讓使用者輸入七個數字，分別存在陣列裡，然後再用迴圈一一把陣列裡的元素再顯示出來，然後再用迴圈把元素的值累加起來，最後計算出平均值。有幾個要注意的地方，第一是迴圈裡的用來當陣列 index 的變數要從 **0** 開始；第二是用來累加總和並計算平均的變數 **average** 要先設定初值等於 **0**；第三是 **scanf()** 裡陣列元素的寫法，當我們要把值存入某個陣列元素時，別忘了加 **&** 符號，例如 **&hours[1]**，只要記得每個陣列元素其實就相當於我們平常在使用的變數，所以要遵守的原則都相同。

[範例]

```
#include <stdio.h>
double power(double n, int p);

int main(void)
{
   double x, xpow;
   int exp;

   printf("Enter a number and the positive integer power\n");
   printf(" to which the number will be raised. Enter q to quit.\n");
   while (scanf("%lf%d", &x, &exp) == 2) {
      xpow = power(x, exp);
      printf("%.3f to the power %d is %.5f\n", x, exp, xpow);
      printf("Enter next pair of numbers or q to quit.\n");
   }
   printf("\nBye!\n");
   return 0;
}

double power(double n, int p)
{
   double pow = 1;
   int i;

   for (i = 1; i <= p; i++) {
      pow *= n;
   }

   return pow;
}
```

這個範例主要是要介紹如何自己寫一個 function 來計算某個數的整數次方。這個自定的 function 取名為 **power()**，在主程式的前面要先宣告 **power()**，把它需要的參數型別，以及回傳值的型別都指定好。所以由宣告可以看出 **power()** 要傳入兩個參數，分別是型別為 **double** 的底數 **n** 以及型別為 **int** 的指數 **p**。至於 function 主體 (definition) 的部份則在 **main()** 後面，我們用了迴圈來達到計算次方的效果。首先 **pow** 的值被設成 **1**，然後迴圈要做 **p** 次，每次都會把 **pow** 乘上 **n** 再存回 **pow** 自己。所以做了 **p** 次之後 **pow** 的值就等於 **n** 的 **p** 次方。最後把 **pow** 的值傳回去，用的語法是 **return pow;**。有了自己訂的 **power()**，就可以在主程式裡重複呼叫。這樣的模組化程式設計方式是我們學習 C 語言要多加練習的重點之一。我們把較複雜而會重複使用的程式碼寫成 function，不僅可以讓程式看起來更簡潔，而且程式會比較有結構、比較容易理解、也比較容易修改。當程式越寫越龐大的時候，這項優勢就會更加突顯。

[練習] 寫一個 function 叫做 **rectangle()**，它需要傳入三個參數，分別是兩個 **int**，各代表寬和高，以及一個 **char**，function 要依照寬和高畫出長方形，長方形用第三個參數所指定的字元來填滿。最後 function 還會把長方形面積當作回傳值傳回去。另外，主程式的部份則是在 **main()** 裡面呼叫兩次 **rectangle()**， 分別畫出一個 **7** 乘 **5** 用 **'#'** 填滿的長方形，以及一個 **12** 乘 **9** 用 **'@'** 填滿的長方形，並且把面積印出來。

[練習]** 讓使用者輸入一個大寫英文字母，譬如 **'E'**，然後輸出像右圖一樣排列的形狀。
提示： 總共需要兩層迴圈。外層用一個迴圈負責跑每一行，然後裡面包了三個迴圈，第一個內層迴圈負責印出適當數量的空白字元，第二個內層迴圈負責以遞增順序印出字母，然後第三個內層迴圈以遞減順序印出字母。

```
    A
   ABA
  ABCBA
 ABCDCBA
ABCDEDCBA
```

[*練習] 讓使用者輸入一個字串 (譬如一個英文單字)，把這個字存入 **char** 陣列中，然後把整個字以相反順序印出來。譬如輸入 **dog** ，則輸出的結果會是 **god**。
提示：用 **strlen()** 取得字串長度 (要記得 **#include <string.h>**)。另外在取出陣列元素時，index 編號要小心。

[補充] 用多重迴圈輸出九九乘法表上半部

```
#include <stdio.h>
int main(void)
{
   int i, j;

   for (i = 1; i <= 9; i++) {
      for (j = 1; j < i; j++) {
         printf("   ");
      }
      for ( ; j <= 9; j++) {
         printf("%3d", i*j);
      }
      printf("\n");
   }

   return 0;
}
```

輸出：

```
  1  2  3  4  5  6  7  8  9
     4  6  8 10 12 14 16 18
        9 12 15 18 21 24 27
          16 20 24 28 32 36
             25 30 35 40 45
                36 42 48 54
                   49 56 63
                      64 72
                         81
```

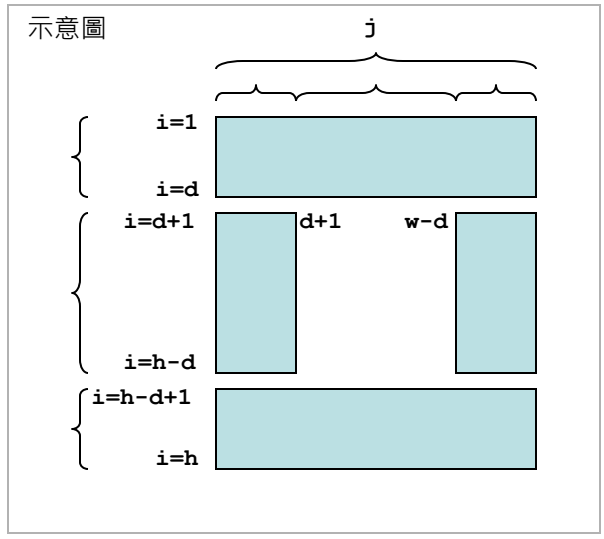
左下角有一個隱藏的空白三角形，是用內層 **j** 迴圈的的第一部份畫出來。

[補充] 用多重迴圈畫出空心的長方形

```
#include <stdio.h>
int main(void)
{
   int w, h, d;
   int i, j;

   scanf("%d%d%d", &w, &h, &d);

   for (i = 1; i <= d; i++) {
        for (j = 1; j <= w; j++) {
            printf("*");
        }
        printf("\n");
    }
   for ( ; i <= h-d; i++) {
        for (j = 1; j <= d; j++) {
            printf("*");
        }
        for ( ; j <= w-d; j++) {
            printf(" ");
        }
        for ( ; j <= w; j++) {
            printf("*");
        }
        printf("\n");
    }
   for ( ; i <= h; i++) {
        for (j = 1; j <= w; j++) {
            printf("*");
        }
        printf("\n");
    }

   return 0;
}
```

輸出：

```
15 8 2
***************
***************
**             **
**             **
**             **
**             **
***************
***************
```



關鍵是要能正確設定"負責控制迴圈次數的變數"的**初始值**和**結束值**。

[補充] 字元陣列的讀寫

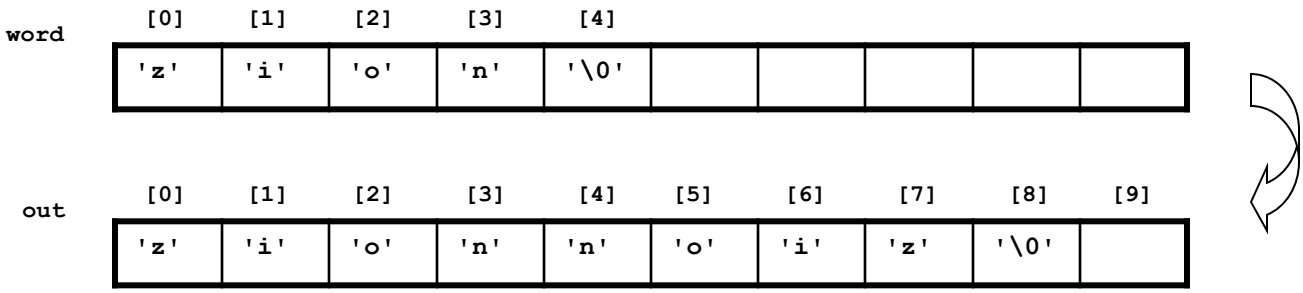
```
#include <stdio.h>
#include <string.h>
int main(void)
{
    int i;
    char word[50];                  /* word 字元陣列用來記錄使用者輸入的字串 */
    char out[100];                  /* out 是用來暫存輸出的結果 */
    int len;

    scanf("%s", word);              /* 讓使用者輸入一個字串 */
    len = strlen(word);             /* 計算使用者輸入的字串長度 */

    for (i = 0; i < len; i++) {     /* 用迴圈把整個 word 從頭到尾看一遍 */
        out[i] = word[i];           /* 把 word 逐字複製到 out 對應的位置 */
        out[2*len-i-1] = word[i];   /* out 另外一邊對稱的位置也要填上同樣的字元 */
    }
    out[2*len] = '\0';              /* 在 out 的後面填上'\0'，讓 out 變成字串 */
    printf("%s\n", out);            /* 這樣就可以用 printf() 配合 %s 格式直接輸出整個字串*/


    for (i = 0; i < 2*len; i++) {    /* 先把 out 用空白字元填滿 (注意:'\0'還在，不會被蓋掉)*/
        out[i] = ' ';
    }

    for (i = 1; i < len; i++) {     /* 接下來要顯示垂直的部份 */
        out[0] = word[i];           /* 把 out 的第一個字元填上 word[i] */
        out[2*len-1] = word[i];     /* 把 out 最後面的字元填上 word[i] */
        printf("%s\n", out);        /* 把 out 輸出 */
    }

    for (i = len-1; i > 0; i--) {   /* 接下來要顯示垂直的另一半對稱的部份，迴圈要倒著跑 */
        out[0] = word[i];           /* 把 out 的第一個字元填上 word[i] */
        out[2*len-1] = word[i];     /* 把 out 最後面的字元填上 word[i] */
        printf("%s\n", out);        /* 把 out 輸出 */
    }


    for (i = 0; i < len; i++) {     /* 重複做一次第一部份的迴圈 */
        out[i] = word[i];
        out[2*len-i-1] = word[i];
    }
    printf("%s\n", out);            /* 因為 out 的結尾 '\0' 並沒有被更動，所以不用再填 */

    return 0;
}
```

底下是程式執行的情況：

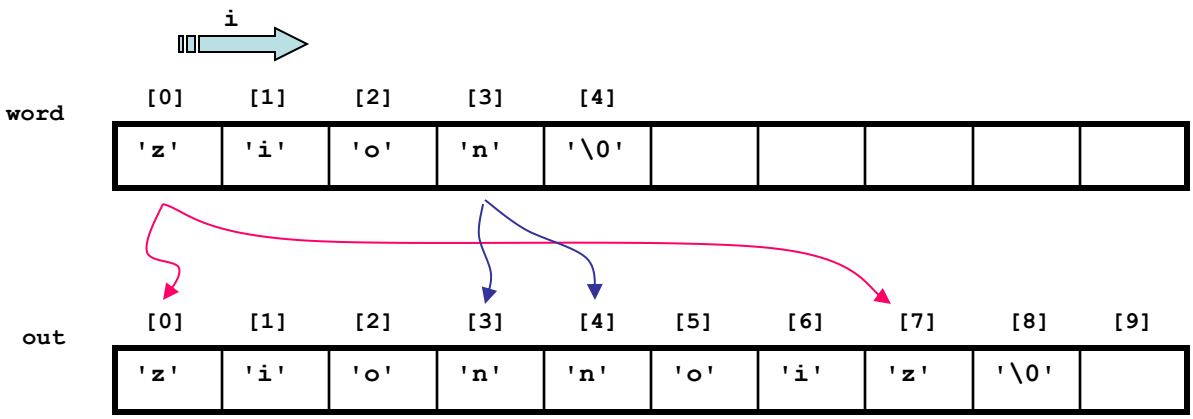
```
zion
zionnoiz
i      i
o      o
n      n
n      n
o      o
i      i
zionnoiz
```

請看下一頁的圖文說明

首先要想辦法產生下面的東西 (注意陣列的索引編號)：

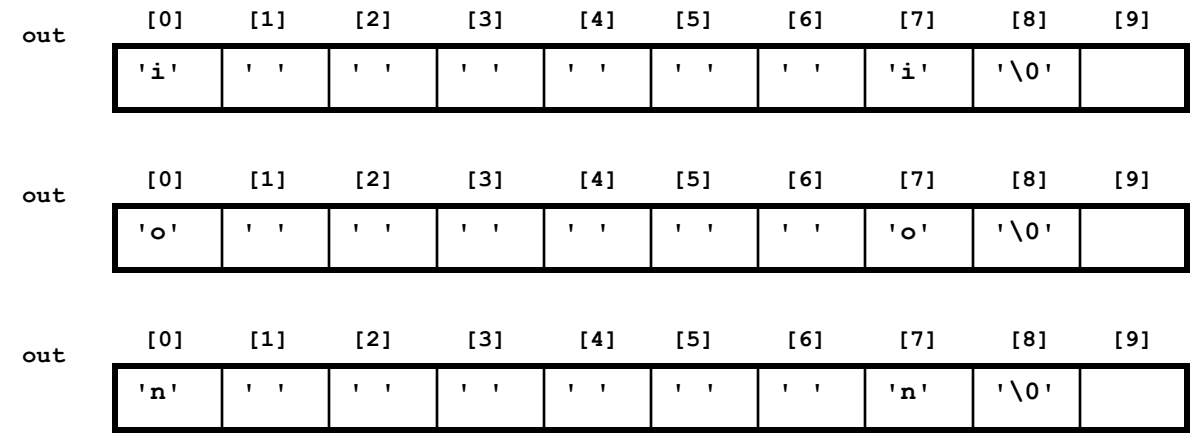


做法是用迴圈把對應位置該放的字元填好：



填好 `out` 之後，就可以用 `printf("%s\n", out);` 把 `out` 的內容整個顯示出來。

接下來垂直的部份，只需要依序產生下面的陣列。先把 `out` 清空，然後在頭尾填上該放的字元 。(在這個例子該放的字元分別是要從 `'i'`， `'o'`， `'n'`，也就是 `word[1]`， `word[2]`， `word[3]`。)



完成前面兩個部份之後，剩下的部分只要把前兩個動作反過來再做一次就可以。

整個程式的關鍵在於如何設定正確的陣列索引 (index)，在陣列中適當的位置放入該放的字元。要特別注意陣列的索引要從 0 開始。字串的最後一個字元要加上 `'\0'`。