Chapter 7

簡易字串

傳統字串

- 固定長度的字元陣列
- 字串末尾字元為空字元('\0')
- 字串長度為空字元之前的所有字元個數
- 字元陣列長度一定要多於字串長度加一

C-style string

傳統字串的設定

- 陣列長度為 20，字串長度為 3，用了 4 個字元

```
char a[20] = { 'c' , 'a' , 't' , '\0' } ;
```

- 陣列長度為 20，字串長度為 9，用了 10 個字元

```
char b[20] = "123456789" ;
```

- 陣列長度為 4，字串長度為 3，字元總數為 4

```
char c[]   = "dog" ;
```

- 字串長度為 4，但字元總數為 5，d 為常字串

```
char *d    = "fish" ;
```

```
// a 為字串，第四個字元之後都是空字元
char a[20] = { 'c' , 'a' , 't' } ;
// b 不是字串，為 3 個字元的字元陣列
char b[3]  = { 'c' , 'a' , 't' } ;
```

常字串與字元陣列（一）

- 常字串內的資料不能在程式執行中被更改

```
char *foo  = "fish" ;
foo[0]     = 'F'    ;      // 錯誤
```

❖ 可在 **char** 之前使用 **const** 加以突顯常字串的常數特性

```
const char *foo = "fish" ;
```

- 若要更改字串，則須以字元陣列方式儲存字串

```
char bar[] = "fish" ;
bar[0]     = 'F'    ;      // 正確
```

常字串與字元陣列（二）

- 指向常字串的指標，在執行中可隨時指向其他位址

```
char *a = "horse" ;          // 指標 a 指向常字串 horse
char *b = "fish"  ;          // 指標 b 指向常字串 fish
a = b ;          // 指標 a 指向指標 b 所指的位址，即指向 fish
```

❖ 以上的常字串 "horse" 會因此失去連結，永遠無法為程式內的其他式子所使用

- 字元陣列在執行中不可指向其他位址

```
char a[] = "horse" ;      // 字串陣列 a 存有 horse\0
char b[] = "fish"  ;      // 字串陣列 b 存有 fish\0
a = b ;                   // 錯誤
```

string literal

常字串與中文字串

- 常字串可以使用 '\' 來斷行

```
char *foo = "123\
␣␣456\
7890" ;
```

foo 指向 "123␣␣4567890"

- 常字串也可使用分號分開連結

```
char foo[] = "123"
             "456" ;
```

foo 儲存 "123456"

- 中文字串：每個中文字以兩個字元代表。字串長度為中文字數的兩倍

```
// 陣列長度為 19，字串長度 18
char foo[] = "月明星稀，烏鵲南飛" ;
```

常犯的錯誤字串設定

- 常犯錯誤的字串設定

```
char foo[6] ;
foo = ”tiger” ;                  // 錯誤 foo 為字元陣列
foo = ”crocodile” ;              // 錯誤 foo 為字元陣列且長度不足
char a[10] , b[10] , c[20] ;
      …
a = b ;                          // 錯誤 a 為陣列
a = c ;                          // 錯誤 a 為陣列且長度不足
```

- 如果使用字元指標設定，可能會造成字串失去連結

```
char *p ;
p = ”tiger” ;                    // 正確
p = ”cat” ;                      // 正確 但原有的 ”tiger” 就失去連繫
char *r = ”sheep” ;
char *s = ”dog” ;
r = s ;                          // 正確 但原有的 ”sheep” 就失去連繫
```

字元陣列的輸入與輸出（一）

- 字串的輸入

  字串只能存入字元陣列內，且字串長度須比字元陣列長度少一個字元，以供空字元所使用

- 字串的輸出

  若要輸出字串，僅須使用字元陣列名稱或字元指標即可。

  此時顯示的字元為空字元之前的所有字元

```
char a[20] ;
cin >> a ;                  // 輸入最多 19 個字元
cout << a << endl;          // 輸出在空字元之前的所有字元
```

字元陣列的輸入與輸出（二）

- 字串輸出的寬度可以使用 **setw** 設定

```
#include <iomanip>
…
char a[10] ;
cin >> a   ;
cout << setw(15) << a << endl ;
```

- 若字元陣列缺少空字元，則輸出的字元總數通常會超出陣列的字元總數

```
char a[3] = {'c' , 'a' , 't'} ;  // a 不是字串
cout << a << endl ;              // 語法正確但輸出錯誤
```

使用字元指標輸入與輸出

- 字元指標可以用來指向字元陣列，有時也可以用來輸出字串。字元指標並沒有任何記憶空間用來儲存輸入的字元資料

```
char a[20] = "horse" ;
char *p = a ;                  // 指標指向字元陣列 a
cout << p << endl ;            // 列印：horse

cin >> p ;                     // 輸入一長度小於等於 19 的字串
cout << p << endl ;            // 列印輸入的字串
```

```
char *q ;                      // 定義新字元指標
cin >> q ;                     // 錯誤：q 沒有分配任何記憶空間
cout << q << endl ;
```

字元陣列的操作（一）

- 傳統字串可以使用下標改變字元資料
- 將字串的資料分開列印

```
char a[20] = "startrek" ;
for ( int i = 0 ; a[i] != '\0' ; ++i ) cout << a[i] << " " ;
```

- 計算字串長度

```
char foo[50] = "暮春三月，江南草長" ;
int len = 0 ;
while( foo[len] != '\0' ) ++len ;
cout << "字串長度 :" << len << endl ; // 字串長度 :18
```

- 字串長度可以使用於 **string.h** 標頭檔內的 **strlen** 函式求得

```
#include <string.h>
…
cout << "字串長度 :" << strlen("borg") << endl ; // 印出 4
```

字元陣列的操作（二）

- 將字串內的小寫字母改成大寫

```
char p[] = "The Lord of The Rings" ;
for ( int i = 0 ; p[i] != '\0' ; ++i ) {
    if ( p[i] >= 'a' && p[i] <= 'z' )
        p[i] = p[i] - 'a' + 'A' ;
}
cout << p << endl ;     // 輸出：THE LORD OF THE RINGS
```

- 以上若使用

```
char *p = "The Lord of The Rings" ;
```

則程式將會產生執行錯誤，原因為字元指標所指向的字串為常字串

傳統字串合併

- 字串合併

```
char a[] = ”暮春三月” ;
char b[] = ”江南草長” ;
char c[100] ;
```

➢ 錯誤方式

```
c = a + b ;
```

➢ 正確方式

依次複製兩字串的所有非空字元到新字元陣列，最後再補上一個空字元

```
int i , j ;
for ( i = 0 ; a[i] != ’\0’ ; ++i ) c[i] = a[i] ;
for ( j = 0 ; b[j] != ’\0’ ; ++j ) c[i+j] = b[j] ;
c[i+j] = ’\0’ ;
```

C++ 字串

- C++ 字串為一可自動調整字元長度的字元陣列，字串長度沒有限制
- C++ 字串長度就是真正的字元長度
- C++ 字串末尾無空字元
- 須加入 **string** 標頭檔

C++-style string

C++ 字串物件

- 字串物件

```
string a           ; // a 字串沒有任何字元
string b(5 , 'z'); // b 字串有 5 個 'z' 字元
string c = "cat" ; // c 字串有 3 個字元，分別為'c' 'a' 't'
string d("cat")  ; // d 字串同上
string e = 'a'   ; // 錯誤，需改成 e = "a" 字串
string f = d     ; // f 字串與 d 字串相同
```

- 字串物件陣列

```
string a[10]     ; // 10 個 C++ 字串元素所構成的字串陣列
```

C++ 字串物件的輸入與輸出

- 直接使用 **cin** 輸入與 **cout** 輸出
- 簡單回聲 (echo) 程式

```
string in ;
while ( 1 ) {
    cin >> in ;
    cout << in << endl ;
}
```

- 早期的 **C++** 編譯器無法列印 **C++** 字串，此時可以使用 **.c_str()** 讓此 **C++** 字串以傳統字串方式輸出

```
string foo = "tiger" ;
cout << foo.c_str() << endl ;
```

簡單 C++ 字串操作（一）

- 字串指定： =

```
string foo ;
foo = ”孔明” ;
```

- 字串合成： + , +=

```
foo = foo + ”臥龍” ;        // foo 的末尾加上一字串

foo += ”臥龍” ;     // 相當於 foo = foo + ”臥龍”
```

```
foo = ”孔明” + ”諸葛亮” ;  // 錯誤，加法的兩側至少
                           // 要有一個為 C++ 字串
```

- + 與 += 的右側也可以是字元

簡單 C++ 字串操作（二）

- 字串長度：.size() 或 .length()

```
cout << foo.length() << endl ;
```

- 字元讀取與更改：[]

```
string foo = ”tiger” ;
for ( int i = 0 ; i < foo.size() ; ++i ) {
      cout << foo[i] << ’ ’ ;
}
```

- 產生傳統字串：.c_str()

```
string foo = ”cat” ;
cout << ”> 傳統字串：” << foo.c_str() << endl ;
cout << ”> C++ ： ”     << foo          << endl ;
```

字串陣列（一）

■ 字串陣列：陣列的元素為字串

1. 陣列有五個字串，每個字串有四個字元

```
char weekday[5][4] = {"mon","tue","wed","thu","fri"};
```

2. 陣列有五個常字串，每個字串的大小由初始字串決定

```
char * weekday[5]  = {"mon","tue","wed","thu","fri"};
```

3. 同上，但字串的個數由編譯器自己核算

```
char * weekday[] = {"mon","tue","wed","thu","fri"};
```

4. 陣列字串個數由初始陣列的元素決定，每個字串的長度由初始字串決定

```
char ** weekday = {"mon","tue","wed","thu","fri"};
```

5. 陣列有五個 C++ 字串

```
string weekday[5] = {"mon","tue","wed","thu","fri"};
```

字串陣列（二）

- 之前的 2. 3. 4. 等三種字串陣列的字串都是常字串 (string literal)，無法在程式中加以改變字串內容

```
char * weekday[5] = { "mon" , "tue" , "wed" ,
                      "thu" , "fri"} ;

weekday[1][0] = 'T' ;   // 錯誤，無法將 't' 改成 'T'
```

字串陣列（三）

- 修改字串陣列的元素

```
char weekday[5][4] = {"mon","tue","wed","thu","fri"} ;
for ( i = 0 ; i < 5 ; ++i) weekday[i][0] += 'A' - 'a' ;
for ( i = 0 ; i < 5 ; ++i) cout << weekday[i] << ' ' ;
```

- 使用向量陣列儲存字串

```
int no ;
cin >> no ;
vector<string> foo(no) ;    //  向量陣列內有 no 個 C++ 字串

// 將陣列內每個字串元素都複製成一樣
for( i = 0; i < 5; ++i) foo[i] = "明月松間照，清泉石上流" ;
```

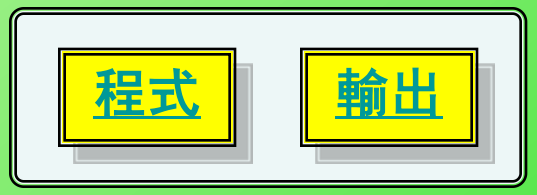
字幕顯示器：跑馬燈

- 將輸出的文字以跑馬燈的方式顯示

使用 `sleep()` 函式讓程式暫時若干時間

使用 `'\r'` 字元讓游標重回同一行的第一個字元處

使用 `flush` 讓要輸出的資料立即顯示在螢幕



字串分解

- 分解長字串內的字

原字串：

```
"Must you be so linear, Jean-Luc"
```

分解成：

```
"Must" , "you" , "be" , "so" ,
"linear" , "Jean" , "Luc"
```

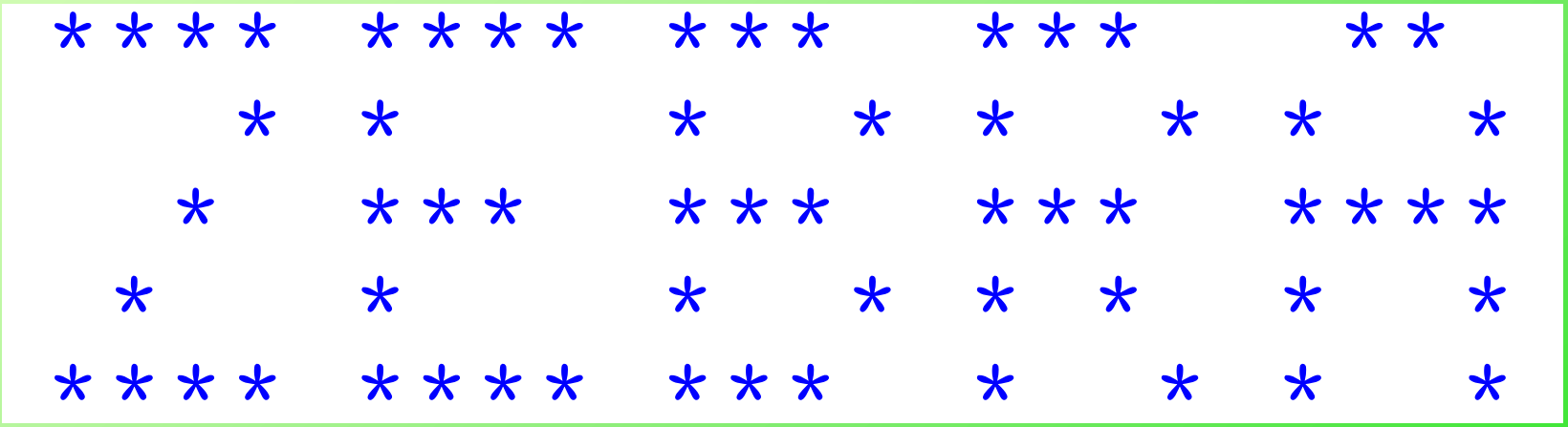
程式 輸出

點矩陣文字字串

- 將輸入字串內的字元以點矩陣方式呈現

輸入：**ZEBRA**

輸出：



程式　輸出