Chapter11：字串處理

字串即為字元的陣列，每個字串以一個 NULL character(空字符)作為結尾；NULL character 是 \0，代表全部為零的一個 byte；也就是說當一個字串有 n 個字元的時候，這個字串總共佔了記憶體 n+1 個 char 的大小。

◆ String literal 就是一個結尾是 \0 (空字符)的陣列

How String Literal Are Stored

➢ 一個字串”abc”：

在記憶體中被儲存為 4 個 character 的陣列，

a	b	c	\0

也就是說”” 是一個 \0 (a single null character)，

\0

其實雙引號(“)本身也是一種運算子，下文會提到。

- Since a string literal is stored as an array, the compiler treats it as a pointer of type char *(pointer to char).
  因為一個 string literal 是被儲存為一個陣列，所以編譯器會把 string literal(陣列)用 pointer to char 當成型別

- Both printf and scanf expect a value of type char * as their first argument(註 1).

- The following call of printf passes the address of "abc" (a pointer to where the letter a is stored in memory):

```
printf("abc");
```

  使用 printf 的時候，若要輸出 abc 這串文字，要先把 "abc" 這個陣列的地址傳給 printf。
  "abc" 本身的型別是 char *，也就是說寫出 "abc" 時，傳出去的會是一個地址(此陣列的地址)，而不是內容。

註 1. Parameter 跟 argument 的差異：粗淺的分辨方法，parameter 指的是在定義函數時給定的變數, 而 argument 指的是使用該函數時實際傳遞給函數的資料。

- We can use a string literal wherever C allows a char * pointer:
  char *p;
  p = "abc";

上述即是把 "abc"(一個陣列→r-value 是地址) 指派給 p(a pointer to an array，指向這個 string literal 的第一個位置)

兩種宣告的差異(char & char *)

1. char ch1, ch2, ch3;

ch1 = “a”;
ch2 = “abc”[1]; //ch2 是 b 字元(the letter b)
ch3 = “abc”; /***WRONG***/ 因為 ch2 是一個字元，而 ”abc” 是一個陣列型別為 char *

2. char *p;
   p = “abc”; //p 是一個指標指向”abc”陣列的第一個元素(a)的位置
3. char *p = "abc"; //p is a pointer to a string literal (an array)
   *p = 'd'; /*** WRONG ***/ 因為此行的 * 是運算子，所以 *p 代表的是”abc”這個 string literal，而陣列不能放在等號的左邊，會造成 undefine behavior

String Literals vs. Character Constants

"a" is represented by a pointer.
'a' is represented by an integer.
➔ printf(“a”); is legal while printf(‘a’); is illegal
printf 裡面要放位置而不是整數

String Variables

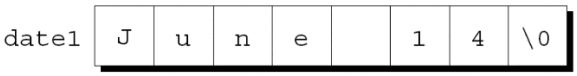
宣告一個字串的時候，要注意其大小是字元個數加一，因為字串最後面是以 \0 結束。

```
#define STR_LEN 80
…
char str[STR_LEN+1];
```

判定 string variables 的大小時，是以結尾的 null character 最判斷，所以一定要注意 null character 的存在。

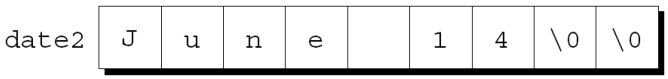
Initializing a String Variable

char date1[8] = "June 14";
注意這裡只有 7 的字元，卻要開長度為 8 的陣列

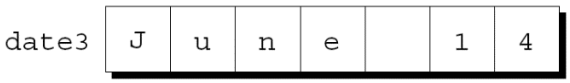


編譯器會自動在陣列的最後加上 \0
看起來像是字串的 initializing，其實是一連串陣列 initializing 的簡寫
char date1[8] = {‘J’, ‘u’, …}

編譯器會把陣列多餘的空間用 \0 填滿
char date2[9] = "June 14";



如果空間不夠，編譯器就不會嘗試去儲存它，並且不會把這個陣列當作是字串
char date3[7] = "June 14";



因為直接輸入字串陣列的長度容易出錯，所以可以讓編譯器自動算出來
char date4[] = "June 14";
如此一來 date4 就會被分到 8 個 character(加上 \0)

Character Arrays vs. Character Pointers

char date[] = "June 14";//declares date to be an array
char *date = "June 14";//declares date to be a pointer
兩者都可以當作字串使用，皆可以 date 當作一個函式的引數(argument)

兩種宣告造成 date 的性質不同:
array version:
在 date 裡的字元可以調整，像是陣列的元素一樣，date 是這個陣列的名字

pointer version:
date 是指向字串的指標，調整 date 並不會改變這個字串，而 date 可以改變為指向其他字串

char *p;//does not allocate space for a string
所以 p 不能當作一個字串使用

char str[STR_LEN+1], *p;
p = str;
此時 p 指向 str 的第一個位置，所以 p 可以當作一個字串使用

char *p;
p[0] = “a”; /*** WRONG ***/
p[1] = “b”; /*** WRONG ***/
p[3] = “c”; /*** WRONG ***/
p[4] = “\0”; /*** WRONG ***/
Since p hasn’t been initialized, we don’t know where it’s pointing, causing undefined behavior

Reading and Writing Strings

Writing Strings Using printf and puts

char str[] = "Are we having fun yet?";
printf("%s\n", str);
output:
Are we having fun yet?
(new line)
printf writes the characters in a string one by one until it encounters a null character.

printf("%.6s\n", str);
output:
Are we
(new line)

we can also use puts to print a string.
puts(str);
puts has only one argument (the string to be printed), puts will always write an additional new-line character.

output:
Are we having fun yet?
(new line)

Reading Strings Using scanf and gets

char str[str_Len];
scanf("%s", str);//不用加&，因為 str 本身是陣列，當成 pointer
使用 scanf，當輸入換行字元、空白鍵、tab 鍵…時，scanf 會跳過，然後讀取直到換行字元、空白鍵、tab 鍵…，所以不能用來讀取有空白字元的字串

若要讀取有弄白字元的字串，可以使用 gets，gets 會一直讀取直到輸入換行字元。gets 不會把最後的換行字元儲存，而是把換行字元丟棄，換成 null character。
gets(str);

當讀取字串時，scanf 和 gets 無法偵測何時會超過字串長度，可能導致 undefined behavior，因此我們可以用以下幾種方式避免：

(1) scanf(“%ns”, str);
    //where n is an integer that indicates the maximum number of character to be stored
(2) gets 就無法避免了，所以用 fgets 會是比較好的方法

Using the C Library

1. The strcpy function
prototype:
char *strcpy(char *s1, const char *s2);
//strcpy is a function with two pointer to char as arguments and return a pointer to char

const char *s2 這裡表示被 s2 指到的字串不會被更改

要注意*s1 和*s2 的長度有可能不一樣：
當*s1 比*s2 長的時候複製字串沒問題，但是當*s1 比*s2 小的時候，因為儲存空間不夠所以無法順利執行，這時我們就要利用 malloc 來解決這個問題。
//str2 是要被複製的字串，複製到 str1
str1 = (char *)malloc(strlen(str2) * sizeof(char));
strcpy(str1, str2);

或者我們可以用 strncpy，這是一個比較，但更安全的方法：
strncpy(str1, str2, sizeof(str1));
但是這個方法有個問題，如果 str2 的長度大於或等於 str1 的長度，strncpy 雖然會順利執行，但是 str1 就不會是 \0，所以我們可以再改良程式碼：
strncpy(str1, str2, sizeof(str1)-1);
str1[sizeof(str1)-1] = ‘0’;
如此一來就可以確保 str1 最後是 \0 結尾

可以觀察到 strncpy 是根據 str1 的大小去做複製的調整，所以如果 sizeof(str2)>sizeof(str1)，str1 就只會複製到部分的內容。
所以要確報可以完整複製 str2 的話，使用 malloc 較佳。

2. The strlen function
prototype:
size_t strlen(const char *s);
使用：
int len;
len = strlen(“abc”);/* len is 3 */

String Idioms

Searching for the End of a String

```
size_t strlen(const char *s)
{
    size_t n = 0;
    for (; *s != '\0'; s++)
        n++;
    return n;
}
```

const char *s 使 s 指到的字串不會更被更改
但 s 本身的值(r-value)可以更改

s 指向的東西不是 \0 的話 s 的值(位置)加一，同時當 s 加一，n 也會加一，如此就可以算出*s 有幾個字元，不包括 \0

const char *s 和 char *(const s)的差異：

const char *s 也可以寫成 chat const *s 表示 s 所指到的字串不會被更改，而 s 本身的值(r-value、指到的位置)可以被更改。

char *(const s)代表 s 的值不(r-value)能被更改，也就是說 s 不會指到其他位置。

上述程式碼可以化簡成：

```
size_t strlen(const char *s)
{
    const char *p = s;
    while(*s++)
        ;
    return s – p;
}
```

先把 s 的值(r-value)傳給 p，以便之後 s – p 的計算

當*s 指的是 \0 時表示 False，反之指到不是 \0 時表示 True；所以 while (*s++)是先判斷 s 是否為 \0，若不是就將 r-value 加一

Copying a String

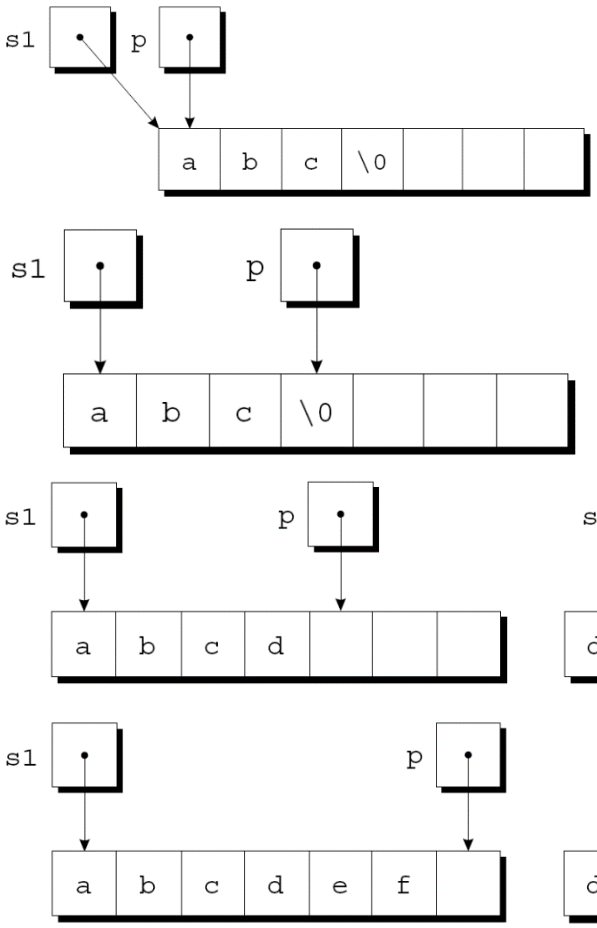
```
char *strcat(char *s1, const char *s2)
{
    char *p = s1;
    while (*p != '\0')
        p++;
    while (*s2 != '\0')
    {
        *p = *s2;
        p++;
        s++;
    }
    *p = '\0';
    return s1;
}
```

先把 p 指到 s1 指到的地方(字串的第一個位置)

把 p 指到的地方移到第一次碰到 \0 的地方(s1 字串的結尾)

依次把 s2 指到的值複製給 p 指到的位置，直到碰到 \0

再字串的結尾加上 \0



上述程式碼可化簡為：

```
char *strcat(char *s1, const *s2)
{
	char *p = s1;
	while (*p)
		p++;
	while (*p++ = *s2++)
		;
	return s1;
}
```

特別解釋 *p++ = *s2++：

先看 *p = *s2，代表把 s2 指到的值複製給 p 指到的位置，接著++運算子使 s2 指到的下一個值複製給 p 指到的下一個位置。那麼何時停止？當指派完*p 是 False 的時候；也就是當 s2 指到的值是 \0 的時候，會先把 \0 複製給 p 指到的位置、再判斷 p(為 False)、停止。

Array of Strings

```
char planets[][8] = {"Mercury", "Venus", "Earth",
                     "Mars", "Jupiter", "Saturn",
                     "Uranus", "Neptune", "Pluto"};
```

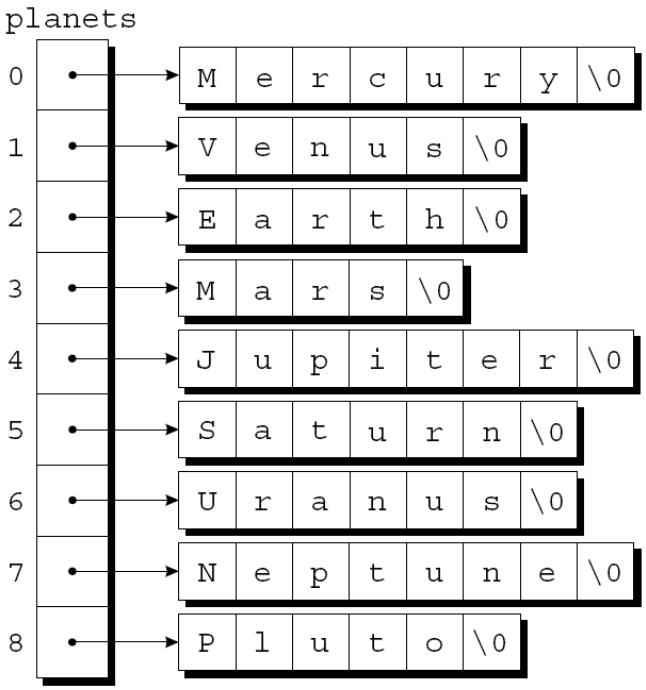
這是一個 char 的二維陣列，裡面每一個字串的長度都相等 (8 個)，但這會造成對於有些字串來說會多出一些 \0。
在這裡我們不會特別討論 Row (橫列(字串個數))，而是針對 Column (直列(字串長度))調整。

	0	1	2	3	4	5	6	7
0	M	e	r	c	u	r	y	\0
1	V	e	n	u	s	\0	\0	\0
2	E	a	r	t	h	\0	\0	\0
3	M	a	r	s	\0	\0	\0	\0
4	J	u	p	i	t	e	r	\0
5	S	a	t	u	r	n	\0	\0
6	U	r	a	n	u	s	\0	\0
7	N	e	p	t	u	n	e	\0
8	P	l	u	t	o	\0	\0	\0

如果要改善這些多餘的空間，我們可以用 ragged array：

```
char *planets[] = {"Mercury", "Venus", "Earth",
                   "Mars", "Jupiter", "Saturn",
                   "Uranus", "Neptune", "Pluto"};
```

planets 是一個由指標組成的陣列 (array of pointer)



每一個字串長度都可以調整成只有一個 \0

再 planets 裡找出第一個字母是 M 的字串：

```
for (i = 0; i < 9; i++)
    if (planets[i][0] == 'M')
      printf("%s begins with M\n", planets[i]);
```

Command-Line Arguments

Example of UNIX ls command:

1. 輸入 ls 在 command line 上
   ls 會顯示在 current dictionary 中所有檔案的名字
2. 輸入 ls -1 在 command line 上
   ls 會顯示一列檔案，展示每個檔案詳細的資訊，像是擁有者、最近更新時間等等
3. 輸入 ls -1 remind.c 在 command line 上
   ls 會顯示名叫 remind.c 這個檔案的詳細資訊

- To obtain access to ***command-line arguments***, `main` must have two parameters:

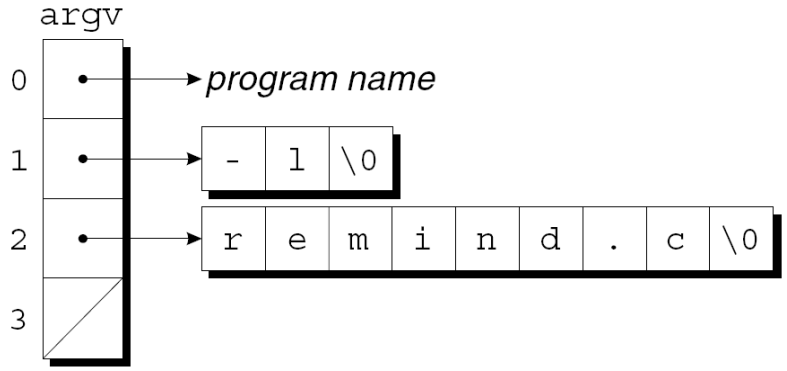
```
int main(int argc, char *argv[])
{
  …
}
```

1. argc (argument count)是 command-line argument 的個數
2. argv (argument vector)是一個由 command-line argument 組成的陣列(是上一章提到的 ragged array)：
   argv[0]指向程式的名稱、
   argv[1]到 argv[argc-1]指向剩下的 command-line argument
   argv[argc]指向 NULL，是一個 null pointer
   null pointer is a special pointer point to nothing—point to macro NULL

If the user enters the command line

ls -l remind.c

then argc will be 3, and argv will have the following appearance:



印出每一行指令
用迴圈：

```
int i;

  for (i = 1; i < argc; i++)
    printf("%s\n", argv[i]);
```

用指標：

```
char **p;

  for (p = &argv[1]; *p != NULL; p++)
    printf("%s\n", *p);
```

一開始宣告 p 用 char **是因為 argv 是一個二維陣列，且希望 p 可以指到 argv 裡的字串；
先將 p 指向 argv[1]，再依序印出 argv 的每一個字串直到 p 是 null pointer (指到的內容是 NULL)