CHAPTER 8 - CHARACTERS AND STRINGS

8.2 Fundamentals of Strings and Characters

- 字串宣告(String declarations)
  - `char color[] = "blue";`
  - `char *colorPtr = "blue";`
  - 字串基本上是字元陣列，最後面再加上 `'\0'`
    - `color` has 5 elements
- 輸入字串(Inputting strings)
  - `scanf`
    - `char word[10];`
      `scanf("%s", word);`
    - Copies input into `word[]`
    - Do not need `&` (because a string is a pointer)
  - 記住必須為`'\0'`預留空間

常用字串函式或巨集

- 字串長度:strlen
- 字串複製:strcpy/strncpy
- 字串串接:strcat/strncat
- 字串比較:strcmp/strncmp/stricmp
- 字串與數值間轉換:atol/atoi/atof/strtod/strtol/sprintf/sscanf
- 字串拆解:strtok/strstr
- 字元檢查:islower/isupper/toupper/tolower/isdigit/isprint/….

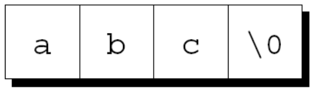
String literals

- **以" "來標示。" "範圍內的都是這個字串的字元。**
- 萬一字串太長想換行接續寫請加上\

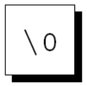
  "a very long string \
      a long string"

  或是"a very long string "
    "a long string"

  編譯器會自動將兩個相鄰的字串接成一個如"XYZ" "ABC" "XYZABC"
- **長度為字元數再加上 '\0'**
  - "abc" has **4** elements.
  - "" has **one** element.



- **以'\0'字元為結尾的字元陣列**



Operations on string literals

- 編譯器其實將string literal以char *方式處理.
  - 例如:

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

    編譯器會將'a'，'b'，'c'，'\0'安排在字元陣列。printf得到的是'a'所在的位置。

- 因此可以有這樣操作

  -
    ```
    char *p;
        p = "abc";
    ```

  -
    ```
    char ch;
        ch = "abc"[1]; // ch is 'b' now
    ```

//A function that converts a number between 0 and 15 into the equivalent hex digit.

```
char digit_to_hex_char(int digit)
{
  return "0123456789ABCDEF"[digit];
}
```

Operations on string literals (cont'd)

- 修改 string literal會導致沒定義的結果(會當掉或是怪怪地)

```
char *p = "abc";

*p = 'd';     /*** WRONG ***/
```

String literals versus character constants

- 只包含一個字元的字串與單一字元仍是不同的。

  `"a"` 是一個a *pointer*.

  `'a'` 是一個(char)(*integer*).

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

```
printf('\n');    /*** WRONG ***/
```

因為`printf`第一個參數其型態必須是`const char*`

String variables

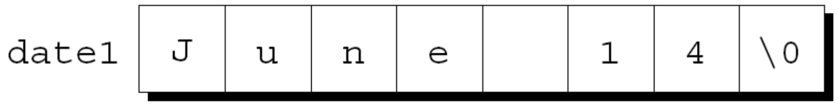
- 你可以採用下面方式為字串來宣告字元陣列

  #define STR_LEN 80 //字串長度最多80

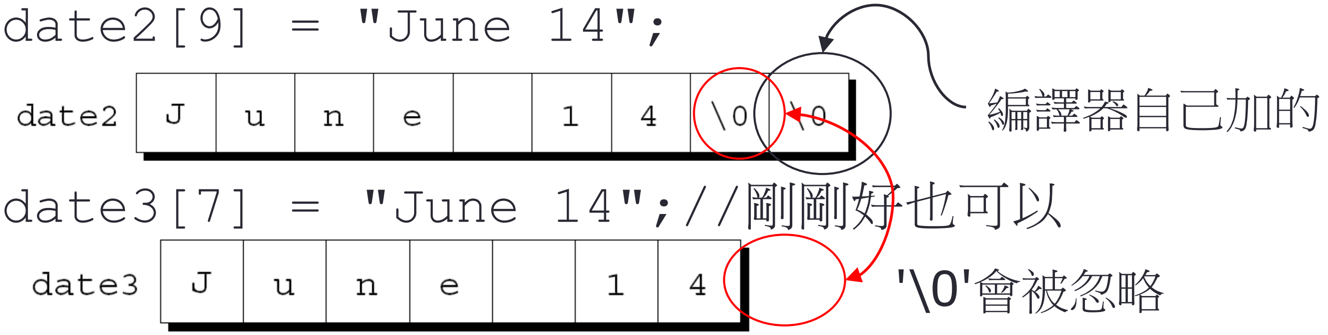
  char str[STR_LEN+1];

  //加一個字元來裝'\0'

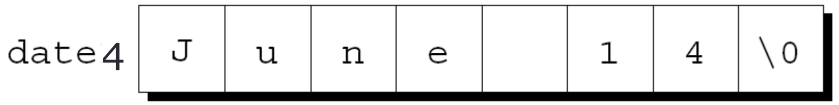
- 字串變數初始化 (其實就是把date1看成字元陣列)
  - `char date1[8] = "June 14";`



  - `char date2[9] = "June 14";`



  - `char date3[7] = "June 14";//剛剛好也可以`
  - `char date4[] = "June 14";//由編譯器自己數`



Character arrays versus character pointers

- date宣告成字元陣列

  `char date[] = "June 14";`

  與宣告成字元指標其實可以混用

  `char *date = "June 14";`
- 不同處
  - 宣告成字元陣列，其每個字元都可以修改。而第二種方式則不應該去改其字元（程式會怪怪的）
  - 宣告成字元陣列date 是陣列名字(char *const)不可指向其他地方。而第二種方式可以。

Character arrays versus character pointers (cont'd)

- 下面的宣告並未對字串配置任何空間

```
char *p;

p[0] = 'a';     /*** WRONG ***/
p[1] = 'b';     /*** WRONG ***/
p[2] = 'c';     /*** WRONG ***/
p[3] = '\0';    /*** WRONG ***/
```

所以在將p當字串用之前必須先指向字串如下例

```
char str[STR_LEN+1], *p;

p = str;
```

- 另一種方式則是指向動態配置空間 (以後會講)

輸出入字串

- 輸入字串
  - **scanf, gets 一直讀到換行符號，所以可以讀整行**
    - **char word[10];**
      scanf("%s", word);
      scanf("%9s", word);
      // gets則沒法設定長度限制，改用fgets
  - 記住必須為'\0'預留空間

char sentence[SENT_LEN+1];
scanf("%s", sentence);
若輸入
 To C, or not to C: that is the question.
scanf將存 "To"到sentence.
若改用gets
gets(sentence);
gets將存" To C, or not to C: that is the question."到sentence.

- 輸出字串
  - **printf, puts 會自動跳行**

```
char str[] = "Are we having fun yet?";
```

- `printf("%s\n", str);`

```
    Are we having fun yet?
```

- `printf("%.2s\n", str);`

```
    Ar
```

%m.ps,
m欄位寬
p精確度
內定往右對齊
%-m.ps
往左對齊

自製輸入字串函數

- 若是我們需要下面功能的自串輸入函數
  - (1) doesn’t skip white-space characters,
  - (2) stops reading at the first new-line character (which isn’t stored in the string), and

那麼我們必須逐字元逐字元的輸入與檢查

```
int read_line(char str[], int n)
{
  int ch, i = 0;

  while ((ch = getchar()) != '\n' && i < n)
      str[i++] = ch;
  str[i] = '\0';    /* terminates string */
  return i;         /* number of characters stored */
}
```

Accessing the characters in a string

- 計算字串內空白字元數

```
int count_spaces(const char s[])
{
   int count = 0, i;

   for (i = 0; s[i] != '\0'; i++)
      if (s[i] == ' ')
         count++;
   return count;
}
```

```
int count_spaces(const char *s)
 {
   int count = 0;

   for (; *s != '\0'; s++)
     if (*s == ' ')
       count++;
   return count;
 }
```

字串字元複製/字串內容比較

- 錯誤例子:

```
char str1[10], str2[10], *str;
…
str1 = "abc";   /*** WRONG ***/
str2 = str1;    /*** WRONG ***/
```

Why??? 到底作了什麼？

```
str = "abs";    /* just pointer assignment */
```

- 為什麼宣告時就可以??

```
char str1[10] = "abc";
```

- `str1 == str2` 是比較其字串內容嗎？？

8.3 Character Handling Library

ctype.h

Prototype	Description
`int isdigit( int c )`	Returns **`true`** if **`c`** is a digit and **`false`** otherwise.
`int isalpha( int c )`	Returns **`true`** if **`c`** is a letter and **`false`** otherwise.
`int isalnum( int c )`	Returns **`true`** if **`c`** is a digit or a letter and **`false`** otherwise.
`int isxdigit( int c )`	Returns **`true`** if **`c`** is a hexadecimal digit character and **`false`** otherwise.
`int islower( int c )`	Returns **`true`** if **`c`** is a lowercase letter and **`false`** otherwise.
`int isupper( int c )`	Returns **`true`** if **`c`** is an uppercase letter; **`false`** otherwise.
`int tolower( int c )`	If **`c`** is an uppercase letter, **`tolower`** returns **`c`** as a lowercase letter. Otherwise, **`tolower`** returns the argument unchanged.
`int toupper( int c )`	If **`c`** is a lowercase letter, **`toupper`** returns **`c`** as an uppercase letter. Otherwise, **`toupper`** returns the argument unchanged.
`int isspace( int c )`	Returns **`true`** if **`c`** is a white-space character—newline (**`'\n'`**), space (**`' '`**), form feed (**`'\f'`**), carriage return (**`'\r'`**), horizontal tab (**`'\t'`**), or vertical tab (**`'\v'`**)—and **`false`** otherwise
`int iscntrl( int c )`	Returns **`true`** if **`c`** is a control character and **`false`** otherwise.
`int ispunct( int c )`	Returns **`true`** if **`c`** is a printing character other than a space, a digit, or a letter and **`false`** otherwise.
`int isprint( int c )`	Returns **`true`** value if **`c`** is a printing character including space (**`' '`**) and **`false`** otherwise.
`int isgraph( int c )`	Returns **`true`** if **`c`** is a printing character other than space (**`' '`**) and **`false`** otherwise.

```
/* Fig. 8.2: fig08_02.c
   Using functions isdigit, isalpha, isalnum, and isxdigit */
#include <stdio.h>
#include <ctype.h>

int main()
{
   printf( "%s\n%s%s\n%s%s\n\n", "According to isdigit: ",
       isdigit( '8' ) ? "8 is a " : "8 is not a ", "digit",
       isdigit( '#' ) ? "# is a " :
       "# is not a ", "digit" );
   printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
       "According to isalpha:",
       isalpha( 'A' ) ? "A is a " : "A is not a ", "letter",
       isalpha( 'b' ) ? "b is a " : "b is not a ", "letter",
       isalpha( '&' ) ? "& is a " : "& is not a ", "letter",
       isalpha( '4' ) ? "4 is a " :
       "4 is not a ", "letter" );
   printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
       "According to isalnum:",
       isalnum( 'A' ) ? "A is a " : "A is not a ",
       "digit or a letter",
       isalnum( '8' ) ? "8 is a " : "8 is not a ",
       "digit or a letter",
       isalnum( '#' ) ? "# is a " : "# is not a ",
       "digit or a letter" );
   printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s\n",
       "According to isxdigit:",
       isxdigit( 'F' ) ? "F is a " : "F is not a ",
       "hexadecimal digit",
       isxdigit( 'J' ) ? "J is a " : "J is not a ",
       "hexadecimal digit",
       isxdigit( '7' ) ? "7 is a " : "7 is not a ",
       "hexadecimal digit",
       isxdigit( '$' ) ? "$ is a " : "$ is not a ",
       "hexadecimal digit",
       isxdigit( 'f' ) ? "f is a " : "f is not a ",
       "hexadecimal digit" );
   return 0;
}
```

```
According to isdigit:
8 is a digit
# is not a digit

According to isalpha:
A is a letter
b is a letter
& is not a letter
4 is not a letter

According to isalnum:
A is a digit or a letter
8 is a digit or a letter
# is not a digit or a letter

According to isxdigit:
F is a hexadecimal digit
J is not a hexadecimal digit
7 is a hexadecimal digit
$ is not a hexadecimal digit
f is a hexadecimal digit
```

8.4 字串轉換函數(String Conversion Functions)

- #include **<stdlib.h>** (general utilities library)
- 將數字字串轉換為整數或浮點數

If the converted value cannot be represented, the behaviors of the three functions are undefined.

Prototype	Description
`double atof( const char *nPtr )`	Converts the string **nPtr** to **double**.
`int atoi( const char *nPtr )`	Converts the string **nPtr** to **int**.
`long atol( const char *nPtr )`	Converts the string **nPtr** to long **int**.
`double strtod( const char *nPtr, char **endPtr )`	Converts the string **nPtr** to **double**.
`long strtol( const char *nPtr, char **endPtr, int base )`	Converts the string **nPtr** to **long**.
`unsigned long strtoul( const char *nPtr, char **endPtr, int base )`	Converts the string **nPtr** to **unsigned** isgraph **long**.

有關Base

- *base*可以是0或2到36的數字
- 如果 *base*是0, *nptr*所指的第一個字元會被用來決定base.
  - 第一個字元是 ‘0’且第二個字元不是‘x’或 ‘X’, 就會使用八進位;否則就是十進位。
  - 第一個字元是 ‘0’且第二個字元是‘x’或 ‘X’, 就會使用十六進位
  - 第一個字元是 ‘1’到 ‘9’就是十進位

```c
/* Fig. 8.6: fig08_06.c
   Using atof */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   double d;

   d = atof( "99.0" );
   printf( "%s%.3f\n%s%.3f\n",
           "The string \"99.0\" converted to double is ", d,
           "The converted value divided by 2 is ",
           d / 2.0 );
   return 0;
}
```

```
The string "99.0" converted to double is 99.000
The converted value divided by 2 is 49.500
```

Fig. 8.9

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   /* initialize string pointer */
   const char *string = "51.2% are admitted"; /* initialize string */

   double d;        /* variable to hold converted sequence */
   char *stringPtr; /* create char pointer */

   d = strtod( string, &stringPtr );

   printf( "The string \"%s\" is converted to the\n", string );
   printf( "double value %.2f and the string \"%s\"\n", d, stringPtr );

   return 0; /* indicates successful termination */

} /* end main */
```

```
The string "51.2% are admitted" is converted to the
double value 51.20 and the string "% are admitted"
```

Fig. 8.10

```
/* Fig. 8.10: fig08_10.c
   Using strtol */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   const char *string = "-1234567abc"; /* initialize string pointer */

   char *remainderPtr; /* create char pointer */
   long x;             /* variable to hold converted sequence */

   x = strtol( string, &remainderPtr, 0 );

   printf( "%s\"%s\"\n%s%ld\n%s\"%s\"\n%s%ld\n",
           "The original string is ", string,
           "The converted value is ", x,
           "The remainder of the original string is ",
           remainderPtr,
           "The converted value plus 567 is ", x + 567 );

   return 0; /* indicates successful termination */

} /* end main */
```

```
The original string is “-1234567abc”
The converted value is -1234567
The remainder of the original string is “abc”
The converted value plus 567 is -1234000
```

安全地輸入數字

```
int num;
puts("input an integer between 0 and 100");
scanf("%d",&num);
//當使用者不輸入整數或在範圍外時會造成錯誤
char buf[100];
fgets(buf,100,stdin);
num = strtol(buf,NULL,0);
if (errno == ERANGE || num < 0 || num > 100) {
   // error handling
}
```

8.5 Standard Input/Output Library Functions

- 在 <stdio.h>裡有關處理字串或字元函數

Function prototype	Function description
`int getchar( void );`	Inputs the next character from the standard input and returns it as an integer.
`char *gets( char *s );`	Inputs characters from the standard input into the array **s** until a newline or end-of-file character is encountered. A terminating null character is appended to the array.
`int putchar( int c );`	Prints the character stored in **c**.
`int puts( const char *s );`	Prints the string **s** followed by a newline character.
`int sprintf( char *s, const char *format, ... );`	Equivalent to **printf**, except the output is stored in the array **s** instead of printing it on the screen.
`int sscanf( char *s, const char *format, ... );`	Equivalent to **scanf**, except the input is read from the array **s** instead of reading it from the keyboard.

```c
/* Fig. 8.13: fig08_13.c
   Using gets and putchar */
#include <stdio.h>

int main()
{
   char sentence[ 80 ];
   void reverse( const char * const );

   printf( "Enter a line of text:\n" );
   gets( sentence );

   printf( "\nThe line printed backwards is:\n" );
   reverse( sentence );

   return 0;
}

void reverse( const char * const sPtr )
{
   if ( sPtr[ 0 ] == '\0' )
      return;
   else {
      reverse( &sPtr[ 1 ] );
      putchar( sPtr[ 0 ] );
   }
}
```

`reverse` calls itself using substrings of the original string. When it reaches the **`'\0'`** character it prints using **`putchar`**

```
Enter a line of text:
Characters and Strings

The line printed backwards is:
sgnirtS dna sretcarahC
```

Example of sprintf and sscanf

```
#include<stdio.h>
#include<string.h>
int main()
{
            char msg[100],msg2[10];
            int year = 2017;
            sprintf(msg,"Happy new year %d",year);
            strcat(msg," %d\n");
            printf(msg,10);         // printf("Happy new year 2017 %d\n",10);
            sscanf(msg,"%s",msg2);
            printf("%s\n",msg2);
            return 0;
}
```

Output:

```
Happy new year 2017 10
Happy
```

8.6 String Manipulation Functions of the String Handling Library

- <string.h>
-

Function prototype	Function description
`char *strcpy( char *s1, const char *s2 )`	Copies string **s2** into array **s1**. The value of **s1** is returned.
`char *strncpy( char *s1, const char *s2, size_t n )`	Copies at most **n** characters of string **s2** into array **s1**. The value of **s1** is returned.
`char *strcat( char *s1, const char *s2 )`	Appends string **s2** to array **s1**. The first character of **s2** overwrites the terminating null character of **s1**. The value of **s1** is returned.
`char *strncat( char *s1, const char *s2, size_t n )`	Appends at most **n** characters of string **s2** to array **s1**. The first character of **s2** overwrites the terminating null character of **s1**. The value of **s1** is returned.
`size_t strlen(const char *s)`	Count the number of characters of string **s**.

Copying a String: strcpy

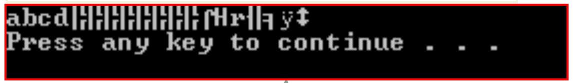
- 複製字串

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

Copying a String: strncpy

- Example:
  char str1[10], str2[10];
  strcpy(str2, "abcd");
    /* str2 now contains "abcd" */
  strcpy(str1, str2);
    /* str1 now contains "abcd" */
- strcpy並沒有檢查str1裝得下str2的字串，萬一裝不下程式就會出錯，較保險的函數是用strncpy
  strncpy(str1, str2, sizeof(str1));
  最多複製這麼多字元
- 但是萬一str2的長度超過sizeof(str1)，'\0'將不會擺到str1的尾巴，因此最安全的寫法是
  strncpy(str1, str2, sizeof(str1) - 1);
  str1[sizeof(str1)-1] = '\0';

```
#include <stdio.h>
#include <string.h>
int main(void)
{
   char a[5];
   char b[6] = "abcde";
   strncpy(a,b,4);
   printf("%s\n",a);
   return 0;
}
```



```
abcd[illegible]
Press any key to continue . . .
```

Counting the number of characters: strlen

```
/* return the length of s */
size_t strlen(const char *s)
    {
      size_t n;

      for (n = 0; *s != '\0'; s++)
        n++;
      return n;
    }
```

```
size_t strlen(const char *s)
 {
   const char *p = s;

   while (*s)
     s++;
   return s - p;
 }
```

```
int len;

len = strlen("abc");   /* len is now 3 */
len = strlen("");      /* len is now 0 */
strcpy(str1, "abc");
len = strlen(str1);    /* len is now 3 */
```

Searching for the End of a String

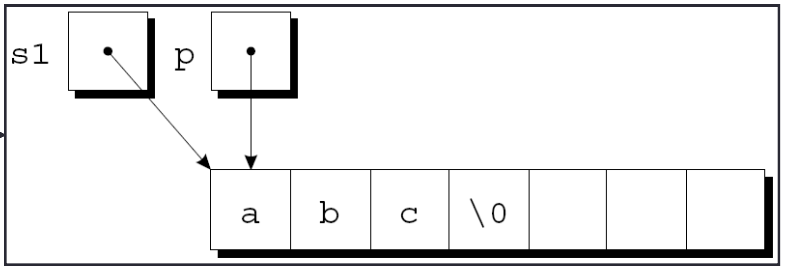
- 找字串的null character('\0')的迴圈:

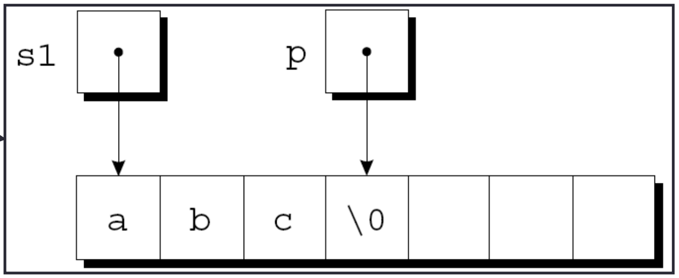
```
while (*s)          while (*s++)
  s++;                ;
```

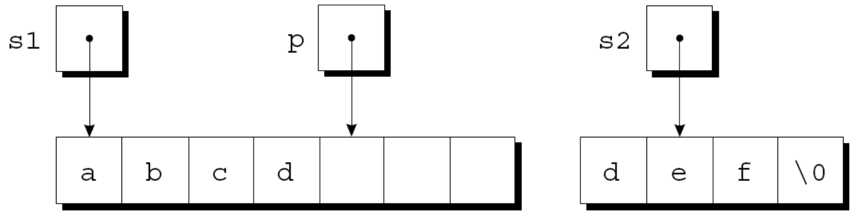
- 第一個迴圈結束時，s指向null character
- 第二個迴圈結束時，s指向null character的下一個字元

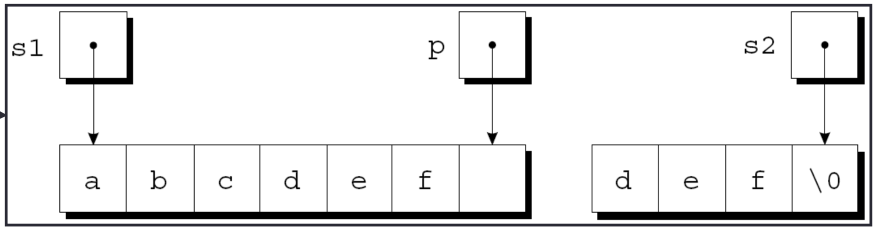
Concatenating two strings:strcat

```
char *strcat(char *s1, const char *s2)
{
  char *p = s1;

  while (*p != '\0')
    p++;
  while (*p++ = *s2++);
  return s1;
}
```









Concatenating two strings:strncat

```
char *strcat(char *s1, const char *s2);
```

- strcat 將str2的內容附加到str1內容的尾巴
  Examples:

```
strcpy(str1, "abc");
strcat(str1, "def");
  /* str1 now contains "abcdef" */
strcpy(str1, "abc");
strcpy(str2, "def");
strcat(str1, str2);
  /* str1 now contains "abcdef" */
```

當str1裝不下原本str1內容與str2時，strcat與strcpy有相同的問題，用strncat較安全

```
strncat(str1, str2, sizeof(str1) - strlen(str1) - 1);
```

- 不過不像strncpy，strncat一定會擺上'\0'

What is the difference between them?

8.7 Comparison Functions of the String Handling Library

- 比較字串(Comparing strings)

```
int strcmp( const char *s1, const char *s2 );
```

- 比較字串 s1 與 s2的大小
  - 如果 s1內容 < s2內容傳回負數,
  - 如果s1與s2內容一樣傳回0,
  - 如果s1內容 > s2內容傳回正數

```
int stricmp( const char *s1, const char *s2 );
```

- 比較字串 s1 與 s2的大小(不分大小寫)

```
int strncmp( const char *s1, const char *s2,
      size_t n );
```

- 最多只比較前n字元

Arrays of Strings

- 方法一：二維字元陣列

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

- 字串長短不一時這種會浪費一點空間。

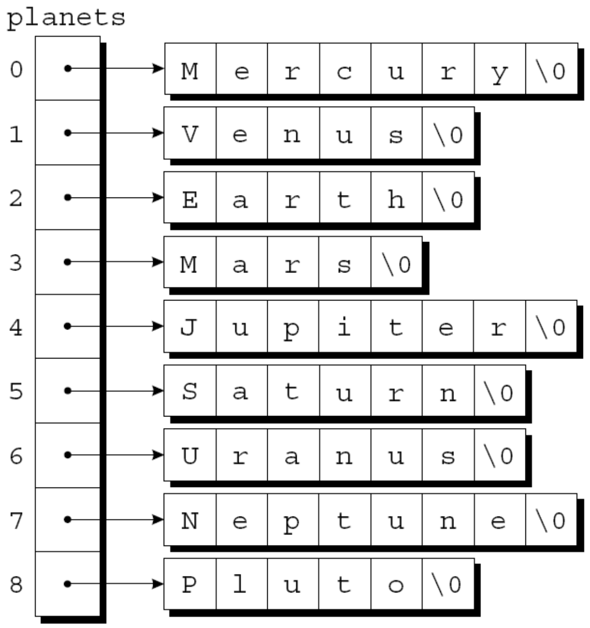
	0	1	2	3	4	5	6	7
0	M	e	r	c	u	r	y	\0
1	V	e	n	u	s	\0	\0	\0
2	E	a	r	t	h	\0	\0	\0
3	M	a	r	s	\0	\0	\0	\0
4	J	u	p	i	t	e	r	\0
5	S	a	t	u	r	n	\0	\0
6	U	r	a	n	u	s	\0	\0
7	N	e	p	t	u	n	e	\0
8	P	l	u	t	o	\0	\0	\0

Arrays of Strings

- 方法二：字元指標陣列

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

- 方法一與二的差異在哪裡??



Command-Line Arguments

- 執行程式時可透過命令列傳遞必要程式參數給程式。
- Examples

LINUX指令

```
ls
ls -l
ls -l remind.c
```

WINDOWS指令

```
shutdown -s -t 0
```

Command-Line Arguments

- main必須藉由兩個參數得到程式參數

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

- argc程式參數個數，至少有一個(程式名稱)
- argv 字串指標陣列，指向程式參數字串
- argv[0] 指向程式名稱字串，argv[1]至argv[argc-1]指向程式參數字串
- argv[argc]是 ***null pointer***—a special pointer that points to nothing.
  - The macro NULL represents a null pointer.

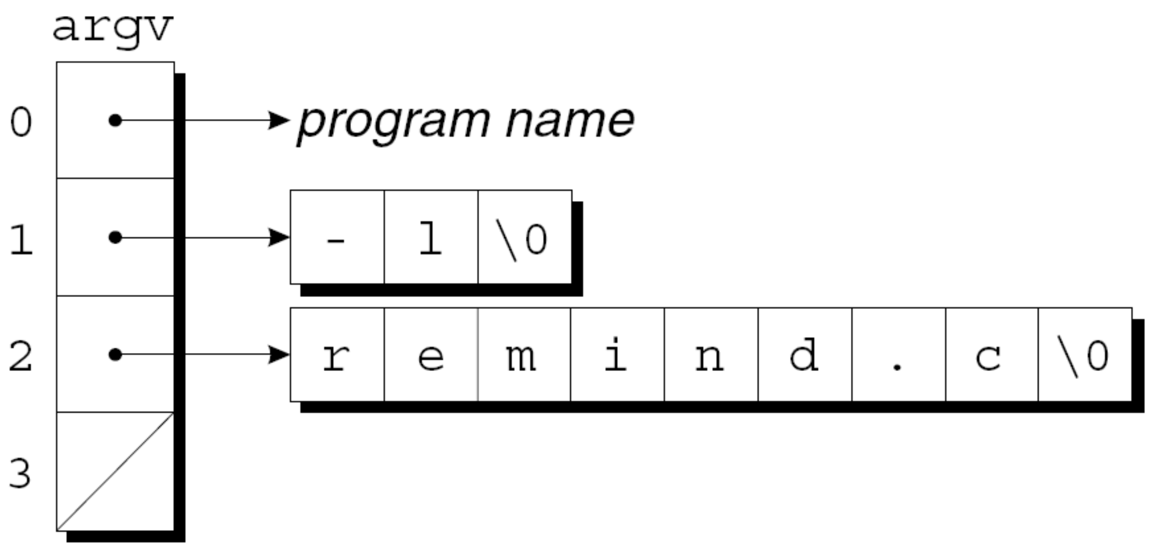
Command-Line Arguments

- EXAMPLE

  ```
  ls -l remind.c
  ```

  argc為3,

  argv是下面樣子:



8.8 Search Functions of the String Handling Library

Function prototype	Function description
`char *strchr( const char *s, int c );`	Locates the first occurrence of character **`c`** in string **`s`**. If **`c`** is found, a pointer to **`c`** in **`s`** is returned. Otherwise, a **`NULL`** pointer is returned.
`size_t strcspn( const char *s1, const char *s2 );`	Determines and returns the length of the initial segment of string **`s1`** consisting of characters not contained in string **`s2`**.
`size_t strspn( const char *s1, const char *s2 );`	Determines and returns the length of the initial segment of string **`s1`** consisting only of characters contained in string **`s2`**.
`char *strpbrk( const char *s1, const char *s2 );`	Locates the first occurrence in string **`s1`** of any character in string **`s2`**. If a character from string **`s2`** is found, a pointer to the character in string **`s1`** is returned. Otherwise, a **`NULL`** pointer is returned.
`char *strrchr( const char *s, int c );`	Locates the last occurrence of **`c`** in string **`s`**. If **`c`** is found, a pointer to **`c`** in string **`s`** is returned. Otherwise, a **`NULL`** pointer is returned.
`char *strstr( const char *s1, const char *s2 );`	Locates the first occurrence in string **`s1`** of string **`s2`**. If the string is found, a pointer to the string in **`s1`** is returned. Otherwise, a **`NULL`** pointer is returned.
`char *strtok( char *s1, const char *s2 );`	A sequence of calls to **`strtok`** breaks string **`s1`** into "tokens"—logical pieces such as words in a line of text—separated by characters contained in string **`s2`**. The first call contains **`s1`** as the first argument, and subsequent calls to continue tokenizing the same string contain **`NULL`** as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, **`NULL`** is returned.

```c
/* Fig. 8.27: fig08_27.c
   Using strspn */
#include <stdio.h>
#include <string.h>

int main()
{
   const char *string1 = "The value is 3.14159";
   const char *string2 = "aehi lsTuv";

   printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
           "string1 = ", string1, "string2 = ", string2,
           "The length of the initial segment of string1",
           "containing only characters from string2 = ",
           strspn( string1, string2 ) );
   return 0;
}
```

```
string1 = The value is 3.14159
string2 = aehi lsTuv

The length of the initial segment of string1
containing only characters from string2 = 13
```

```c
/* Fig. 8.29: fig08_29.c
   Using strtok */
#include <stdio.h>
#include <string.h>

int main()
{
   char string[] = "This is a sentence with 7 tokens";
   char *tokenPtr;

   printf( "%s\n%s\n\n%s\n",
           "The string to be tokenized is:", string,
           "The tokens are:" );

   tokenPtr = strtok( string, " " );

   while ( tokenPtr != NULL ) {
      printf( "%s\n", tokenPtr );
      tokenPtr = strtok( NULL, " " );
   }

   return 0;
}
```

```
The string to be tokenized is:
This is a sentence with 7 tokens

The tokens are:
This
is
a
sentence
with
7
tokens
```

Effect of strtok on s1

```
#include<stdio.h>
#include<string.h>
int main()
{
   char p[] = "(abc(def(ghi";
   int i;
   char *q;

   int len = sizeof(p);
   printf("p:%s\n",p);
   q = strtok(p,"(");
   while(q!= NULL) {
         printf("q:%s\np:",q);
         for(i = 0; i < len; i++) {
             if (p[i]!='\0') {
                putchar(p[i]);
             } else {
                printf("\\0");
                }
         }
         printf("\n");
         q = strtok(NULL,"(");
   }
   return 1;
}
```

strtok may change th
separation character
in p to '\0'.

```
// the first call of strtok
q=strtok(p,"(");

//  the second call of strtok
q=strtok(NULL,"(");

// the third call of strtok
q=strtok(NULL,"(");
```

Output:

```
p:(abc(def(ghi
q:abc
p:(abc\0def(ghi\0
q:def
p:(abc\0def\0ghi\0
q:ghi
p:(abc\0def\0ghi\0
```

8.9 Memory Functions of the String-handling Library

- 有關記憶體函數(Memory Functions)
  - In `<stdlib.h>`
  - 能夠處理任何型態的記憶體區塊
  - 搬移，複製，設定數值，比較，搜尋。
- 參數是void* (Pointer parameters are `void *`)

8.9 Memory Functions of the String-

Prototype	Description

`void *memcpy( void *s1,`
`const void *s2,`
`size_t n )`
	Copies `n` characters from the object pointed to by `s2` into the object pointed to by `s1`. A pointer to the resulting object is returned.

`void *memmove( void *s1,`
`const void *s2,`
`size_t n )`
	Copies `n` characters from the object pointed to by `s2` into the object pointed to by `s1`. The copy is performed as if the characters are first copied from the object pointed to by `s2` into a temporary array, and then copied from the temporary array into the object pointed to by `s1`. A pointer to the resulting object is returned.

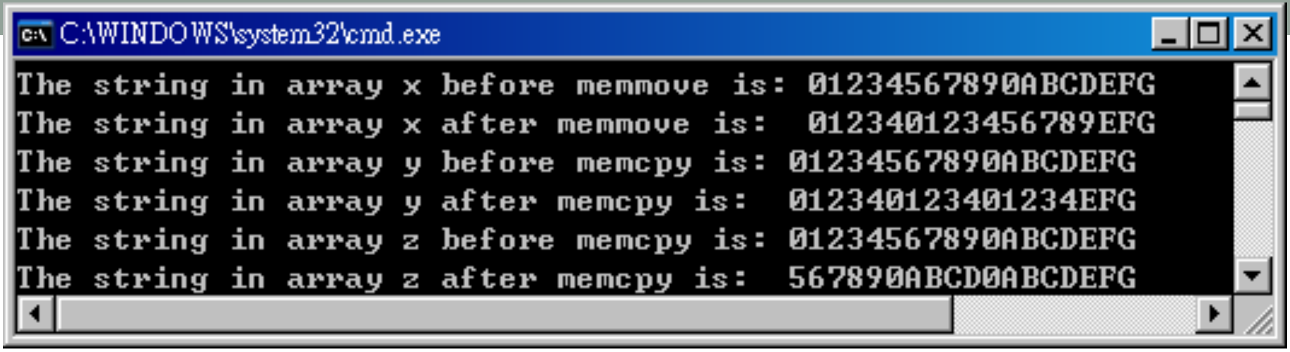
`int memcmp( const void *s1,`
`const void *s2,`
`size_t n )`
	Compares the first `n` characters of the objects pointed to by `s1` and `s2`. The function returns `0`, less than `0`, or greater than `0` if `s1` is equal to, less than or greater than `s2`, respectively.

`void *memchr(const void *s,`
`int c, size_t n )`
	Locates the first occurrence of `c` (converted to `unsigned char`) in the first `n` characters of the object pointed to by `s`. If `c` is found, a pointer to `c` in the object is returned. Otherwise, `0` is returned.

`void *memset( void *s,`
`int c, size_t n )`
	Copies `c` (converted to `unsigned char`) into the first `n` characters of the object pointed to by `s`. A pointer to the result is returned.

```c
/* Fig. 8.32: fig08_32.c
   Using memmove */
#include <stdio.h>
#include <string.h>

int main()
{
   char x[] = "Home Sweet Home";

   printf( "%s%s\n",
           "The string in array x before memmove is: ", x );
   printf( "%s%s\n",
           "The string in array x after memmove is:  ",
           memmove( x, &x[ 5 ], 10 ) );

   return 0;
}
```

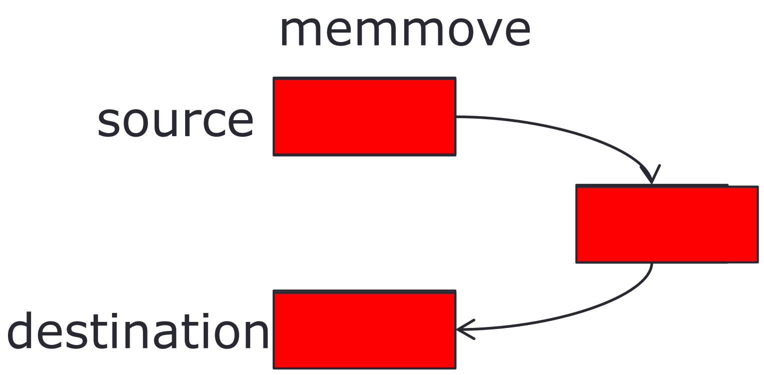
```
The string in array x before memmove is: Home Sweet Home
The string in array x after memmove is:  Sweet Home Home
```

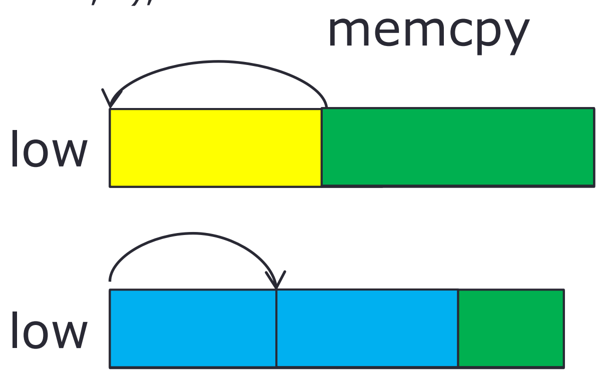


```
The string in array x before memmove is: 01234567890ABCDEFG
The string in array x after memmove is:  0123401234567890EFG
The string in array y before memcpy is: 01234567890ABCDEFG
The string in array y after memcpy is:  01234012340123 4EFG
The string in array z before memcpy is: 01234567890ABCDEFG
The string in array z after memcpy is:  567890ABCD0ABCDEFG
```

```
#include<stdio.h>
#include<string.h>
void main()
{
        char x[] = "01234567890ABCDEFG";
        char y[] = "01234567890ABCDEFG";
        char z[] = "01234567890ABCDEFG";

        printf( "%s%s\n","The string in array x before memmove is: ", x );
        memmove( &x[ 5 ], x, 10 ) ;
        printf( "%s%s\n","The string in array x after memmove is:  ", x);

        printf( "%s%s\n", "The string in array y before memcpy is: ", y );
        memcpy( &y[ 5 ], y, 10 ) ;
        printf( "%s%s\n", "The string in array y after memcpy is:  ", y);

        printf( "%s%s\n", "The string in array z before memcpy is: ", z );
        memcpy( z, &z[ 5 ], 10 ) ;
        printf( "%s%s\n", "The string in array z after memcpy is:  ", z);

}
```

memcpy由低位置複製至高位置時，要注意!!





8.10 Other Functions of the String Handling Library

- `char *strerror( int errornum );`
  - 根據錯誤代號 **errornum**傳回錯誤訊息字串

```c
/* Fig. 8.37: fig08_37.c
   Using strerror */
#include <stdio.h>
#include <string.h>

int main()
{
   printf( "%s\n", strerror( 2 ) );
   return 0;
}
```

```
No such file or directory
```