國立屏東大學 資訊工程學系 程式設計(二)

14 字串

14.1 字串常值(String Literals)

所謂的字串(string)是指一些字元的集合，例如“Hello”這個用雙引號框起來的字元集合就是一個字串。在C語言中，這種用雙引號框起來的字串又稱為字串常值(string literals)，我們已經在printf()函式中時常使用到。每個在程式中的字串常值，將會在編譯時自動配置到某個記憶體位址，這個記憶體位址為一塊連續的記憶體空間，包含多個字元儲存於其中，並以一個特殊的字元『\0』代表字串的結尾。通常『\0』又被稱為是null character(空字元)。請參考figure 1，其中顯示了一個包含有“Hello”字串的記憶體空間，並以\0做為字串的結束。

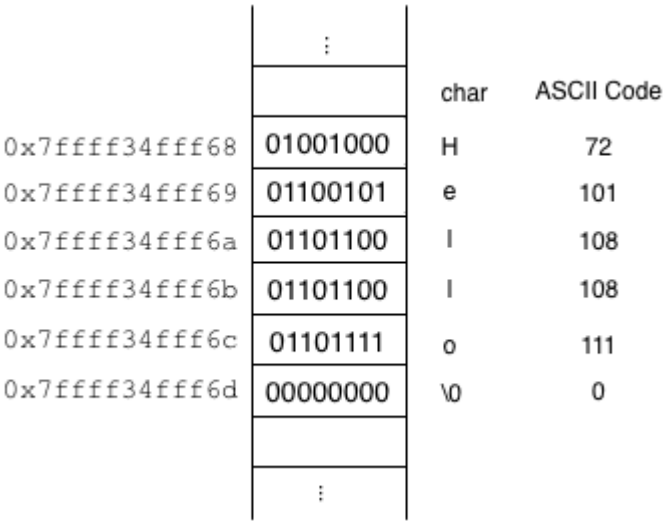

Fig. 1: 在記憶體中的字串常值

當然，你也可以簡單得以figure 2來思考這個在記憶體中的字串常值。

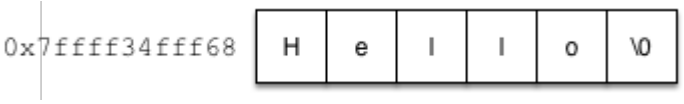

Fig. 2: 在記憶體中的字串常值

下面這個程式，以指標來檢視儲存在記憶體中的字串常值，請將它加以編譯並執行看看其結果為何。

helloString.c

```c
#include <stdio.h>

int main()
{
  char *p;
  char *q;
  int i;

  p="Hello";
  q=p;
```

```
  printf("%s\n", p); // %s為字串的format specifier

  for(i=;i<6;i++)
  {
    printf("%c = %d at %p\n", *q, *q, q);
    q++;
  }
  return ;
}
```

當然，經過上述的說明，可能你已經想到也可以使用陣列來處理字串，請參考下面的程式：

helloString2.c

```
#include <stdio.h>

int main()
{
  int i;

  for(i=;i<6;i++)
  {
                                // 1           2           3
    printf("%c = %d at %p\n", "Hello"[i], "Hello"[i], &"Hello"[i]);
  }
  return ;
}
```

前述程式中“Hello”被當成一個陣列來使用。其中在printf()函式中，出現過三次“Hello”字串常值。原則上，在程式碼中，每出現一次字串常值，編譯器就會自動為其配置一個適當的記憶體空間，可是若是完全相同的字串內容，編譯器也會有智慧地不再重覆地配置空間。

此外，在字串常值中的字元，除ASCII的可見字元外，亦可以有逸出序列字元(escape sequences)包含\n 為換行\t 為tab\b為倒退等，詳細請參考table 1

Escape Sequence	意義
\a	Alert(bell)
\b	Backspace
\f	form feed
\n	new line
\r	carriage return
\t	Horizontal tab
\v	Vertical tab
\\	backslash
\?	?
\'	'
\“	”

Tab. 1: Escape Sequence

請參考以下的程式：

escape.c

```
#include <stdio.h>

int main()
{
  int x;

  printf("Hello\n World\t\t!:_____\b\b\b\b\b");
  scanf("%d", &x);
  printf("Your input is %d\n", x);
  return ;
}
```

當所要使用的字串常值比較長，無法在一行內表達完成時，也可以使用『\』來串接多行，或是直接以多個字串的方式處理C語言的編譯器會自動將其合併，請參考下面的程式，並且執行看看其結果為何。

multiline.c

```
#include <stdio.h>

int main()
{

  printf("12345 \
  6789\n");

  printf("12345"
  "6789\n");

  return ;
}
```

要特別、特別注意的是，字串常值的內容是不可以被更改的

在本節的最後，我們提供一個有趣的函式，輸入撲克牌的點數，它會傳回一個字元來表示該點數：

```
char pointChar(int p)
{
   return "A23456789TJQK"[p-1];
}
```

14.2 字串變數

其實，關於這點你應該已經在前一個小節的說明中發現了，在C語言中，我們有兩種方式來宣告並處理字

串變數：其一為陣列，其二為指標。若我們要宣告一個可儲存或操作含有10個字元的字串變數str那麼我們會使用下面的宣告:

```
char str[11];
```

這是因為考慮到字串必須以『\0』結尾。通常，我們會利用下面的方法來宣告一個字串：

```
#define STR_LEN 80

char str[STR_LEN + 1];
```

我們可以在宣告字串變數的同時，給定其初始值，例如：

```
char str[6] = "Hello";
```

就如同figure 2一樣C語言的編譯器會自動在其後增加一個『\0』做為結束的標記。上述的宣告等同於下列的程式碼：

```
char str[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

但是，如果我們的陣列宣告過大，所給的初始值不夠時又會如何呢?C語言的編譯器會在其不足處皆補上『\0』，因此，以下兩行程式碼是等價的：

```
char str[7] = "Hello";
char str[7] = {'H', 'e', 'l', 'l', 'o', '\0', '\0'};
```

如果你還記得我們曾說過，當陣列的初始值過少時C語言的編譯器會幫我們補0，那麼你就會瞭解為何字串變數的宣告初始值不足時會被補『\0』了(因為『\0』的整數值為0)。

如果情況反過來，初始值超過陣列的大小那又會如何？在這種情況下C無法幫我們在字串結束處補上『\0』，因此有可能在未來的操作上出現問題，請參考下面的程式：

endlessString.c

```
#include <stdio.h>

int main()
{
  char str[5]="Hello";
  printf("%s\n", str);
  return ;
}
```

請試試看編譯後執行的結果如何，再試著將str[5]改成str[4]看看會發生什麼事。有些版本的C語言編譯器會幫我們處理好這個問題，但有的不會，最好的解決方法是不要犯這種錯誤，或是改用以下的宣告方法，讓C語言的編譯器自動幫我們處理好陣列的大小問題：

```
char str[] = "Hello";
```

我們也可以用指標的方式，來進行字串變數的宣告，例如：

```
char *str = "Hello";
```

會產生一個字串變數p□但它是以指標的方式來處理。請參考下面的程式：

string.c

```
#include <stdio.h>

int main()
{
  char strA[]="Hello";
  char *strP = "World";

  printf("%s %s\n", strA, strP);

  strA[]='h';

  *strP='w';

  printf("%s %s\n", strA, strP);

  return ;
}
```

以char strA[]=“Hello”宣告的字串，其“Hello”只是被當成是陣列的初始值，編譯器還是在記憶體內建立了一個連續的空間，用以存放在該字串中的字元；可是char *strP=“World”;在編譯的過程中，它會先在記憶體內產生一個“Hello”的字串常值，然後讓指標strP指向該字串常值所在的地方。由於字串常值是不可更改內容的，因此未來strP='w';的操作會在執行時發生錯誤。

14.3 字串的輸出

在C語言裡，你可以使用printf()與puts()來進行字串的輸出。其中printf()函式必須搭配%s的format specifier來輸出字串，並可使用%x.ys來進一步指定輸出的格式，請參考以下的程式：

```
#include <stdio.h>

int main()
{
  char str[] = "Hello World";

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

  printf("%20s\n", str);

  printf("%.3s\n", str);
```

```
  printf("%10.3s\n", str);
  return ;
}
```

你也可以使用puts()函式，來將字串加以輸出，要注意的是puts()永遠會幫我們在最後增加一個換行的動作。

```
puts(str);
```

14.4 字串的輸入

14.4.1 scanf()函式

當我們宣告了字串變數後，除了用給定初始值的方式產生字串的內容外，也可以用scanf()函式取得字串的內容scanf()用以讀取字串的format specifier是%s下面的程式碼顯示了用scanf()取得使用者的姓名，並將之輸出：

```
#define LEN 10
char name[LEN + 1];

printf("Please input your name:");
scanf("%s", name);
printf("Your name is %s.\n", name);
```

在scanf()函式的格式字串中，'['是一個特殊的format specifier用以指定一個字元集合，稱為scanset所輸入的字串其必須由scanset內的字元所組成。例如我們可以定義一個scanset為[abc]那麼只有由a, b, c這三個字元所組成的字串才能被接受。請參考下面的程式碼：

```
scanf("%[atc]", str); //取回由a, t, c所組成的字串


scanf("%3[atc]", str); //取回由a, t, c所組成的字串, 並限制字串長度不超過3
```

您也可以指定不由特定字元集所組成的字串，只要在scanset的開頭加上'^'，即可，例如：

```
scanf("%[^atc]", str); //取回不包含a, t, c的字串
```

如果想限制為從a到z的字母，是否只能寫成%[abcdefghijklmnopqrstuvwxyz]?其實我們可以使用連字號'-'來解決，例如%[a-z]%[a-zA-Z]%[^a-z]%[0-3]%[a-e]%[^0-9]等。

如果我們想要限制輸入的字串含有'['或']'怎麼辦？您只需要將'['或']'放到scanset的前面即可，例如%[]abc]由a, b, c與]組成。
如果要取回(08)7238700這樣的電話格式，可以使用下列的方式：

```
   scanf(" (%[^)])%s", &area, &number);
```

最後，我們以一個程式範例結束此部份的說明：

restricted.c

```c
#include <stdio.h>

int main()
{
  char str[20];

  scanf("%[abc]", str);
  printf("%s\n", str);
  return ;
}
```

14.4.2 gets()與fgets()函式

使用scanf()在讀取字串時，有一個必須注意的問題，就是它只會讀到第一個white space就會停止輸入。因此，這個程式的執行可能會有以下的結果：

```
[04:34 user@ws home] ./a.out
Please input your name: Jun Wu
Your name is Jun.
```

注意到了嗎□name字串的內容，其實是{'J','u','n','\0'}□為了解決此問題，可以改用gets()函式，它可以持續讀取使用者的輸入直到遇到換行為止，例如：

```c
#define LEN 10
char name[LEN + 1];

printf("Please input your name:");
gets(name);
printf("Your name is %s.\n", name);
```

其執行結果為：

```
[04:34 user@ws home] ./a.out
Please input your name:   Jun Wu
Your name is    Jun Wu
.
```

這裡又產了兩個新的問題：

1. 在輸入字串內容前的空白也視為字串內容，以及
2. 最後的換行(enter)也會被放在字串內。

另外，還有一個更嚴重的問題，因為gets()函式，會持續讀取直到遇到換行為止，因此，若使用者輸入內容過多，會造成系統用以讀取標準輸入的緩衝區(buffer)溢出(overflow)的問題，在執行時這會造成許多問題。因此，新版的C語言(C11)已經將gets()函式取消，或是你會在編譯時看到□warning: the `gets' function is dangerous and should not be used.□的警告。要解決這個問題，可以改用fgets()函式，其原型為:

```
char *fgets(char *str, int n, FILE *file);
```

此函式是設計用以從檔案file中，讀取不超過n個字元的字串，存放到str中。我們可以將第三個引數改成stdin(標準輸入管道)，就可以將其用在取得使用者來自標準輸入的字串了，請參考下面的程式：

getStrings.c

```
#include <stdio.h>

#define LEN 10

int main()
{
  char name[LEN+1];

  fgets(name, 10, stdin); //從stdin讀取不超過10個字元至name字串
  puts(name);

  return ;
}
```

但是這個方法還存在輸入字串內容前的空白與最後的換行字元的問題，我們在後續的章節中將說明如何設計函式以解決此問題。在此，我們先介紹另一種簡單但有效的方法，就是使用scanset來解決：

```
char str[LEN+1];
scanf("%[^\n]", str);
```

14.4.3 逐一讀入字元來取得字串內容

當然，你也可以配合迴圈，將字串以字元的方式逐一的輸入，請參考下面的程式：

readALine.c

```
#include <stdio.h>

int readALine(char str[], int n)
{
  int ch, i=;
  while((ch=getchar())!='\n')
  {
    if(i<n)
    {
      str[i++]=ch;
    }
  }
  str[i]='\0';
  return i;
}
```

```
int main()
{
  char str[20];

  readALine(str, 20);
  printf("%s\n", str);

  return ;
}
```

14.5 字串與函式呼叫

字串也可以做為函式設計時的引數或傳回值，以引數為例，下面的程式示範了傳入一個字串，計算並傳回其中包含空白字元的個數：

countSpace.c

```
#include <stdio.h>

int countSpace(const char s[])
{
  int count=, i;
  for(i=;s[i]!='\0';i++)
  {
    if(s[i]==' ')
    {
      count++;
    }
  }
  return count;
}

int main()
{
  char str[]="This is a test.";

  printf("There are %d space(s) in the string.\n", countSpace(str) );
  return ;
}
```

要注意在宣告函式時[]const char s[]表示該引數為一個字串。更明確來說，這個引數所傳入的是一個字串所在的記憶體位址，在main()函式中呼叫時，我們以countSpace(str)將str這個字串的位址傳入即可。其中的const保證了所傳入的值不可被更改。

我們也可以使用指標的方式，來設計相同的程式，請參考下例：

function2.c

```c
#include <stdio.h>

int countSpace(const char *s)
{
  int count=;

  printf("the argument s at %p.\n", s);

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

int main()
{
  char str[]="This is a test.";

  printf("%s at %p.\n", str, str);
  printf("There are %d space(s) in the string.\n", countSpace(str) );
  return ;
}
```

我們也可以讓字串做為函式的傳回值，但要注意的是，此情況下僅能以char *做為函式的傳回值，不能以char []做為函式傳回值：

getMessage.c

```c
#include <stdio.h>

char *getMessage(int i)
{
  if(i==)
    return "Welcome!";
  else
    return "Hello!";
}

int main()
{
  char *str;
  printf("%s\n", getMessage());

  str=getMessage(1);
```

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

14.6 字串處理函式

在C語言的函式庫中，提供了許多與字串操作相關的函式，其宣告位於string.h中。我們在本節列舉部份常用的字串處理函式:

- char *strcpy(char *s1, const char *s2);

將字串s2的內容複製到字串s1中，且複製完成的新字串也會傳回。

stringCopy.c

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

int main()
{
  char *str1, *str2;

  strcpy(str1, "abcd");

  printf("%s\n", str1);

  str2=strcpy(str1, "hello");

  printf("%s\n", str2);

  return ;
}
```

- char *strncpy(char *s1, const char *s2, size_t n); 將s2字串中前n個字元複製到s1中，其中size_t其實就是int型態。
- size_t strlen(const char *s); 傳回字串s的長度(不包含null character)。
- char *strcat(char *s1, const char *s2); 將字串s2的內容串接於s1之後。
- int strcmp(const char *s1, const char *s2); 比較字串s1與s2的內容，傳回值取決於s1與s2的內容：
  - 傳回0，若s1與s2相同
  - 傳回>0的值，若s1>s2
  - 傳回<0的值，若s1<s2

strcmp比較兩個字串的方法，就是一般字典在排列英文字的方法，我們稱之為Lexicographic order。若s1>s2，則：

- s1語s2前面i個字元相同，但s1的第i+1個字元小於s2。例如“abc”小於“abd”。或 “ abc”小於“bcd”。
- s1的內容與s2相同，但s1的長度小於s2。例如“abc”小於“abcd”。

至於每個字元的比較基準則是以ACSII編碼為依據：

- 數字小於字母
- 大寫字母小於小寫字母
- 空白小於字母與數字

關於更多的字串處理函式，可以參考其它相關書籍或網站，例如張凱慶先生的程式語言教學誌

另外，還有一些函式可用以將字串轉換為其它型態的數值，例如「atoi()函式」可以將字串轉成整數，

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

int main(void)
{
    printf("%d\n", atoi("123"));
    printf("%d\n", atoi("45.678"));
    printf("%d\n", atoi("abc"));
    printf("%d\n", atoi("5+2"));
    return ;
}
```

其輸出結果為：

```
123
45

5
```

除了「atoi()函式」外，C語言還提供了「atof()函式」與「atol()函式」等，用以將字串轉換為浮點數與長整數，這些都定義在「stdlib.h」中。

14.7 字串陣列

我們也可以設計一個陣列來存放多個字串，例如：

month.c

```
#include <stdio.h>

int main()
{
  int i;

  char month[][9] = { "January", "February", "March", "April",
                      "May", "June", "July", "August",
                      "September", "October", "November", "December"};
  for(i=;i<12;i++)
  {
    printf("%s\n", month[i]);
```

```
  }
  return ;
}
```

其實，這個程式有一個錯誤存在，請試著找出問題所在並加以更正。

我們也可以用指標的陣列來存放這些字串：

month2.c

```
#include <stdio.h>

int main()
{
  int i;

  char *month[] =    { "January", "February", "March", "April",
                       "May", "June", "July", "August",
                       "September", "October", "November", "December"};
  for(i=;i<12;i++)
  {
    printf("%s\n", month[i]);
  }
  return ;
}
```

figure 3與figure 4分別是這些字串以二維陣列與指標陣列儲存的記憶體示意圖：

	0	1	2	3	4	5	6	7	8	9
0	J	a	n	u	a	r	y	\0	\0	\0
1	F	e	b	r	u	a	r	y	\0	\0
2	M	a	r	c	h	\0	\0	\0	\0	\0
3	A	p	r	i	l	\0	\0	\0	\0	\0
4	M	a	y	\0	\0	\0	\0	\0	\0	\0
5	J	u	n	e	\0	\0	\0	\0	\0	\0
6	J	u	l	y	\0	\0	\0	\0	\0	\0
7	A	u	g	u	s	t	\0	\0	\0	\0
8	S	e	p	t	e	m	b	e	r	\0
9	O	c	t	o	b	e	r	\0	\0	\0
10	N	o	v	e	m	b	e	r	\0	\0
11	D	e	c	e	m	b	e	r	\0	\0

Fig. 3

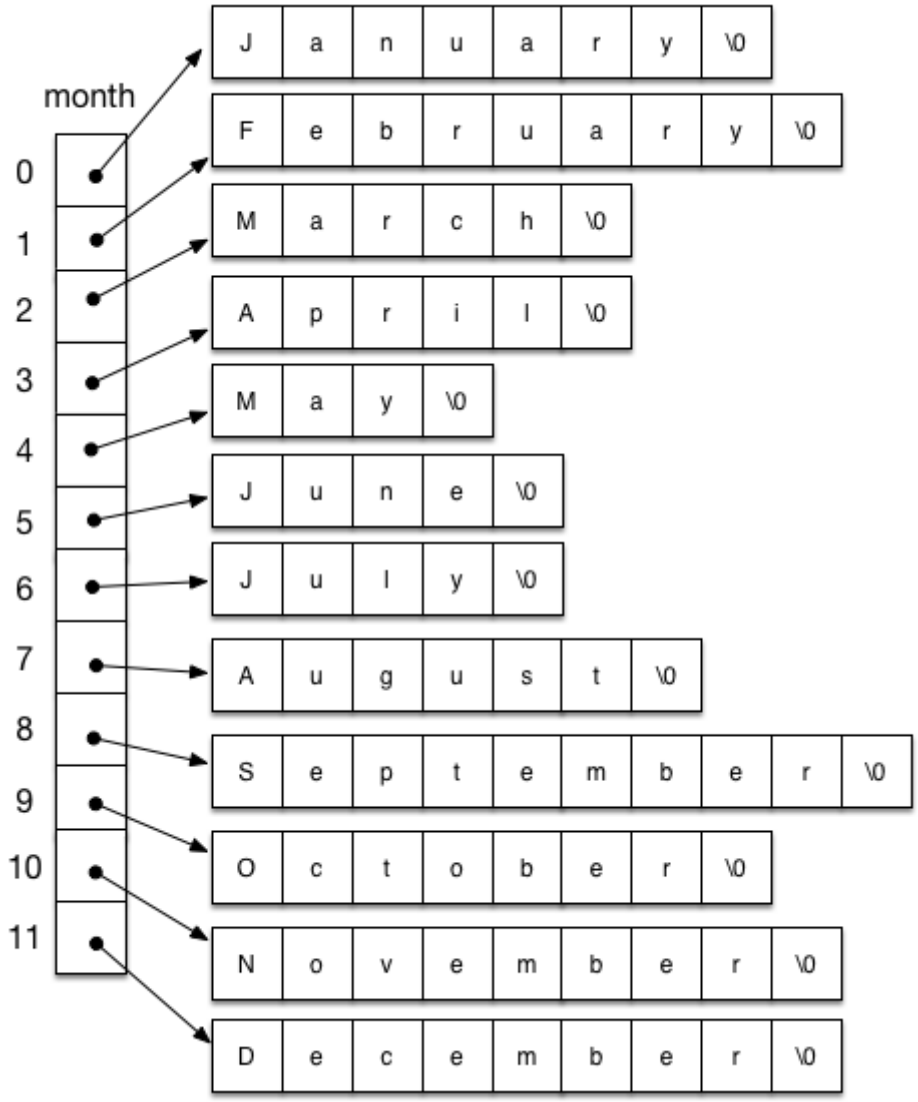

Fig. 4

14.8 命令列引數

所謂的命令列引數(command-line arguments)就是在main()函式中的引數。main()雖然是所謂的程式進入點，但它也是一個函式，也可以有引數。不過由於main()函式是由我們在作業系統的命令列透過執行程式而啟動，因此其引數就稱為命令列引數。例如，我們在linux系統中，可以執行下列的指令：

```
ls

ls -l

ls *.c -l
```

在上述例子中，ls為我們欲執行的指令(也就是一個可執行的程式)，-1，*.c等就是命令列引數。 我們自己所撰寫的程式，也可以讓main()函式接收這些引數，並進行後續的程式處理。若要使用命令列引數，main()函式的原型如下：

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

其中argc為引數的個數□argv為引數的指標陣列。請試著寫出一個程式，將其所接收的命令列引數輸出。