Computer Programming Using C
COP 3275 - Summer 2017

Lecture 19: Strings

By Ahmed E. Khaled
• Recap to the previous lecture:

  – Pointer declaration
  – Pointer operators
  – Pointer assignment
Although we’ve used variable char and created array of characters, however still lack any convenient way to process a series of characters (string).

This lecture covers both string constants (or literals) and string variables (that can change during the execution of a program).
String Literals

• A string literal is a sequence of characters enclosed within double quotes:

  "When you come to a fork in the road, take it."

• String literals may contain escape sequences (as those used in printf and Scanf).

• For example, each \n character in the string

  "Candy\nIs dandy\nBut liquor\nIs quicker.\n  --Ogden Nash\n"
• C compiler treats string literals as character arrays.

• When the compiler encounters a string literal of length $n$, it sets aside $n + 1$ bytes of memory to hold the characters of the string and one extra character (null character to mark end of the string).

• The null character is a byte whose bits are all zero, so it’s represented by the \0 escape sequence.
• Take the string literal “abc” as an example, the compiler creates character array of size 4.

```
 a b c \0
```

• An empty sting literal “” is stored as a single null character (character array of size 1)

```
 \0
```
• Since a string literal is stored as a character array, the compiler treats it as a *pointer of type char*.

• Both `printf` and `scanf` expect a value of type `char *` as their first argument.

• The following call of `printf` passes the address of "abc" (a pointer to where the letter a is stored in memory).
Operations on String Literals

```c
char *p;
p = "abc";
/* p points to the first character of the string*/.

char ch;
ch = "abc"[1];
/* array subscription, where the new value of ch will be the letter b */
```
• Attempting to modify a string literal causes undefined behavior (it is possible, but not recommended):

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

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

• A program that tries to change a string literal may crash or behave erratically (with some compilers).
String Variables

• Any one-dimensional array of characters can be used to store a string and must be terminated by a null character (\0).

• Difficulties with this approach:
  – It can be hard to tell whether an array of characters is being used as a string.
  – String-handling functions must be careful to deal properly with the null character.
  – Finding the length of a string requires searching for the null character.
• If a string variable needs to hold 80 characters, it must be declared to have length 81:

```c
int STR_LEN = 80;
char str[STR_LEN+1];
```

• Adding 1 to the desired length allows room for the null character at the end of the string.

• Thus str can store strings of **no more than** 80 characters.

• However, that doesn’t mean that it will always contain a string of 80 characters, where the length of the string depends on the position of the terminating null character.
• A string variable can be initialized at the same time it’s declared:

```
char date1[8] = "June 14";
```

• The compiler will automatically add a null character so that `date1` can be used as a string:

```
date1  J u n e  l 4  \0
```

• "June 14" is not a string literal in this context, where the compiler views it as an abbreviation for an array initializer.

• We could have written:

```
char date1[8]= { 'J', 'u',..., '4', '\0' }; 
```
• If the initializer is too short to fill the string variable, the compiler adds extra null characters (as default value for initializing array)

char date2[9] = "June 14";

Appearance of date2:

```
   date2  J  u  n  e   1  4  \0  \0
```
• The declaration of a string variable may omit its length, in which case the compiler computes it:
  char date4[] = "June 14";

• The compiler sets aside eight characters for date4, enough to store the characters in "June  14" plus a null character.

• Omitting the length of a string variable is especially useful if the initializer is long, since computing the length by hand is error-prone.
#include <stdio.h>
#include <string.h>

main () {
    char str1[12] = "Hello";  
    char str2[12] = "World";
    char str3[12];
    int len;

    /* copy str1 into str3 */
    strcpy(str3, str1);
    printf("strcpy( str3, str1) : %s\n", str3);

    /* concatenates str1 and str2 */
    strcat(str1, str2);
    printf("strcat( str1, str2) : %s\n", str1);

    /* total length of str1 after concatenation */
    len = strlen(str1);
    printf("strlen(str1) : %d\n", len);
}
# Important functions for strings

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

void main () {
    char str1[15]; char str2[15];
    int ret;

    strcpy(str1, "abcdef"); strcpy(str2, "ABCDEF");

    ret = strcmp(str1, str2); 

    if(ret != 0) {
        printf("str1 is not equal to str2"); 
    } else {
        printf("str1 is equal to str2"); 
    }
}
```