Strings

Prepared by: RNC Recario
rncrecario@gmail.com
Institute of Computer Science UPLB
Jan 2012

OBJECTIVES

At the end of the laboratory session, the student is expected to

- Define what a string is.
- Perform manipulation of strings.
- Use string functions available using `<string.h>`.

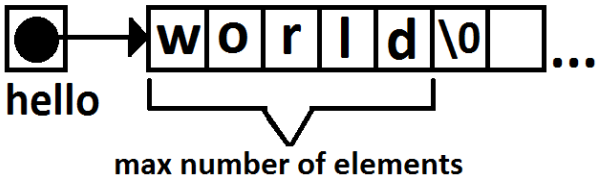
Strings

So far we were able to discuss arrays and multidimensional arrays in general but have not touched one of the most important datatypes that exist – strings. A string, by definition, is an array (more specifically a one dimensional array) or collection of characters. In some Programming Languages (PLs), string is considered to be a primitive or basic datatype while in some, it is not. In C, strings are typically viewed as a non-primitive datatype.

Because a string is an array, the same ideas with the general array apply with strings. However, there is a difference. A string is terminated by a null character represented by the symbol '`\0`'. Consider the following code below.

```
01 #include<stdio.h>
02
03 int main(){
04    char hello[5];
05
06    printf("Enter a string: ");
07    scanf("%s", hello);
08
09    printf("The string is %s.\n", hello);
10
11    return 0;
12 }
```

The string variable hello can accept 10 characters. Now suppose we ran this code and put a value "world" for the variable hello. The word "world" will be put into the hello variable. The "world" input will consume 5 elements plus the '`\0`' – a total of 6. Note also that the format specifier for the string is %s. More interestingly, look at line number 7. Observe that `hello` is used without the `&` (as in `&hello`). This is because `hello` already holds the memory address of the first element (remember: it is an array of characters. And since it is an array, `hello` serves as an integer pointer).



Try running the program you created (using the above code) with inputs having spaces. Also try inputs with more characters (exceeding or equal to 10). What happens?

Fortunately for us, it is already easy to perform computations and operations involving strings. The C programming language provides a library for string manipulations. The library is <string.h>. The following are some of the functions provided by the <string.h> library.

`char *strcpy( char *s1, const char *s2)`
–copies the string s2 into the character array s1. The value of s1 is returned.

`char *strncpy( char *s1, const char *s2, size_t n)`
–copies at most n characters of the string s2 into the character array s1. The value of s1 is returned.

`char *strcat( char *s1, const char *s2)`
–appends the string s2 to the end of character array s1. The first character from s2 overwrites the '\0' of s1. The value of s1 is returned.

`char *strncat( char *s1, const char *s2, size_t n)`
–appends at most n characters of the string s2 to the end of character array s1. The first character from s2 overwrites the '\0' of s1. The value of s1 is returned.

`char *strchr( const char *s, int c)`
–returns a pointer to the first instance of c in s. Returns a NULL pointer if c is not encountered in the string.

`char *strrchr( const char *s, int c)`
–returns a pointer to the last instance of c in s. Returns a NULL pointer if c is not encountered in the string.

`int strcmp( const char *s1, const char *s2)`
–compares the string s1 to the string s2. The function returns 0 if they are the same, a number < 0 if s1 < s2, a number > 0 if s1 > s2.

`int strncmp( const char *s1, const char *s2, size_t n)`
–compares up to n characters of the string s1 to the string s2. The function returns 0 if they are the same, a number < 0 ifs1 < s2, a number > 0 if s1 > s2.

`size_t strspn( char *s1, const char *s2)`
–returns the length of the longest substring of s1 that begins at the start of s1and consists only of the characters found in s2.

`size_t strcspn( char *s1, const char *s2)`
–returns the length of the longest substring of s1 that begins at the start of s1and contains none of the characters found in s2.

`size_t strlen( const char *s)`
–determines the length of the string s. Returns the number of characters in the string before the '\0'.

`char *strpbrk( const char *s1,  const char *s2)`
–returns a pointer to the first instance in s1 of any character found in s2. Returns a NULL pointer if no characters from s2 are encountered in s1.

`char *strstr( const char *s1,  const char *s2)`
–returns a pointer to the first instance of string s2 in s1. Returns a NULL pointer if s2 is not encountered in s1.

`char *strtok(char *s1, const char *s2)`
–repeated calls to this function break string s1 into "tokens"--that is the string is broken into substrings, each terminating with a '\0', where the '\0' replaces any characters contained in string s2. The first call uses the string to be tokenized as s1; subsequent calls use NULL as the first argument. A pointer to the beginning of the current token is returned; NULL is returned if there are no more tokens.

Let us use some of the functions to give you an idea of how they are used. Suppose that we want to determine the length or number of characters that you have typed for let us say variable hello1. The following code accomplishes the said task.

```
01   #include<stdio.h>
02   #include<string.h>
03   #define stringsize 20
04
05   int main(){
06
07      char hello1[stringsize];
08
09      printf("Enter a string: ");
10      scanf("%s", hello1);
11
12      printf("The string is %s.\n", hello1);
13      printf("The string length is %d.\n", strlen(hello1));
14   }
```

Copying the contents of an array is also easy thanks to `strcpy`. The following code shows an example of how this is done.

```
01   #include<stdio.h>
02   #include<string.h>
03
04   int main(){
05      char hello1[5], hello2[5];
06
07      printf("Enter a string: ");
08      scanf("%s", hello1);
09      //copy the contents to hello2
10      strcpy(hello2, hello1);
11      printf("The strings are %s and %s.\n", hello1, hello2);
12   }
```

Parameter Passing with Strings

Parameter passing with strings is easy because the parameter passing is already pass by reference. This is possible since the variable declared is already a pointer (as in the case of hello1 from the previous code). An example of parameter passing with strings is shown below.

```
01 #include<stdio.h>
02 #include<string.h>
03
04 void getinput(char * str1, char * str2){
05      printf("Enter a string: ");
06      scanf("%s", str1);
07      printf("Enter a string: ");
08      scanf("%s", str2);
09 }
10
11 void printinput(char * str1, char * str2){
12      printf("The string is %s.\n", str1);
13      printf("The string is %s.\n", str2);
14 }
15
16 main(){
17     char hello1[10], hello2[10];
18
19      getinput(hello1, hello2);
20      printinput(hello1, hello2);
21 }
```

Arrays of Strings

It is also possible to create an array of strings. This can be viewed as a 2D array of characters when a string is viewed from the perspective of being a 1D array of characters.

```
01 #include<stdio.h>
02 #include<string.h>
03
04 void getnames(char str[5][10]){
05   int i;
06   for(i=0; i<5; i++){
07         printf("Enter name: ");
08         scanf("%s", &str[i]);
09   }
10 }
11
12 void printnames(char str[5][10]){
13   int i;
14   for(i=0; i<5; i++){
15         printf("%s\n", str[i]);
16   }
```

```
17 }
18
19
20 main(){
21    char names[5][10];
22
23    getnames(names);
24    printnames(names);
25 }
```

Observe however that each element (each name) is accessed with the `&`. Although the address of the array of names was passed, each element of the array needs individual access hence the `&`.

References:

http://www.edcc.edu/faculty/paul.bladek/c_string_functions.htm

Exercise 6: Strings

Prepared by: RNC Recario
rncrecario@gmail.com
Institute of Computer Science UPLB
Jan 2012

Download the necessary file `u1l_exer6_recario.c` from our Google site. The exercise is quite easy: Get 2 input names from the user. After getting the input names, determine the number of common characters between the two names REGARDLESS of case (uppercase A is the same with lowercase a). Spaces are not consider part of the names as well as the suffix Jr, Sr, II, III, etc. It is your choice whether to include middle names but at the minimum, the names have the first and last names. Spaces are required to part of the input!

After getting the input, we are now ready to compute FLAMES. The number of characters common between the two names determines the result of the FLAMES. An Example is shown below:

```
Enter name 1: Juan Dela Cruz
Enter name2: Rosalinda Sy

Calculating …

>>>FLAMES
+--------------------------------+
Juan Dela Cruz |  6  | Sweetheart|
Rosalinda Sy   |  6  | Sweetheart|
+--------------------------------+
|Total         | 12  | Sweetheart|
+--------------------------------+
Thanks for using FLAMES! :-)
```

If the result is greater than six (6), the counting starts back to letter F down to S and so on and so forth. You may choose to imitate the interface shown above but I suggest you create your own and be creative!

Enjoy! ☺