CptS 121 - Program Design and Development

Lab 9: Strings

Assigned: Week of June 5th, 2019
Due: At the end of the lab session

I. Learner Objectives:

At the conclusion of this programming assignment, participants should be able to:

- Declare strings in C
- Apply library functions found in <string.h>
- Distinguish between character arrays and strings in C
- Implement array notation or pointer arithmetic to manipulate strings
- Compare strings
- Copy strings

II. Prerequisites:

Before starting this programming assignment, participants should be able to:

- Utilize output parameters and pointers in a C program
- Apply the dereference or indirection C operator
- Declare array in C
- Apply arrays in C to various problems
- Pass arrays into functions
- Initialize arrays using an initializer list
- Construct loops to traverse through arrays
- Compose iterative statements (“while”, “for”, and/or “do-while” statements)
- Compose decision statements (“if” conditional statements)
- Apply top-down design

III. Overview & Requirements:

This lab, along with your TA, will help you navigate through applying strings in C. Recall arrays are based on the premise of contiguously allocated blocks of memory. Strings are also arrays in which each character composing it is contiguous in memory. All strings in C must be terminated by the null character, i.e. '\0'. If the null character is missing at the end of the character array, then a valid C string has not been created. Thus, standard string library functions and programmer defined functions will not produce predictable results when manipulating the character array.

Labs are held in a “closed” environment such that you may ask your TA questions. Please use your TAs knowledge to your advantage. You are required to move at the pace set forth by your TA. Please help other students in need when you are finished with a task. You may work in pairs if you wish. However, I encourage you to compose your own solution to each problem. Have a great time! Labs are a vital part to your education in CptS 121 so work diligently.

Tasks:

1. (a) Write a program that reverses strings (Jon Campbell from Microsoft indicated this could be a possible interview question). You need to define a function called string_reverse () that accepts a string as an argument and reverses the string without creating an extra array to perform the reversing, i.e. "CptS 121 is cool!" becomes "!looc si 121 StpC". This is a form of in place reversal. The function should return a pointer to the reversed string. Implement a solution to this function using array notation only. NOTE: you may NOT use any <string.h> library functions to solve this problem!

(b) Rewrite the string_reverse() function using pointer notation and pointer arithmetic only.

2. Write a program that tests the following functions. Yes, you must implement the below functions without using any <string.h> library functions. You may use array and/or pointer notation to define the functions.

- char *my_strcpy (char *destination, const char *source) - Copies all characters in the array pointed to by source into the array pointed to by destination. The null character is also copied. The function returns destination.

- char *my_strcat (char *destination, const char *source) - This function appends a copy of the string pointed to by source (including the null character) to the end of the string pointed to by destination. The append overwrites the null character at the end of destination. The string pointed to by destination is returned.

- int my_strcmp (const char *s1, const char *s2) - This function compares the string pointed to by s1 to the string pointed to by s2. If the string pointed to by s1 comes before the string pointed to by s2 in dictionary ordering, then -1 is returned. If the string pointed to by s1 is the same as the string pointed to by s2, then 0 is returned (the compare function is case sensitive). Otherwise 1 is returned.

- int my_strlen (const char *s) - This function returns the length of the string pointed to by s. The computation of length does NOT include the null character.

3. Recall that bubble sort is a technique for sorting an array. A bubble sort compares adjacent array elements and exchanges their values if they are out of order. In this way, the smaller values "bubble" to the top of the array (toward element 0), while the larger values sink to the bottom of the array. After the first pass of a bubble sort, the last array element is in the correct position; after the second pass the last two elements are correct, and so on. Thus, after each pass, the unsorted portion of the array contains one less element. Write and test a function that implements this sorting method on strings! You MAY use <string.h> library functions to help with sorting the strings. NOTE: you will need to run your sort on an array of strings. Any array of strings is viewed as a 2-dimensional array of characters in which each row is terminated by the null character. Thus, each row represents one string in the array of strings. Consider the below information.

char array_strings[10][50]; /* Declare an array of strings which contains at most 10 strings (the number of rows) which have a length of at most 50 characters (the number of columns) */

/* To copy a string into one row of the 2-D array use string copy as follows. */
strcpy (array_strings[0], "a string"); /* Note: supplying one dimension as an index into a 2-D array supplies a starting address for a row! */
strcpy (array_strings[1], "cat");

/* Logical view */

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**...**	**49**
0	'a'	' '	's'	't'	'r'	'i'	'n'	'g'	'\0'	...	?
1	'c'	'a'	't'	'\0'	?	?	?	?	?	...	?
...	...	...	...	...	...	...	...	...	...	...	...
9	?	?	?	?	?	?	?	?	?	...	?

IV. Grading Guidelines:

- This lab is worth 10 points. Your lab grade is assigned based on completeness and effort. To receive full credit for the lab you must show up on time and continue to work on the problems until the TA has dismissed you.