**Lab 5: C Strings and String Functions**

**Goals**

* Understand operations on C strings.
* Become more familiar with pointers.
* Build a program that consists of multiple source modules

**Preparing for today's lab:**

In your cpsc1070 directory, create the subdirectory **lab5** and **cd** into it.

Use the following command to copy the files for this lab:

lab1071copy 5

One of the files that is now in your lab5 directory is is mystrings.c. The module contains a do-nothing skeleton of the 3 string functions that are described below. Use an editor to modify the code in mystrings.c to carry out the string operations as described below.

A second file in your lab5 directory is lab5test.c. Do not modify this module. The code in this module contains the test code for this lab and makes calls to the procedures you will develop in mystrings.c. As we will see throughout the semester it is common to place procedures into separate .c modules. This keeps the resulting program more manageable and promotes the reuse of procedures in other programs.

**Testing**

* Do NOT put a main function in mystrings.c. It will break the tester.
* Compile lab5test.c and mystrings.c together:

gcc -o lab5 -Wall -g lab5test.c mystrings.c

**C String Functions**

A C string is null-terminated array of characters. (That means that after the last character in the string, there is a binary zero ('\0').) To use the regular C string library functions, you should #include <string.h>.  By convention C string library functions begin with "str."  To see a complete inventory of the string functions click [here](https://www.cs.clemson.edu/course/cpsc1070/labs/lab05/string.txt).   
  
In mystrings.c you will *not*be using any of the functions from the C library. Instead you will write three functions that work like their library counterparts.   The [zstrcpy()](https://www.cs.clemson.edu/course/cpsc1070/labs/lab05/zstrcpy.c) function can be used as a model. 

For this lab you will write three (3) string functions in the mystrings.c module. Your code

* may NOT use array notation (i.e. you may not use any square brackets)
* may NOT use any functions you did not write (e.g. strcpy(), strcmp(), toupper(), etc.),
* may NOT use any memory allocation functions (don't worry if you don't know what this is),

Here are the three functions:

* + **int mystrlen(char \*s1);**   
      
    This function should return the length of the string pointed to by s1.  The length does not include the terminating 0 byte. For example, the a string declared as char \*s = "Hello";  has length 5.
  + **char \*mystrupper(char \*s1);**  
      
    The function mystrlower will convert any character in s1 that is an lower case letter (i.e. a-z) to upper case. Characters that are not lower case letters are left unchanged. The return value is the original value of s1 (i.e. the pointer that is pointing to the modified string).

For example, if the original string is "char str1[]="HeLlO";", the call

mystrupper(str1);

would convert the contents of the string str1 to "HELLO".

Remember that characters are represented in memory by their ASCII code. If you enter the command "man ascii" note that the decimal code value of an 'A' is 65, 'B' is 66, 'C' is 67, ..., 'Z' is 90. Likewise note that the decimal code value for 'a' is 97, 'b' is 98, 'c' is 99, ... 'z' is 122.

Here is a quick table that shows the printable characters and their decimal equivalent:

30 40 50 60 70 80 90 100 110 120

---------------------------------

0: ( 2 < F P Z d n x

1: ) 3 = G Q [ e o y

2: \* 4 > H R \ f p z

3: ! + 5 ? I S ] g q {

4: " , 6 @ J T ^ h r |

5: # - 7 A K U \_ i s }

6: $ . 8 B L V ` j t ~

7: % / 9 C M W a k u DEL

8: & 0 : D N X b l v

9: ' 1 ; E O Y c m w

You can read this table by looking at the position of the character in the table and noting its row and column value. For example consider the character 'i'. Read the number at the top of the column, in this case 100, and then the number to the left for the row, in this case 5. Add the two. So the decimal code for 'i' is 100+5=105.

So how might we detect if a character is a lower case letter? How might we convert it to its upper case equivalent? Note: you DO NOT want a massive if-then-else -- this can be done with simple arithmetic. You CAN DO arithmetic on character data items in C, e.g. you can add and subtract integer values to/from a character data item. Hint: note that the upper and lower case code for a letter differs by 32.

There is a function available in C that will convert a lower case character to upper case -- it is called toupper(). For the sake of this exercise you MAY NOT use toupper() or any other function that you did not write yourself in your solution.

* + **int mystrcmp(char \*s1, char \*s2);**  
      
    The function compares successive pairs of elements from two strings  until it finds elements that are not equal (i.e this is equivalent to the standard C strcmp() function).   
      
    If all elements are equal, the function returns zero. If the differing element from s1 is greater than the element from s2, the function returns a positive value. Otherwise, the function returns a negative value. Note: the result of the comparison is denoted by the "sign" of the returned value -- the actual value is immaterial. For example, a return value of -1 or -35 from strcmp() means the same thing -- the first string is considered to be "less than" the second string. In your solution to mystrcmp() you may return -1, 0 and 1 if you wish; or you any other values that are consistent with the meaning of the sign of the return value.

*Short note on the meaning of string comparisons:* as we have seen, a C string is just an array of characters. As noted above with strupper() internally in memory each character is represented by an 8-bit number.

Note in the table above that "a" is represented by the numerical value "97" and "b" is represented by "98", so "a" is less than "b". Note that capital "A" is represented by "65". So from this we would say (perhaps illogically) that "a" is GREATER THAN "A", i.e. "a"'s numerical representation is greater than capital "A"'s.

**Requirements**

Reminder: you can ONLY use pointer notation in your solution, you may not use array notation, i.e. no square [] brackets.

**Submitting the lab**

Use [handin](https://handin.cs.clemson.edu/) to submit your mystrings.c procedure.

It is your responsibility to make sure that you have correctly and sucessfully completed the submission.