CIS 278 (CS1) Programming Methods: C++

Assignment 7: Strings

Skip to Main Content

Learning Objectives

After the successful completion of this learning unit, you will be able to:

- Implement syntactically correct strings and c-strings.

Introduction

This week we take a slight detour to discuss the two different ways that string data can be handled in C++ programs.

1. The original C-String technique, developed in the late 1960's for the C language
2. The object oriented string object technique. Developed in the mid 1980's

It is important to be comfortable with both, as they are both in wide use.

Assignment 7.1 [45 points]

No initial file comment is required for this assignment. Function comments are still required.

Implement the following functions. Each function deals with null terminated C-Style strings. You can assume that any char array passed into the functions will contain null terminated data. Place all of the functions in a single file and then (in the same file) create a main() function that tests the functions thoroughly. You will lose points if you don't show enough examples to convince me that your function works in all cases.

Please note the following:

1. You may not use any variables of type string. This means that you should not #include <string>. Also, you may not use any c-string functions other than strlen(). If you use any other c-string functions, you will not get credit. Note, however, that functions such as toupper(), tolower(), isalpha(), and isspace() are NOT c-string functions, so you can use them. Also note that this prohibition is only for the functions that you are assigned to write. You can use whatever you want in your main() function that tests them.

2. In most cases it will be better to use a while loop that keeps going until it hits a '\0', rather than using a for loop that uses strlen() as the limit, because calling strlen() requires a traversal of the entire array. You could lose a point or two if you traverse the array unnecessarily.

3. None of these function specifications say anything at all about input or output. **None of these functions should have any input or output statements in them.** The output should be done in the calling function, which will probably be main(). The only requirement about main() is that it sufficiently test your functions. So, you can get user input in main() to use as arguments in the function calls, or you can use hard-coded values -- up to you, as long as the functions are tested thoroughly.

4. Here's a hint about how to work with c-strings in main(). There are several different ways that you could assign values to c-string variables, but I think the easiest is just hardcoding a lot of examples. For example:

```
char str1[] = "Hello World";
char str2[] = "C++ is fun!";
```

Whatever you do, don't try to create and initialize a c-string on one line using pointer notation, like this:

```
char* str1 = "Hello world";
```

This is dangerous (and officially deprecated in the C++ standard) because you haven't allocated memory for str1 to point at.

5. Since it is just being used for testing, In this case it is fine to have a very long main() function if it is just a sequence of statements without any loops.

Here are the functions:

1. This function finds the last index where the target char can be found in the string. it returns -1 if the target char does not appear in the string. The function should be case sensitive (so 'b' is not a match for 'B').

   ```
   int lastIndexOf(const char* inString, char target)
   ```

2. This function alters any string that is passed in. It should reverse the string. If "flower" gets passed in it should be reversed ***in place*** to "rewolf". For efficiency, this must be done "in place", i.e., without creating a second array.

   ```
   void reverse(char* inString)
   ```

3. This function finds all instances of the char 'target' in the string and replace them with 'replacementChar'. It returns the number of replacements that it makes. If the target char does not appear in the string it should return 0.

   ```
   int replace(char* inString, char target, char replacementChar)
   ```

4. This function returns true if the argument string is a palindrome. It returns false if it is no. A palindrome is a string that is spelled the same as its reverse. For example "abba" is a palindrome. So are "hannah" and "abc cba".

   Do not get confused by white space characters, punctuation, or digits. They should not get any special treatment. "abc ba" is not a palindrome. It is not identical to its reverse.

   Your function should not be case sensitive. For example, "aBbA" is a palindrome.

   You must solve this problem "in place", i.e., without creating a second array. As a result, calling your reverse() function from this function isn't going to help.

   ```
   bool isPalindrome(const char* inString)
   ```

5. This function converts the c-string parameter to all uppercase.

   ```
   void toupper(char* inString)
   ```

6. This function returns the number of letters in the c-string.

   ```
   int numLetters(const char* inString)
   ```

Submit Your Work

Name your source code file according to the assignment number (a1_1.cpp, a4_2.cpp, etc.). Execute the program and copy/paste the output that is produced by your program into the bottom of the source code file, making it into a comment. Use the Assignment Submission link to submit the source file. When you submit your assignment there will be a text field in which you can add a note to me (called a "comment", but don't confuse it with a C++ comment). In this "comments" section of the submission page let me know whether the program works as required.

Keep in mind that if your code does not compile you will receive a 0.