CS 11 Data Structures and Algorithms

Assignment 2: Characters, Strings, Structs

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.

Assignment 2.1 [15 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 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.

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. 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)
   ```

Assignment 2.2 [15 points]

Note: Do not use classes or any variables of type string to complete this assignment

Write a program that reads in a sequence of characters entered by the user and terminated by a period ('.'). Your program should allow the user to enter multiple lines of input by pressing the enter key at the end of each line. The program should print out a frequency table, sorted in decreasing order by number of occurences, listing each letter that ocurred along with the number of times it occured. All non-alphabetic characters must be ignored. Any characters entered after the period ('.') should be left in the input stream unprocessed. No limit may be placed on the length of the input.

Use an array with a struct type as its base type so that each array element can hold both a letter and an integer. (In other words, use an array whose elements are structs with 2 fields.) The integer in each of these structs will be a count of the number of times the letter occurs in the user's input.

Let me say this another way in case this makes more sense. You will need to declare a struct with two fields, an int and a char. (This should be declared above main, so that it is global; that is, so it can be used from anywhere in your program.) Then you need to declare an array (not global!) whose elements are those structs. The int in each struct will represent the number of times that the char in the same struct has occurred in the input. This means the count of each int should start at 0 and each time you see a letter that matches the char field, you increment the int field.

Don't forget that limiting the length of the input is prohibited. If you understand the above paragraph you'll understand why it is not necessary to limit the length of the input.

The table should not be case sensitive -- for example, lower case 'a' and upper case 'A' should be counted as the same letter. Here is a sample run:

```
Enter a sequence of characters (end with '.'): do be Do bo. xyz

Letter:    Number of Occurrences
     o     3
     d     2
     b     2
     e     1
```

Note: Your program must sort the array by descending number of occurrences. You may use any sort algorithm, but I would recommend using the selection sort from **lesson 9.6**. Be sure that you don't just sort the output. The array itself needs to be sorted. Don't use C++'s sort algorithm.

Submit your source code and some output to show that your code works.

Hints:

You will want to be familiar with various C++ functions for dealing with characters such as isupper, islower, isalpha, toupper, and tolower. You should assume for this assignment that the ASCII character set is being used. (This will simplify your code somewhat.)

Note that your struct should be declared above main(), and also above your prototypes, so that your parameter lists can include variables that use the struct as their base type. It should be declared below any global consts.

Students seem to get hung up on the requirement that the user can enter multiple lines of input and that you should leave characters after the period in the input stream. The intention of these requirements (believe it or not) is to steer you toward a better and simpler solution, so please don't get too hung up on them! Another way of stating the requirement is this: just keep reading characters one at a time, ignoring newline characters (that is, treating them just like any other non-alphabetic character), until you get to a period.

More hints about how this will work: The user enters a bunch of characters then hits the enter button. When the user hits the enter button, those characters go into the input stream. THEN cin.get(ch) reads the first character from the input stream, then it is counted, then the second time through the loop the second character is read from the input stream and counted, then the third time through the loop the third character is read from the input stream and counted, and so on until the loop condition becomes false or all of the characters that the user entered have been read.

At that point, if all of the characters have been read, then the cin.get(ch) causes the program to stop and wait for more characters to be entered.

If the loop has exited, then any remaining characters that the user entered will stay in the input stream, so that next time you do a cin.get(ch) or cin >> ch, those characters will be read.

One last way of saying this: If the user enters several characters, but does not enter '.', and then hits the enter key, the loop should process all the characters entered, and then just wait for more characters, until it finally sees a '.'

Assignment 2.3 [15 points]

Rewrite your most recent high scores program so that each name/score pair is stored in a struct named highscore. Except as noted below, this new program will continue to meet all of the requirements of your most recent high scores program. Your new program should meet the following requirements:

1. The highscore struct should have two fields:
    - an int named score
    - and a char array named name. The char array should have 24 elements, making the maximum length of the name 23. (If you prefer to use a char pointer and a dynamically allocated array, that is fine as well. However, this may result in a number of complications, so be prepared for the challenge.)
2. The data should be stored in a single array, a dynamically allocated array of highscore structs.
3. Your program should use three functions that accept the array of highscore structs:

```
void initializeData(highscore scores[], int size)
void sortData(highscore scores[], int size)
void displayData(const highscore scores[], int size)
```

4. You may use any sort algorithm, but I would recommend using the selection sort from **lesson 9.6**. Don't use C++'s sort() function, but you can use the swap() function.
5. Note that when you swap your array elements, you can swap the entire struct. You don't need to swap the name and the score separately.
6. You may assume that the user enters names that are 23 characters or less. Getting this to work correctly if the user enters names that are too long -- that is, making it so that you put the first 23

characters in the name variable and ignore the remaining characters on the line -- is complicated. You can do this as an extra challenge if you want, but it's not required.

Submit Your Work

Name your source code file(s) according to the assignment number (a1_1.cpp, a4_2.cpp, etc.). Execute each program and copy/paste the output into the bottom of the corresponding source code file, making it into a comment. Use the Assignment Submission link to submit the source file(s). 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(s) work as required.