Introduction to Computer Science I

Summer Sessions 2021 - COM SCI31-1 - STAHL

Project 4

Programming Assignment 4
String Arrays

Time due: 9:00 PM Monday, July 26th

Introduction

As you gain experience with arrays, you'll discover that many applications do the same kinds of things with them (e.g., find where an item is in an array, or check whether two arrays differ). You'll find that it's helpful to have a library of useful functions that manipulate arrays. (For our purposes now, a library is a collection of functions that developers can call instead of having to write them themselves. For a library to be most useful, the functions in it should be related and organized around a central theme. For example, a screen graphics library might have functions that allow you to draw shapes like lines and circles on the screen, move them around, fill them with color, etc. In this view, the Standard C++ library is really a collection of libraries: a string library, a math library, an input/output library, and much more.)

Your assignment is to produce a library that provides functions for many common manipulations of arrays of strings. For each function you must write, this specification will tell you its interface (what paramet ers it takes, what it returns, and *what* it must do). It's up to you to decide on the implementation (*how* it will do it).

The source file you turn in will contain all the functions and a main routine. You can have the main routine do whatever you want, because we will rename it to something harmless, never call it, and append our own main routine to your file. Our main routine will thoroughly test your functions. You'll probably want your main routine to do the same. If you wish, you may write functions in addition to those required here. We will not directly call any such additional functions. If you wish, your implementation of a function required here may call other functions required here.

The program you turn in must build successfully, and during execution, no function (other than main) may read anything from cin or write anything to cout. If you want to print things out for debugging purposes, write to cerr instead of cout. When we test your program, we will cause everything written to cerr to be discarded instead — we will never see that output, so you may leave those debugging output statements in your program if you wish.

All of the functions you must write take at least two parameters: an array of strings, and the number of items the function will consider in the array, starting from the beginning. For example, in

```
string folks[8] = {  "samwell", "jon", "margaery", "daenerys",
                     "tyrion", "sansa", "llewmas ", "noj" };
int max = locateMaximum( folks, 5 ); // should return 4 and not inspect the last three elements....
```

even though the array has 8 elements, only the first 5 had values we were interested in for this call to the function; the function must not examine any of the others.

The one error your function implementations don't have to handle (because they cannot) is when the caller of the function lies and says the array is bigger than it really is. For example, in this situation. the function can't possibly know the caller is lying about the number of items in the array:

```
string values[5] = { "jon", "mamabbcc!", "jon", "123", "45" };
int max = locateMaximum( folks, 444444) ;     // Bad driver code call
                                              // your implementation doesn't have to check for this, because
it can't
```

To make your life easier, whenever this specification talks about strings being equal or about one string being less than or greater than another, the case of letters matters. This means that you can simply use comparison operators like == or < to compare strings. Because of the character collating sequence, all upper case letters come before all lower case letters, so don't be surprised by that result. The FAQ has a note about string comparisons.

Your task

Here are the functions you must implement:

`int locateMaximum( const string array[ ], int n );`
Return the index of the largest item found in the passed array or -1 if n <= 0. For example, for the array `folks[ 5 ]` shown above, `locateMaximum( folks, 4 )` should return the value 0, corresponding to the index of `"samwell"`. If there are multiple duplicate minimum values, return the smallest index that has this largest value. The largest value is determined by its dictionary-sorted order which is what < and > use in C++ to determine `true` and `false`.

`bool hasDuplicatedValues( const string array[ ], int n );`
If two or more elements of the passed array are identical, return `true` otherwise return `false` or if n <= 0 return `false`. For example, for the array `folks[ 8 ]` shown above, `hasDuplicatedValues( folks, 8 )` should return `false`. For the array `values[ 5 ]` shown above, `hasDuplicatedValues( values, 5 )` should return `true`.

`int countAllVowels( const string array[ ], int n );`
Return the number of vowels found in each element of the passed array or if n <= 0 return `-1`. For example, for the array `folks[ 5 ]` shown above, `countAllVowels( folks, 5 )` should return `14`. For example, for the array `folks[ 8 ]` shown above, `countAllVowels( folks, 8 )` should return `19`. For the purposes of this function, the characters 'a', 'e', 'i', 'o', 'u', 'y' and 'A', 'E', 'I', 'O', 'U' and 'Y' should be considered vowels.

`int moveToEnd( string array[ ], int n, int position );`
Adjust the passed array, moving the element found in the index value `position` to the end of the array (index value `n-1`) and moving the element found at the end of the array (index value `n-1`) to the index value `position`, returning the value of `position`. If n <= 0 or position is < 0 or position >= n, return `-1` and do not adjust the passed array at all. For example, for the array `folks[ 5 ]` shown above, `moveToEnd( folks, 5, 1 )` should return 1 and have folks[1] = "tyrion" and folks[4] = "jon".

`int countIntegers( const string array[ ], int n );`
Return the number of integer values found in the passed array or if n <= 0 return `-1`. For example, for the array `values[ 5 ]` shown above, `countIntegers( values, 5 )` should return `2`. For example, for the array `folks[ 5 ]` shown above, `countIntegers( folks, 5 )` should return `0`. For the purposes of this function, an integer should be made up of solely digit characters with no leading '+', '-' or decimal point.

`int rotateRight( string array[ ], int n, int amount );`
Adjust the passed array, rotating elements in the array `amount` number of times, returning the total count of all the elements that have been rotated. If n <= 0 or amount < 0, return `-1`. For example, for the array `folks[ 5 ]` shown above, `rotateRight( folks, 5, 1 )` should return `5` and have folks[0] = "tyrion" folks[1] = "samwell" folks[2] = "jon" folks[3] = "margaery" and folks[4] = "daenerys". For example, for the array `folks[ 5 ]` shown above, `rotateRight( folks, 5, 2 )` should return `10` and have folks[0] = "daenerys" folks[1] = "tyrion" folks[2] = "samwell" folks[3] = "jon" and folks[4] = "margaery".

`int shiftLeft( string array[ ], int n, int amount, string placeholder );`
Adjust the passed array, shifting elements left `amount` number of times and using the `placeholder` value for the left-most positions that get vacated. Return the total number of times the placeholder value got entered into the array (never a value more than `n`). If n <= 0 or amount < 0, return `-1` and do no adjust the passed array at all. For example, for the array `folks[ 5 ]` shown above, `shiftLeft( folks, 5, 1, "foo" )` should return `1` and have folks[0] = "jon" folks[1] = "margaery" folks[2] = "daenerys" folks[3] = "tyrion" and folks[4] = "foo". For example, for the array `folks[ 5 ]` shown above, `shiftLeft( folks, 5, 2, "foo" )` should return `2` and have folks[0] = "margaery" folks[1] = "daenerys" folks[2] = "tyrion" folks[3] = "foo" and folks[4] = "foo".

Additionally, I have created a testing tool called CodeBoard to help you check your code. CodeBoard enables you to be sure you are naming things correctly by running a small number of tests against your code. Passing CodeBoard tests is not sufficient testing so please do additional testing yourself. To access CodeBoard for Project 4, please click the link you see in this week named CodeBoard for Project 4. Inside the file named user_functions.cpp, copy and paste your implementation of the assigned functions. CodeBoard uses its own main( ) to run tests against your code. Click Compile and Run. However please be aware that no editing changes can be saved inside CodeBoard. In this anonymous user configuration, CodeBoard is read-only and does not allow for saving changes.

In an effort to assist CS 31 students with the contents of your .zip file archive, Howard has created a Zip File Checker which will echo back to y
the contents of your .zip file. Please use this file checker to ensure you have named all your files correctly.

Programming Guidelines

Your program must *not* use any function templates from the algorithms portion of the Standard C++ library or use STL <list> or <vector>. If you don't know what the previous sentence is talking about, you have nothing to worry about. Additionally, your code must *not* use any global variables which are variables declared outside the scope of your individual functions.

Your program must build successfully under both Visual C++ and either clang++ or g++.

The correctness of your program must not depend on undefined program behavior. Your program could not, for example, assume anything about t 's value in the following, or even whether or not the program crashes:

```
int main()
{
    string s[3] = { "samwell", "jon", "tyrion" };
    string t = s[3];  // position 3 is out of range
    …
```

What you will turn in for this assignment is a zip file containing these two files and nothing more:

1. A text file named **array.cpp** that contains the source code for your C++ program. Your source code should have helpful comments that explain any non-obvious code.

2. A file named **report.doc** or **report.docx** (in Microsoft Word format) or **report.txt** (an ordinary text file) that contains in addition **your name** and **your UCLA Id Number** :
   a. A brief description of notable obstacles you overcame.
   b. A list of the test data that could be used to thoroughly test your functions, along with the reason for each test. You must note which test cases your program does not handle correctly. (This could happen if you didn't have time to write a complete solution, or if you ran out of time while still debugging a supposedly complete solution.) Notice that most of this portion of your report can be written just after you read the requirements in this specification, before you even start designing your program.

How nice! Your report this time doesn't have to contain any design documentation.

As with Project 3, a nice way to test your functions is to use the assert facility from the standard library. As an example, here's a very incomplete set of tests for Project 4:

```
#include <iostream>
#include <string>
#include <cassert>

using namespace std;

int main()
{
    string a[6] = { "123", "456", "delta", "gamma", "beta", "delta" };

    assert(hasDuplicatedValues(a, 6 ) == true);
    assert(hasDuplicatedValues(a, 3 ) == false);

    cerr << "All tests succeeded" << endl;
    return( 0 );
 }
```

The reason for the one line of output at the end is to ensure that you can distinguish the situation of all tests succeeding from the case where one test silently crashes the program.

Make sure that if you were to replace your main routine with the one above, your program would build successfully under both Visual C++ and either clang++ or g++. This means that even if you haven't figured out how to implement some of the functions, you must still have *some* kind of implementations for them, even if those implementations do nothing more than immediately return.

By July 26, there will be links on the class webpage that will enable you to turn in your zip file electronically. Turn in the file by the due time above. Give yourself enough time to be sure you can turn something in, because we will not accept excuses like "My network connection at home was down, and I didn't have a way to copy my files and bring them to a SEASnet machine." There's a lot to be said for turning in a preliminary version of your program and report early (You can always overwrite it with a later submission). That way you have something submitted in case there's a problem later.

G31 Build Commands

```
g31 -c array.cpp
g31 array.o -o runnable
./runnable
```

Submission status

Submission status	No attempt
Grading status	Not graded
Due date	Monday, 26 July 2021, 9:00 PM PDT
Time remaining	4 days 7 hours
Last modified	-
Submission comments	▸ Comments (0)

Add submission

You have not made a submission yet.

◀ Midterm Sample Problems

Jump to...

Project 4 FAQ ▶