Programming Assignment 4
String Arrays

Time due: 9:00 PM, ~~February 21st~~ (Extended to Feb 23rd)

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 must 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.

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", "magdalena", "jon" };
bool b = hasNoCapitals( folks, 5 );
// should return true and not inspect the last three elements
of the array....
```

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 data[5] = { "mamaBbcca", "mamaBbcca", "12,", "98.76",
"tyrion" };
bool b = hasNoCapitals(data, 25 );
// this is 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 `data[ 5 ]` shown above, `locateMaximum( data, 5 )` should return the value 4, corresponding to the index of `"tyrion"`. If there are multiple duplicate maximum values, return the smallest index that has this maximum value. The maximum value should be determined by its dictionary-sorted order which is what < and > use in C++ to determine `true` and `false`.

`int countFloatingPointValues( const string array[ ], int n );`
Return the number of numerical floating point values found in the passed array or `-1` if n <= 0. For this operation, floating point values are numerical values that optionally include a single decimal point. They should not start with a leading + or - or include commas - only digits and an optional decimal point. For example, for the array `data[ 5 ]` shown above, `countFloatingPointValues( data, 5 )` should return `1`. For the array `folks[ 8 ]` shown above, `countFloatingPointValues( folks, 1 )` should return `0` because the first element is not a floating point value.

`bool hasNoCapitals( const string array[ ], int n );`
If all the elements in the passed array include not a single CAPITAL letter (that is, the letters A-Z), return `true` otherwise `false`. If n <= 0, return `true` since indeed there are no elements with a single CAPITAL letter. For example, for the array `data[ 5 ]` shown above, `hasNoCapitals( data, 5 )` should return `false` because of the value `"mamaBbcc!"`. For example, for the array `folks[ 8 ]` shown above, `hasNoCapitals( folks, 8 )` should return `true`.

`int shiftLeft( string array[ ], int n, int amount, string placeholder );`
For the passed array, shift all of its elements left by `amount`, filling in the right-most values with the `placeholder` value and return the number of times the `placeholder` value was used or return `-1` if n <= 0 or amount < 0. For example, for the array `data[ 5 ]` shown above, `shiftLeft( data, 5, 2, "foo" )` should return `2` and adjust the array `data` to `{ "12", "98.76", "tyrion", "foo", "foo" }`. For example, for the array `data[ 5 ]` shown

above, `shiftLeft( data, 5, 10, "bar" )` should return `5` and adjust the array `data` to `{ "bar", "bar", "bar", "bar", "bar" }`. `shiftLeft( data, -5, 10, "foobar" )` should return `-1` and not change any of the array `data`. Similarly, `shiftLeft( data, 5, -5, "foobar" )` should return `-1` and not change any of the array `data`.

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", "789", "gamma", "beta", "delta" };

    assert( hasNoCapitals(a, 6) == true );

    assert( countFloatingPointValues(a, 3) == 3 );

    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.

G31 Build Commands

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