CECS 275: LAB – STRING ARRAY

REMEMBER: The homework tips are here, and the coding standards are here.

OBJECTIVE: Get some firsthand experience with the [] operator overload and memory management. I also want you to gain some experience with writing your own exception classes.

BACKGROUND: One of the annoying features of the array in C++ is that it does not "know" its own length, and there are no checks for out of bounds errors. In this lab, we will create an string array object that will throw an exception in the [] operator override if the user attempts to access an element which is outside of the bounds of the array, and will provide a size() member function to give the size of the array. Along the way, we will learn a bit about memory management as well.

The beauty of encapsulation in general is that it allows you, the developer, to hide details of the implementation from the user that they do not need to know about. This gives you flexibility in your implementation. In this case, every time that you have to increase the size of the string array, you essentially move it to a new location in memory. If the user were accessing the array directly, you would not be able to move the array. By encapsulating the array, you provide the user an interface that never changes, no matter how many times you have to increase the space allocated to the array.

This String class lacks several methods. We do not have a way to remove elements, nor can we insert new elements into an arbitrary spot within the String array. A find function would be really handy as well. However, these are certainly easy enough to add later. I wanted to keep this lab assignment as reasonable as possible.

Exception classes can be nothing more than a formatted string, or they can be highly structured collections of data. Either way, their purpose is to alert the caller to what went wrong when the function attempted to perform its work so that the caller has enough information to resolve the problem and try again.

PROCEDURE: For this assignment, you will need to:

1. Build the String class (note, the casing is as important here as it is in sausage):
    a. Member variables (all private):
        i. The number of elements – note this is the number of elements that the user has added to this particular String array. The size of the array is an**other** member variable.
        ii. The size of the array – the number of elements that the String array could have before we need to increase the size of the array.
        iii. **Pointer** to a dynamically allocated array of strings. Note that the individual elements in this string array are just C++ standard strings. Do not make the mistake of trying to create an infinitely recursive data structure by making the array in String to be an array of String, …
    b. Member functions:
        i. Private

1. `resizeIfNecessary` – If the number of elements already in the String is >= the size of the array, then:
    a. Allocate a new array of double the size of the old one.
    b. Copy all of the values over to the new array.
    c. Delete the old array.
    d. Set the String pointer to the new array.
2. The destructor. Use the delete [] command to deallocate the memory occupied by the array of strings.

ii. Public

1. Constructors:
    a. Default – 0 arguments that sets the number of elements and the array size to 0, and sets the pointer to the array to nullptr.
    b. Single argument constructor – the number of elements. The number of elements is still zero, but now you allocate an array of strings large enough to hold the number of elements given you in the parameter list.
2. operator [] override
    a. if the user tries to access an array element whose index is < 0 or > the number of elements, then throw an IndexOutOfBounds exception (see below).
    b. Return a reference to the string so that the user can use the [] for both a left and a right hand value.
3. add – Adds a new element to the end of the string array. Remember to call resizeIfNecessary before putting that new element on the end of the string array.
4. getnElements – Returns the number of elements already in the string array.
5. getSize – Returns the size of the array. You should note that this violates the encapsulation of the String class since the user would never really care what the size of the array is. But for this lab, I do care about the size of the array, I want to watch it grow as you add elements to the String instance.

c. IndexOutOfBounds exception class

i. Member variables (all private):
ii. The value of the index that was out of bounds.
iii. A text message.
iv. Constructor – just two parameters: the offending index and the message.

v. Getters –
   1. getIndex – retrieve the value of the index that was out of bounds.
   2. getMessage – retrieve the text of the message.

2. In your main() function you will need to:
   a. Add 5 strings to your String array. After each add call, write out to the console how many elements are in the array and the size of the array.
   b. Go into a for loop and retrieve each element in turn, using the [] operator.
   c. Attempt to access an element that is "off the end" of the array and prove that the exception gets thrown.
   d. Attempt to create a new String instance with room for -5 elements in it.

SAMPLE OUTPUT:

```
At the start, no elements, the number of elements is: 0 and the capacity is:
2
After adding one element, the number of elements is: 1 and the capacity is: 2
After adding the second, the number of elements is: 2 and the capacity is: 2
After adding the third, the number of elements is: 3 and the capacity is: 4
After adding the fourth, the number of elements is: 4 and the capacity is: 4
After adding the fifth, the number of elements is: 5 and the capacity is: 8
Billy
Joel
concerts
are
wonderful
Error, tried to access element: 5 and got message: Attempted to access index
out of bounds.
Attempting to create String with negative size:
Error message: Invalid size for the array of strings. for creating String
with: -5
Completed satisfactorily
```

TURN IN: Be sure to include:

- Your source code:
  - String.h
  - String.cpp
  - Your driver .cpp file.
- The filled out collaborate.doc file to let me know how/what each of you contributed to the overall result.
- Sample output in the file: output.txt.
- Do not forget to demonstrate your code to me before you leave.