Kripash Shrestha

CS 202 Spring 2017

Project 5 Documentation

4/7/2017

The purpose of this program is to create my own string class. This will be helpful for me as I will learn to allocate and deallocate objects. My first attempt to finish this project was to transfer the given class structure to my own file. After doing that, I created the templates for my functions in the another cpp filed and named is MyString.cpp. I created my own string copy, string compare, string length and string concatenation function for use in the object.The deallocating function was very straight forward as I check for the member m_buffer for null. And if it is null, I leave it alone, if not, I delete it and then point it to null. Also, I would have to change the member m_size to 0. The buffer_allocate function was very similar expect I checked if m_buffer was not null and then I called buffer_deallocate if it wasn't. If it was null, I would set the m_size to the size passed in the parameter and allocate memory based on the size parameter passed.

The default constructor is used to set the m_size to 0 and then the m_buffer to null. The paramterized constructor sets the m_buffer to null at first and then sets the m_size of the object by calling my string object and using the parameter string passed for it. Then I allocate memory based on the m_size that I retrieved and call my copy string function to copy the parameter string to m_buffer of the object. The copy constructor sets the object's m_buffer to null and then gets the parameter object's m_size and assigns it to the object's m_size. Then the function will use that m_size to allocate memory for the object's m_buffer and calls my copy string to copy the parameter object's m_buffer string to the object's m_buffer. The destructor is just used to delete the instance of the object and deallocate m_buffer of the object.

The size returns the size of the object's m_size since I include null terminator for memory allocation. The length returns the size of the object's m_size minus since the null terminator isn't written in simple English.

The operator overloading was simple and straight forward for the most part expect for the addition (+) operator. I started segmentation faulting in this function because I did not check for if m_buffer was null or not before deleting the memory. So, in some cases, I would be trying to delete memory that did not exist or that I did not have access to, which caused me to segmentation fault. I solved this issue by putting cout statements in my allocate, deallocate, and operator overloaded assignment (=) operator to get to the root of the problem. After realizing it was not my allocate and deallocate functions, I tested my assignment operator overloaded function. It was working fine as all the couts printed properly. Immediately jumping to my addition operator function, I realized it seg faulted because the first cout statement after the delete m_buffer did not work. I then added the check for if m_buffer was null or not before deleting the m_buffer. With this I could solve the problem of seg fault. I also realized that because of that, the size was randomly being assigned, so at some points the size could be negative or too large. And I realized there could be 3 outcomes, a random allocation that somehow works at random times, a size too large for the allocation to work, making the program seg fault and then a negative size which causes allocation error.

If I had more time to do the project, I would work on testing out more than the given test fields and parameters in the main. And I would like to add in more cout statements to show what is happening where. I had all this beforehand but decided to remove them to have the program work as intended.