Abraham Meza

Christos Papachristos

CS 202 – 1001

April 03, 2019

Project 7 Documentation

**Description:**

For this project you will create your own **String** class. You may use **square bracket**-indexing, **pointers**, **references**, all **operators**, as well as the **<string.h>** or **<cstring>** library functions (however the std::string type is still not allowed).  
The following header file extract gives the required specifications for the class:

Continuing through Computer Science II (202), our instructor assigned us a similar project that demonstrates our abilities to complete a fragment of code; however, the project was designed to expand upon the idea of dynamic memory and to create a functional test driver that examines and applies all functions and methods devised from the given header file. By providing students a given header file that illustrates the structure of the class skeleton, our instructor wants students to demonstrate their abilities to implement that given format and to be able to test it through a main file. Compared to previous projects, we are now instructed to create a sample driver that checks all required specifications. The purpose of the program is to efficiently allocate and deallocate string memory while conducting string methods that relate to the string library. As students continue to polish their knowledge and different implementations with classes, they are also being evaluated to be able to prove the functionality of their program through individual testing.

For my design, I referred to the header file provided, and analyzed the different functions and methods our instructor wanted. Based on the information provided, students had to implement the provided class skeleton into a test driver devised by the student. I began by organizing my program in accordance to the header file. Firstly, I began with my constructors and initialized/assigned the given variables to the instruction requirements. In addition, I referred to the instructor’s previous driver programs and incorporated confirmation statements that would allow me to keep track of the execution. As I continued to implement the constructors, I realized that dynamic memory had to begin within the constructors since it laid out the foundation of class manipulation. As a result, I implemented try and catch statements that referred to the required allocation and deallocation functions that would allocate the constructor data appropriately. Once I finished the constructor implementations, I began to build my allocation and deallocation functions. Similar to the in-class lectures, I created try/catch statements that would analyze the length/size of the strings to appropriately create the minimal amount of memory space for any given string. Once I devised their implementations, I worked on the string methods (size() / length() / c\_str()). The implementation was very self-explanatory as it matched the structure of class getters. To allow the process to become easier, I incorporated the string library to allow me to find the length/size of the string without the use of for loops. Once I finished the string methods, I implemented the overload operator functions. Unlike the other projects, this project held a variety of overload functions. Even so, their contents were very transparent as they related to comparisons, assignments, and modification, which could be easily executed through loops and conditional statements. Even so, I realized that any modification to the string meant that the memory had to be re-allocated to add/remove memory space if needed. Through each operator overload, I implemented a try/catch and delete statement that would deallocate and re-allocate the previous string memory with the modified version; as a result, the dynamic memory would always be efficiently used.

After finishing the MyString program file, I moved onto working on my test driver. At first I did not know how to test the individual functions and methods except by calling them. As a result, I looked at the previous projects where the instructor provided us with a test driver and mimicked the structure. In doing so, I was able to structure my main file correctly, and tested my implementations with pre-determined buffer strings that were located in the instruction sample. Though the professor encouraged us to implement our own strings, I resorted to the sample strings within the parameters to double check that the implementation as correct just to be safe. I implemented additional confirmation statements and organizers in order to clearly read the output and track the execution on the terminal screen.

Subsequently after a few tests, I was satisfied with my results. Though the instructor did not provide a sample output, he was very clear to what the functionality was supposed to be for each string method. By comparing the driver to the previous projects, I was able to map out and correct any possible mistakes. In doing so, I was able to reliably test my code and modify it accordingly. In addition, my confirmation statements allowed me to make sure that every specification was completed and functional.

All in all, the project was very efficient in teaching students about dynamic memory, while allowing them to further master class manipulation and implementation through the use of their own test drivers. Personally, I enjoyed the simple task of completing the instructors code. Even so, I was confused at first due to the fact that I did not know how to test my implementation correctly. Personally, I am used to directly implementing my code to perform a certain task, rather than checking that everything works correctly. As a result of this project, students are now able to see the benefits of practicing memory properties and how to precisely debug their code. If I were to have more time with the project, I would modify it by printing out the pre-determined strings to allow the user to see its correlation to the specifications provided. Overall, the project was a success and was a good example to learn from.