Abraham Meza

Christos Papachristos

CS 202 – 1001

April 24, 2019

Project 9 Documentation

**Description:**

For this project you will create a Queue class, both an Array-based and a Node-based variant. A Queue is a First In First Out (FIFO) data structure. A Queue exclusively inserts data at the back (**push**) and removes data from the front (**pop**). The Queue’s **front** data member points to the first inserted element (the front one), and the **back** data member points to the last (the rear one).

Continuing through Computer Science II (202), our instructor assigned us a project that implements all of our previous knowledge on test drivers and class implementation in order to create different queue variants. Instead of completing a fragment of code, students are to design two different queue systems (node-based and array-based) in order to show that they have mastered queue modification and pointer manipulation within class skeletons. Though the instructor does not provide a given code to follow, he provides sample header files within the project instructions. Though he shows the main concepts that need to be incorporated, the students must figure out how to implement the data in a queue like format. Compared to previous projects, we are instructed to create a sample driver that checks all required specifications. . The purpose of this project tis to create queues that can be modified accordingly with predetermined data in the form of nodes and array manipulation (indexing). In doing so, students will be able to modify queues accordingly to incorporate accurate information and efficient dynamic memory handling, if needed. 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 driver testing of their own creation.

For my design, I referred to the header file samples provided in order to structure the programming implementation order and methods. I began to devise the implementation of the array-based queue. Given the many parameters, I was able to structure the program without much difficulty. I began to create the constructors and initialized/assigned the given variables to their corresponding members. As an array environment, I had to manipulate the data through the utilization of *for* loops for access to specific elements. Once I had assigned the data accordingly, I realized that the main functionality of queues is to incorporate array modification in the order through the utilization of push() and pop() methods. Through the realization, I implemented the methods within the *for* loops of the constructors to maintain the structure of the array throughout the program. Another thought that presented itself was the method in which dynamic memory would be handled within the array-based queue. I had realized that there was no need to dynamically allocate/deallocate memory in an array-based queue due to the fact that it was always going to be consistent (max limit), thus the destructor had no expected implementation. After working on the constructors, I moved onto the individual method and operators. Most of the methods incorporated into the class skeleton revolved around the idea of array accessing: i.e. front () manipulates pointers to the first element, back () manipulates pointers to the last element, etc. As a result, the implementation of the positioning methods was self-explanatory as it required the concept of array indexing and calling. From thereafter, I began to focus on the queue methods. The most important methods in the program are push() and pop(). After researching the concepts of push and pop within queue data, I was able to implement them through the use of *if* statements to check the status of the array (if empty()) in order to see if the value passed as an argument could be added to the line. While push() adds an assigned value to the end of the array queue, pop() simply removes the targeted value. After implementing the crucial methods, I continued with the helper methods that would simply check the size (m\_size) of array in order to perform conditional statements for other methods. Once I completed the methods, I began to work on the operators. Similar to the many projects in the past, the assignment operator served the same purpose and held the same structure; however, instead of assigning values to one another, a queue assignment operator would simply push the elements given to be identical to the given array. From there, I implemented the insertion operator to be linked to the serialize() method in order to print the queue to screen. Queues can be seen as a one dimensional array, thus I simply utilized array indexing (*for* loop) to access and print the present data within the array.

After concluding and testing the array-based queue, I began to construct the node-based programming. Though both queues were to execute identical functionalities, their programming utilized different manipulations. While array-based lists utilized the idea of array indexing, node lists utilize pointer manipulation to access/modify specific data. Through my knowledge of pointer manipulation, I simply referenced my created array-based queue and modified it to be compatible with pointer arithmetic. By creating several pointer variables and objects that would track and transfer data, I was able to successfully implement the same methods. As a result of using pointer arithmetic (arrow operators, dot operators, etc.), I could simply point to the next object. By accessing the data of the pointed object, I could manipulate it accordingly without much trouble. Through the utilization of new pointer declarations and counting variables, I was able to modify and control the data and dynamic memory. Since the node-based queue could have infinite members, I had to manage the dynamic memory throughout the program. I would simply delete old data and reallocated the memory of the node elements. Similar to the array-based queue, the helper methods simply tracked the size of the node through the utilization of temp variables and loops. All other methods utilized *while* loops to scan through the elements and used arrow operators to navigate through the positioning (m\_front/m\_back). Generally speaking, the implementation of node-based programming was straightforward as data was manipulated through temporary objects and loops.

After finishing the both programing files, I moved onto working on my test driver. I immediately realized that the most efficient way to organize the queue data was to organize it in Datatype structures; in doing so, the data could be assigned easier. I simply recycled the instructor’s previous implementation of data types and modified the other programs accordingly. Similar to the previous project, I looked onto previous assignments and samples in order to construct the test driver. In doing so, I was able to structure my main file correctly, and tested my implementations with pre-determined data. I implemented additional confirmation statements and organizers in order to clearly read the output and track the execution on the terminal screen, then went back and deleted the placeholders after the debugging process.

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 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 queues and the differences between array and node data-manipulation, while allowing them to further master class implementation through the use of their own test drivers. In doing so, students were able to reflect and strengthen their utilization of pointers and array indexing. Students are now able to see the strengths and weaknesses of each structure. If I were to have more time with the project, I would like to be able to modify the usage of Datatypes within the main file, without recycling the instructor’s work. Overall, the project was a success and was a good example to learn from.