Nicholas Ang

CS 202 - 1101

Project 9 Documentation

Purpose: The purpose of this project is to test our ability to create queue based dynamic data structures as well as manipulate them to hold and change data. This project requires us to be able to dynamically allocate and deallocate memory and create Node and Array based queues. The Array Queue uses a circular queue while the Node Queue uses a linear linked queue. Additionally it requires the use of pointers and other basic C++ coding to be able to manipulate the queues.

Design: The project is split up into two main folders with the source code and the libraries and header files. The proj9.cpp is the main source file which holds the test driver for the Node Queue class and the Array Queue class. The test driver essentially tests all the functions in the classes to test if it works properly. The ArrayQueue class has 3 constructors, a destructor, 11 methods, and two operator overloads. The ArrayQueue object is designed to be a circular array. The default constructor sets all the initial values to 0 while the parameterized constructor takes in a value and sets each element in the array to the value. The copy constructor copies all the values of the other object and also copies each element to its appropriate position. The destructor clears the queue and destroys the object. The operator overload for assignment = is similar to the copy constructor but it checks for self assignment before doing anything. After that, it clears the array and then starts copying the values of the right hand side over. Then it copies the elements over into the appropriate spot. The front and back functions return the m_array at the respective spot of front or back. The const versions of the front and back functions similarly do the same thing as the

normal but for const. The size function outputs the size of the queue. The empty and full functions of the Array Queue class check whether the queue is empty or full and return true or false. The push function creates a new element and adds it to the back of the queue. The pop function takes the front most element and deletes it from existence. Since it is a circular array, the push and pop methods use the % operator to properly divide the number so that it goes in a circle. The clear function clears out the queue and sets the queue values to 0. The serialize function takes an Array Queue object and outputs it to the terminal. The operator overload for << just calls the serialize function to output to terminal. The NodeQueue class has a similar function to the ArrayQueue object. However, the NodeQueue object is designed to be a linear forward linked queue. It has the same amount of constructors, methods, and operator overloads with essentially the same functions. The default constructor sets m_front and m_back to NULL. The parameterized constructor for NodeQueue sets the initial node as the front then adds on from there until the last which is m_back, setting each node to the value given. The copy constructor copies each node while setting the node to m_next until it finds the end of the other node queue which is NULL. The assignment operator overload for = does something similar but clears first and checks for self assignment. The destructor calls the clear function before destroying the NodeQueue object. The size function outputs the size of the queue. The empty and full functions of the Array Queue class check whether the queue is empty or full and return true or false. The front and const front functions return the node of m_front and outputs its data. The back and const back functions return the node of m_back and outputs its data. The clear function dynamically deallocates the queue and deletes every node inside the queue. Then it sets everything back to NULL. The push function creates a new node and sets the m_back->m_next

to the node and m_back to the node. It also checks whether the node is the first of its kind. The pop function removes and deletes the front most node from existence. It only does this by checking whether the node queue is empty or not and if it is empty, do nothing. The serialize function loops through the queue and outputs every element inside the queue. The operator overload for << calls the serialize function and outputs the data to the terminal.

Problems/Challenges: This project was fairly straightforward after we learned more about queues and implementing linked lists. After doing project 8, I was able to understand more about dynamically allocated data structures so project 9 was not that hard. I had a bit of trouble with free() as I had made a mistake in my copy constructor and had to debug to find where I missed the code. I needed to properly allocate new objects in the copy constructor. I also had a problem with push and pop functions in the NodeQueue class but I realized that I had to check for NULL and check if the queue was empty and the new node would be the first. Otherwise, the project was not too bad.

Possible Changes: I would try to clean up the code a bit and try out more things with the queue. I would also try to see how to move the queue back and forth and experiment with different functions for a queue. I would try to learn more about data structures and dynamic memory applications to try and apply it to my code.