Design Program # 2

Eduardo Robles

For this week’s program I have designed a program that will consist of manipulating stacks and queues of stops. The queue will keep a list of the places you will be going to. It will be designed using a circular linked list. This will allow the list to grow and shrink as necessary. The stack will keep the stops as the traveler progresses through the road trip. This will be implemented with a linked list of arrays. This will allow for dynamic growth and direct access. A stop class will be created to keep track of all the information and have in in one place. This class will take care of setting all the information and creating an object. A main class will be created where all the testing and user input will take place.

The first class I will create will be the Stop class. This will be the data that will be stored in the nodes for the stack and the queue. The class will have private member variables that will represent the data. Each member variable will have its own set function to make changes to the data. This will allow the data to not be directly accessed by the user. A display function will also be created to display all the data in order. This display function will have a return value of an int to show success /fail. The constructor will initialize all the variables to a NULL state. The destructor will deallocate all the dynamic memory and set everything back to NULL. Separating all the stop information into a class will allow for easy maintenance and changes. Any errors regarding a stop will be easy to pinpoint and fix.

For the stack part a class will be created. The class will be a linear linked list of arrays where each node has a Stop object and a pointer to the next stop. The combination of linear linked list and array allows for more direct access.

The default constructor will take a value for the size of the array. It will set its private pointer variables head and rear to a NULL state. This will then later allow the array to be dynamically allocated according to the size. To add things to the list a push function will be implemented that takes a Stop object by reference. The data will be added to the array part in the linked list. The function will return 1 if the operation was successful false if it fails. Once an array is filled up a new node will be created and new array will be used for adding data. To remove things from the stack a pop function will be created.

This pop function takes a stop object by reference so the user can have what is being removed. The function will retrieve the data and remove top most item on the stack. It will return a 1 if successful and 0 if the stack is empty. In order to look at what’s at the top of the stack a peek function will be created. This takes a stop object by reference that will be updated to the top most item. This function will return 1 if successful 0 if the stack is empty. Nothing will be deleted when using the peek.

The final function will be the display. Its going to have a public and private version in order to use recursion and keep the user from accessing head. This function will simply print out the entire stack and return 1 if successful, false other wise. If the list is empty nothing should be displayed and a failure should be return to the client program.

For the queue part a class will be created. The class will be a circular linear linked list of stops. This will represent where the user will be going at each given time. A node will contain a Stop object and a pointer to the next node. It will have a head and tail pointer to allow manipulation of the data.

To add stuff to the queue an enqueue function will be created. It will take a stop object by reference and a new node will be added at the end of the list. If the queue is empty, then it will be first thing added. The function will return 1 if successful and 0 otherwise. This doesn’t have to be down recursively because we will have a rear pointer and can access it directly to append.

To remove things a dequeue function will be created. When removing from the queue it will be from the front so no traversals are needed. It will take a Stop object by reference for an argument so that it can be filled with the Stop being removed. This doesn’t need to be done recursively since we have direct access to the front of the queue with the pointer. The function will return 1 if successful and 0 if the queue is empty and nothing is there.

Like the stack class a peek function will also be created allowing us to see what’s first in the queue. It will take a Stop object by reference and will filled with the first item in the queue. Because we have a head pointer, we don’t have to traverse and no recursion is really needed. All that needs to be done is access the data at the head pointer. If the queue is empty a 0 will be returned, otherwise it will copy the information to the argument and return a 1;

Last a display function will be created. This function will have both a public and private version. The public version will take no arguments and then call its private which will a node pointer as an argument. The private one will be recursively called and will print out the contents of the queue in order. These functions will return an integer value indicating whether the display was successful or not. If the queue is empty a 0 will be returned , if not empty a 1 will be returned.