**Zakeriya Muhumed**
**Program #2 Writeup**
**CS163**

For me, this program assignment was challenging, but it was also rewarding. I was tasked with putting stacks and queues-related software into action. To use these data structures in a practical setting is the goal of this assignment. The situation involves paying bills, which takes place each month. I have a stack of bills waiting to be processed and recorded into a spreadsheet, just like the majority of individuals. I recorded the name, amount, description, type and due date. I travel through the stack.

I have to develop the stack class and the queue class for this project. The bills I needed to pay were represented by the stack class. I'd add new bills to the digital stack every time they came in. I would process the bills in the reverse order when it came time to pay them. This is due to the fact that a stack enables us to retrieve data in the reverse order from which it was sent. The first thing to be retrieved was the last item pushed.

I recorded information about each bill, including the name of the payee, the amount, the purpose for which it was used, the category of expense, and one more area of my choosing.
I represented the expenses I was keeping track of with the queue class. The need to maintain the sequence in the order in which the expenses were paid made a queue ideal for this situation. I required three queue objects for this assignment to reflect the three different expenditure kinds.

An array of arrays in a linear linked list was used to implement the stack. An unpaid bill was stored in each component. The arrays were allotted dynamically, and they were all the same size. To fully test the code, I made sure the arrays were relatively tiny, such as 5 elements. Push, pop, peek, and display were among the stack  and queue ADT's features.

The circular linked list used to implement the queue had a rear pointer that pointed to the last expense and a rear->next pointer that pointed to the first expense. Enqueue, Dequeue, Peek, and Display were necessary queue functions.

I made a menu interface for my primary set of operations to ensure that the stack and queue ADTs operate effectively. This was done to ensure that a genuine application based on the ADT public functions might be created by a different team of software engineers.

I started by setting up the Stack ADT. I had to build a Stack object, push a bill to the top, display all bills that were overdue, pop the newest bill from the top, peek at the newest bill at the top of the stack, and free up all dynamic memory.

I then carried on by putting the Queue in place. I had to build a queue object, enqueue a paid expense to the back, display the queue, dequeue as I worked on my budget, peek into the queue to see what was coming up, and free up all dynamic memory.

As an extra credit option, I implemented exception handling to handle the success or failure of my public member functions instead of returning the success or failure value. This was a great way to improve the functionality of my program.

Instead of returning the success or failure result from my public member functions whether they succeed or fail, I included exception handling as an extra credit option. This was an excellent technique to increase my program's capabilities.

In conclusion, learning about stacks, queues, and new data structures through programming was a terrific experience. It was more interesting to me since I could use these data structures in a real-world situation.