**ASSIGNMENT 2 README FILE**

By: Khadija Swailem 202301222

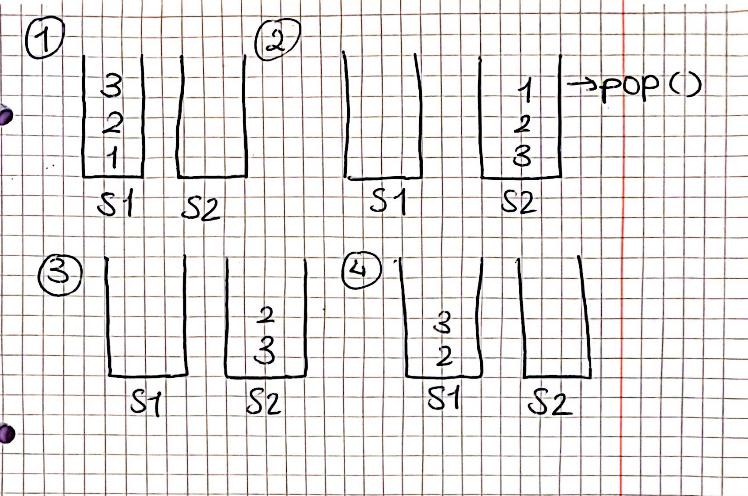
***Aim of program:***

The aim of this program is to implement queues using stack abstract data type and its methods pop and push only.

***Thought process and methodology for attempt 1:***

I used the help of inheritance to allow the stack to access the functions from the linked list class such as append() for the push method and removeEnd() for the pop method and isEmpty() to check is the stack is empty or not and printlist() to display what I enqueue and dequeue(this function is for debugging and display purposes only to review if the code runs as intended or not). I used two attempts in which one of them is more optimized than the other, in terms of the dequeue function. I made a quick sketch to help me visualize the implementation of attempt 1.At first I wanted to make the enqueue process to set two stacks one to enqueue and then if we want to enqueue again I would have to pop the existing elements and push them to stack 2 then push the new element to be enqueued to stack 1 then pop the elements from stack 2 and push them back into stack 1 so the element that was just enqueued to be the last one in the stack to achieve FIFO concept and always have the element that was entered first at the top of the stack making dequeue O(1).However, this resulted in making the enqueue operation O(n) which was unnecessary so I used only push to enqueue and leave the first entered element at the bottom of the stack and then if I want to dequeue I would do the following.I would first check if the queue is empty or not by using the isEmpty() method on the stack, if empty then an error message is printed.If the stack isn’t empty, then I can dequeue from it so I will pop all elements from stack 1 and push to stack 2,pop the top of stack 2 and then pop everything from stack 2 and push back into stack 1 to get the original queue order but with first element dequeued.

Diagram:



***Thought process and methodology for attempt 2:***

In the second attempt, my goal was to improve the complexity of the dequeue function and to achieve that without having to pop and push twice each time.At first I wanted to access the element to be dequeued directly through a function I created called returntop() that returns a pointer to the first element which is the bottom of the stack and it was implemented in the stack class in which then I can pass what it points to to the pop function by reference and use it to dequeue the first element(which is the bottom of the stack).I tried to run this code and there were some errors in the dequeuing process and it wasn’t the expected output so I decided to change it. In another attempt I implemented a function called isReverse that iterates over the list starting from the tail pops from the stack and if the item popped is equal to where cur is then keep going and set the flag to 1 meaning it’s reversed , so in first iteration the tail should be equal to the first item being popped,but still I couldn’t figure out how to use it and it was complexity O(n) and it was requested to have it complexity O(1) so I deleted the function. After discussing the assignment in class a second time I realized my understanding was wrong so I decided to try and create a flag to tell if the stack is reversed or not ,to set the value to 1 if reversed and to 0 if not reversed. I still used the same concept of accessing the top element and removing it but by using a function that already existed in the linked list class source code called removeat() and using insert() in push method as explained in the design approach of attempt 2 in the assignment report.

***How to compile my code:***

I made two make files one for each attempt so to run the code copy the contents of the make file under the name ‘Make file attempt 1’ and ‘Make file attempt 2’ into the terminal and run it.

Or a simpler type way g++ assignment 2 attempt 1.cpp

./a.out and g++ assignment 2 attempt 2.cpp ./a.out into terminal.