CSC 228 Data Structures and Algorithms, Spring 2018
Instructor: Dr. Natarajan Meghanathan

Project 5: Stack-based Implementation of Queue

Due: March 7th, 11.59 PM

You are given the code skeleton for implementing a queue using Stack (that is in turn implemented using a Doubly Linked List).

Complete the code for the enqueue(), dequeue() and isEmpty() functions of the Queue class. The code for the main function is also given to you. After you implement the above three functions, you can run your main function and capture screenshot. The queue size can range from 5 to 10 and the maximum value for an element in the queue can be 50.

As you can notice, there are two Stacks (stack1 and stack2) declared as private member variables in the Queue class. You need to use these two Stacks for implementing the functionalities of a queue. I suggest the following design (you are free to choose your own design; provide a detailed explanation in your project report if your design is different from mine).

Use stack1 to store the elements of the queue (with the invariant that the topmost element of stack1 is the element at the front of the queue and the bottommost element of stack1 is the element at the end of the queue) and use stack2 as an auxiliary data structure to implement the enqueue function. Since the topmost element of stack1 is the element in the front of the queue, the dequeue function can be simply implemented as the result of a pop operation on stack1. It is the enqueue function that needs to be thought out in greater detail and implemented. I suggest the following idea for enqueue of an integer 'data':

First check if stack1 is empty or not. If it is empty, simply push the 'data' to it and return from the enqueue function. If stack1 is not empty to start with, then pop out all the elements of stack1 and push each of them to stack2. After stack1 gets empty, push the 'data' to it. Now, pop out all the elements of stack2 and push them back to stack1. As a result of this, the newly enqueued 'data' will be in the bottom of stack1.

Submission (through Canvas):
(1) Write a pseudo code of the enqueue function implemented for this problem. Explain how the invariant that "the topmost element of stack1 is the element at the front of the queue and the bottommost element of stack1 is the element at the end of the queue is maintained with each enqueue and dequeue operation."
(2) Discuss the theoretical time complexity of the enqueue and dequeue implementations.
(3) Include the complete C++ code of the (Doubly Linked List-based) Stack-based Queue implementation, including the main function.
(4) Show a screenshot of the execution of your implementation for a queue of size anywhere chosen from 5 to 10, with the maximum value of an element in the queue being 50.