CSCI 313 – HW 2

Prof. M. Fried

Problem 4

Muhammad Abbas Ali Sajjad

When creating a stack, there are a few things we must consider. The first being how we want to implement the stack using either an array or linked-list like data structure. I chose to implement my stack as a series of nodes connected as a linked-list top down. Each Node implemented as a class in the stack holds data and a pointer to the next node in the chain.

I created a Stack class using a generic template in order to incorporate any data type. The Stack header file includes the class declaration with data members being top, bottom and int size. Top and Bottom nodes being pointers to generic nodes as Node<T>*. I chose to also use a size or length counter when adding to or removing elements to the Stack in order to make it easier to add more member functions to the Class. The Node class constructor was created so new declarations of a node can incorporate the newly added data without having to dereference followed by accessing data via dot notation.

The first method I had to create before solving the problem was push(), which adds new elements to the top of the Stack. We first check to see if the Stack is currently empty or has at least one element, by checking whether size == 0 or not. To add the 1st element to the Stack we create a new node and make both Top and Bottom Node pointers reference it. If we're adding a 2nd or later element we create a new node and have its next pointer reference the Top node and reassign the new node as the Top node. Adding a new element, i.e. a node gave us a O(1). The second method created was pop(). The main difference between a Stack data structure vs a singly-linked list is the data elements are added and removed from the TOP of the stack only. Hence, the acronym LIFO often used to describe Stacks, as "Last In First Out". In order to ensure our pop() operation is O(1). We decided to have a top down approach to our Stack, meaning the Top node points downward to the next node in line which continues until the Bottom node is reached. If we want to pop() off a node we just remove the Top node instead of traversing the Stack Bottom up. The pop() function checks first to make sure the Stack isn't empty and then creates a temporary pointer to the Top node, reassigns the current Top pointer and deleted the old Top node, essentially popping off the element atop the Stack.

In order to show all the elements in the Stack, we created a member function, printStack(). Which starts at the Top node using a temporary node pointer and traverses the Stack printing data as it goes, with tmp = tmp->next until the Bottom node is reached and tmp != nullptr. In order to print the entire stack we have a time complexity of O(n).

The insert(int index, T data) method first checks to ensure the index value provided is legal and not out of bounds, and if the index is index 0, the push() function is called. Since our Stack is top down, the Top node is index 0 and the Bottom node is index size – 1. A temp node pointer is created and traverses the Stack until it reached the node right before the index of interest. Creates a new node and points to the node at index of interest and reassigns the previous node to the new node, at O(n).