CSCI 313 HW2

Prof M. Fried

Muhammad Abbas Ali Sajjad

Problem 1

The stack data structure is an important structure when dealing with data that needs to be arranged in away such that the most recent addition to our structure is the first to be evaluated or dealt with. There are two common ways to allocate memory for our stack. The first being an array where the size of the array would be fixed and the top of the stack would correspond to the index value of the appropriate value. The other common way being an implementation of a linked-list where instead of contiguous elements in memory we utilize a series of nodes which house data and include a pointer to another node. The stack design is to have the node pointer at the top reference the most recently added element, the bottom being the element that was first added.

The most common methods for the stack are push() and pop(). The push() function creates a new node and then proceeds to point to the top most node and the top most node pointer is then reassigned to the new node. The pop() method deletes the top most node, like we mentioned before, the stack only removes from the top the same place we remove it. Hence a temporary node pointer is created and the top node pointer traverses to the next node and then the node referenced by the temporary node is deleted. The main difference as compared to the array based implementation is that when we pop() off the top most element in a list we delete the node that was dynamically allocated and it must be deleted because we don't want to have any memory leaks. The array based implementation would only reference a new top node and not delete the array space.

Stacks are very useful when it comes to arranging data in away where we want the most recent elements being the most accessible. Examples of stacks being used are webpage history for example, every time we visit a webpage we add it to the stack and if we want to go back we access the top most element and pop() as we regress.

Another powerful use for stacks are mathematical operations, programming compilers typically do the hard work for us. There are 3 ways to carry out mathematical operations using a stack. Prefix, infix and postfix notations. We're going to focus on postfix notation. The goal of postfix is to carry out a mathematic expression one operator at a time. The stack here is useful to keep track of the operands and file read object is used to grab the operator and check to see which of the operations are currently supported. The stack takes in one operand at a time from the file, pops them off and carries out the appropriate operation and pushes them back onto the stack. If another operation is to be carried out the next operand is added and both are popped out, operation is conducted and the cycle continues. The postfix stack differs from prefix and infix in that the operation is evaluated after the operands are pushed to the stack as opposed to before and in between.

The advantages of using stacks is because pop() and push() are both O(1). Another being it is much easier to track expression if we work with 2 operands at a time and a stack allows us to quickly and efficiently evaluate 2 operands, calculate a result and add it to the next operation.