# CSE207 Lab Manual (Stack and Queue)

## Stack

## **Create a stack class. Then solve the following problems using objects of stack class.

The stack class will have push(int item), pop(), getLength(), isEmpty() and isFull() operations.

1. You are given an array of numbers. You have to create a stack with the sum of digits of each number from given array.

   For example, a given array is: 231, 501, 46, 57, 53. You have to calculate the sum of digits of each number i.e., (2+3+1=6, 5+0+1=6, 4+6=10, 5+7=12, 5+3=8) and push them into a stack. Then you will have to print the elements of stack by popping them until stack is empty.

   | Sample Input | Sample Output |
   |---|---|
   | a. 231 501 46 57 53 | 8 12 10 6 6 |
   | b. 96 555 48 91 13 | 4 10 12 15 15 |

2. Write a program to check whether a given string a palindrome or not. A palindrome is a word which is same forward and backward. For example, **reviver** is same either way you read it forward or backward. You must have to use stack to solve the problem.

   Hint: First take the input as a string. Then push the characters of the string into a stack one by one. After that, create another stack and push the characters of the previous stack according to the LIFO principle. Then compare two stacks whether those are identical or not. If two stacks are identical then the string is palindrome.

   | Sample Input | Sample Output |
   |---|---|
   | `a. reviver` | `palindrome` |
   | `b. 1221` | `palindrome` |
   | `c. tea` | `not palindrome` |

3. Write a program for expression parsing. For this particular program, you have to consider only parenthesis parsing. You have to read an expression from user and determine whether there is any bracketing error in that expression or not. For example, consider the expression `((a+b)*c)`. This expression does not have any error because the bracketing in this expression is maintained properly. But `(a+b)*c)` has error in it. You have to use stack to solve this problem.

   Hint: Traverse through the string and if you find any openingparenthesis i.e., `(` or`{` or `[`, push them into the stack. And if you find any closing parenthesis i.e., `)` or `}` or `]`, traverse the stack and pop out the first corresponding opening parenthesis. Repeat the whole process till the null

pointer of the string. After that, if the stack is empty, the expression has no error in it otherwise there is parenthesis error in the expression. And one other thing, if there is a closing parenthesis found before finding corresponding opening parenthesis then the expression is wrong.

| Sample Input | Sample Output |
|---|---|
| a. ((a+b)*c) | valid |
| b. [{a+c)] | invalid |
| c. {a+d*(b+c)}[x+y] | valid |
| d. }a-b(g*f) | invalid |

4. Write a program to convert an infix expression to postfix expression.

   Hint: Input an arithmetic expression as string. Now go through each character of the string. If a character is operand, directly push it to the output. If the character is operator, you have to push it to the stack. Remember, you cannot push an operator into the stack if there is any high priority operator in the stack. In that case, you have to pop high priority operators and push them to the output. If you find an opening parenthesis, push it to the stack and if a closing parenthesis is found, you have to pop corresponding opening parenthesis from the stack. The parentheses do not go to the output. Repeat the whole process until the input string and the intermediate stack is empty.

   Priority of operator: (higher the operator, higher the priority)
   a. * /
   b. + -

| Sample Input | Sample Output |
|---|---|
| a. A*(B+C)*D | ABC+*D |
| b. (A/(B*C))-((D*E)+F) | ABC*/DE*F+- |

## Queue

## **Create a queue class. Then solve the following problems using objects of queue class.

The stack class will have enqueue(int item), dequeue(), getLength(), isEmpty() and isFull() operations.

5. Write a program that implements queue operations using two stacks. You will be able to use only push and pop functions of the stacks. [Hint: Follow the class lecture]

6. Take an integer N as input and then take N number of elements as inputs and insert them in the queue using enqueue function. Then reverse the queue using the allowable

functions. [Hint: Pop the elements from the queue and insert into the stack. (Topmost element of the stack is the last element of the queue). Then pop the elements of the stack to insert back into the queue. (The last element is the first one to be inserted into the queue)]

| Sample Input | Sample Output |
|---|---|
| Queue Before: 10 20 30 40 50 | Queue after: 50 40 30 20 10 |

7. Given an integer k and a queue of integers, we need to reverse the order of the first k elements of the queue, leaving the other elements in the same relative order.

   [**Hint:** Create an empty stack.
   One by one dequeue items from given queue and push the dequeued items to stack.
   Enqueue the contents of stack at the back of the queue
   Dequeue (size-k) elements from the front and enqueue them one by one to the same queue.]

| Sample Input | Sample Output |
|---|---|
| 10, 20, 30, 40, 50, 60, 70, 80, 90, 100<br>k = 5 | 50, 40, 30, 20, 10, 60, 70, 80, 90, 100 |