**C-DAC Mumbai Date 29/09/2024**

**Subject: Algorithm and Data Structure**

**Assignment 3**

**Solve the assignment with following thing to be added in each question.**

-Program

-Flow chart

-Explanation

-Output

-Time and Space complexity

Submission Date: 01/10/2024

**1. Implement a Stack using an array.**

* **Test Case 1**:  
  Input: Push 5, 3, 7, Pop  
  Output: Stack = [5, 3], Popped element = 7
* **Test Case 2**:  
  Input: Push 10, Push 20, Pop, Push 15  
  Output: Stack = [10, 15], Popped element = 20

**import java.util.Arrays;**

**import java.util.Scanner;**

**class Stack1{**

**static final int max= 3;**

**int arr[] = new int[max]; ;**

**int top;**

**Stack1()**

**{**

**top = -1;**

**}**

**boolean isEmpty()**

**{**

**return (top < 0);**

**}**

**boolean isFull()**

**{**

**return (top == max-1);**

**}**

**boolean push(int x){**

**if(top >= max-1){**

**System.out.println("stack overflow");**

**return false;**

**}**

**else**

**arr[++top] = x;**

**return true;**

**}**

**int pop(){**

**if(top < 0){**

**System.out.println("stack is underflow");**

**return -1;**

**}**

**return arr[top--];**

**}**

**/\***

**void display(){**

**if(isEmpty()){**

**System.out.println("stack Empty");**

**}**

**else**

**System.out.println("stack elemnets are: ");**

**System.out.println();**

**}**

**void show(int index){**

**if(index < 0) //base condition**

**return;**

**System.out.print(arr[index]+" ");**

**show(index-1); //recursive call**

**}\*/**

**public String toString(){**

**StringBuilder sb = new StringBuilder("[");**

**for(int i=0; i<=top;i++){**

**sb.append(arr[i]);**

**if(i<top){**

**sb.append(",");**

**}**

**}**

**sb.append("]");**

**return sb.toString();**

**}**

**public static void main(String []args){**

**Scanner sc = new Scanner(System.in);**

**Stack1 s1 = new Stack1();**

**Stack1 s2 = new Stack1();**

**s1.push(5);**

**s1.push(3);**

**s1.push(7);**

**int popped1 = s1.pop();**

**System.out.println("Stack = " + s1 + ", Popped element = " + popped1);**

**s2.push(10);**

**s2.push(20);**

**s2.push(30);**

**int popped2 = s2.pop();**

**System.out.println("Stack = " + s2 + ", Popped element = " + popped2);**

**}**

**}**

Time complexity:

 **Push:** O(1)

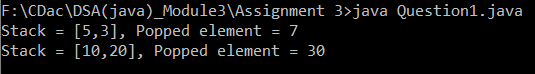
 **Pop:** O(1)

Space complexity

 **Push:** O(1)

 **Pop:** O(1)

 **Space:** O(n) //n =3🡪max



**2. Check for balanced parentheses using a stack.**

* **Test Case 1**:  
  Input: "({[()]})"  
  Output: Balanced
* **Test Case 2**:  
  Input: "([)]"  
  Output: Not Balanced

**3. Reverse a string using a stack.**

* **Test Case 1**:  
  Input: "hello"  
  Output: "olleh"
* **Test Case 2**:  
  Input: "world"  
  Output: "dlrow"

**4. Evaluate a postfix expression using a stack.**

* **Test Case 1**:  
  Input: "5 3 + 2 \*"  
  Output: 16
* **Test Case 2**:  
  Input: "4 5 \* 6 /"  
  Output: 3

**5. Convert an infix expression to postfix using a stack.**

* **Test Case 1**:  
  Input: "A + B \* C"  
  Output: "A B C \* +"
* **Test Case 2**:  
  Input: "A \* B + C / D"  
  Output: "A B \* C D / +"

**6. Implement a Queue using an array.**

* **Test Case 1**:  
  Input: Enqueue 5, Enqueue 10, Dequeue  
  Output: Queue = [10], Dequeued element = 5
* **Test Case 2**:  
  Input: Enqueue 1, 2, 3, Dequeue, Dequeue  
  Output: Queue = [3], Dequeued elements = 1, 2

**7. Implement a Circular Queue using an array.**

* **Test Case 1**:  
  Input: Enqueue 4, 5, 6, 7, Dequeue, Enqueue 8  
  Output: Queue = [8, 5, 6, 7]
* **Test Case 2**:  
  Input: Enqueue 1, 2, 3, 4, Dequeue, Dequeue, Enqueue 5  
  Output: Queue = [5, 3, 4]

**8. Implement a Queue using two Stacks.**

* **Test Case 1**:  
  Input: Enqueue 3, Enqueue 7, Dequeue  
  Output: Queue = [7], Dequeued element = 3
* **Test Case 2**:  
  Input: Enqueue 10, 20, Dequeue, Dequeue  
  Output: Queue = [], Dequeued elements = 10, 20

**9. Implement a Min-Heap.**

* **Test Case 1**:  
  Input: Insert 10, 15, 20, 17, Extract Min  
  Output: Min-Heap = [15, 17, 20], Extracted Min = 10
* **Test Case 2**:  
  Input: Insert 30, 40, 20, 50, Extract Min  
  Output: Min-Heap = [30, 40, 50], Extracted Min = 20

**10. Implement a Max-Heap.**

* **Test Case 1**:  
  Input: Insert 12, 7, 15, 5, Extract Max  
  Output: Max-Heap = [12, 7, 5], Extracted Max = 15
* **Test Case 2**:  
  Input: Insert 8, 20, 10, 3, Extract Max  
  Output: Max-Heap = [10, 8, 3], Extracted Max = 20

**11. Sort an array using a heap (Heap Sort).**

* **Test Case 1**:  
  Input: [5, 1, 12, 3, 9]  
  Output: [1, 3, 5, 9, 12]
* **Test Case 2**:  
  Input: [20, 15, 8, 10]  
  Output: [8, 10, 15, 20]

**12. Find the kth largest element in a stream of numbers using a heap.**

* **Test Case 1**:  
  Input: Stream = [3, 10, 5, 20, 15], k = 3  
  Output: 10
* **Test Case 2**:  
  Input: Stream = [7, 4, 8, 2, 9], k = 2  
  Output: 8

**13. Implement a Priority Queue using a heap.**

* **Test Case 1**:  
  Input: Enqueue with priorities: 3 (priority 1), 10 (priority 3), 5 (priority 2), Dequeue  
  Output: Dequeued element = 10 (highest priority), Priority Queue = [5, 3]
* **Test Case 2**:  
  Input: Enqueue with priorities: 7 (priority 4), 8 (priority 2), 6 (priority 3), Dequeue  
  Output: Dequeued element = 7, Priority Queue = [6, 8]

**14. Design an algorithm to implement a stack with a getMin() function to return the minimum element in constant time.**

* **Test Case 1**:  
  Input: Push 5, Push 3, Push 7, Get Min  
  Output: Min = 3
* **Test Case 2**:  
  Input: Push 10, Push 8, Push 6, Push 12, Get Min  
  Output: Min = 6

**15. Design a Circular Queue with a fixed size, supporting enqueue, dequeue, and isFull/isEmpty operations.**

* **Test Case 1**:  
  Input: Size = 4, Enqueue 1, 2, 3, 4, isFull()  
  Output: True
* **Test Case 2**:  
  Input: Size = 3, Enqueue 5, 6, Dequeue, Enqueue 7, isEmpty()  
  Output: False