**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

**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

Ans=

class Stack {

private int[] arr;

private int top;

private int maxSize;

// Constructor to initialize the stack

public Stack(int size) {

maxSize = size;

arr = new int[maxSize];

top = -1;

}

// Push method to add an element to the stack

public void push(int element) {

if (top == maxSize - 1) {

System.out.println("Stack Overflow");

} else {

arr[++top] = element;

}

}

// Pop method to remove and return the top element from the stack

public int pop() {

if (top == -1) {

System.out.println("Stack Underflow");

return -1;

} else {

return arr[top--];

}

}

// Display the elements in the stack

public void displayStack() {

if (top == -1) {

System.out.println("Stack is empty");

} else {

System.out.print("Stack = [");

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

System.out.print(arr[i]);

if (i != top) System.out.print(", ");

}

System.out.println("]");

}

}

}

// Main class to test the stack operations

public class StackImplementation {

public static void main(String[] args) {

// Test Case 1

Stack stack1 = new Stack(5);

System.out.println("Test Case 1:");

stack1.push(5);

stack1.push(3);

stack1.push(7);

int poppedElement1 = stack1.pop();

stack1.displayStack();

System.out.println("Popped element = " + poppedElement1);

// Test Case 2

Stack stack2 = new Stack(5);

System.out.println("\nTest Case 2:");

stack2.push(10);

stack2.push(20);

int poppedElement2 = stack2.pop();

stack2.push(15);

stack2.displayStack();

System.out.println("Popped element = " + poppedElement2);

}

}

**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"

Ans =

class CharStack {

char[] arr; // Array to store stack elements

int top; // Points to the top element

int maxSize; // Maximum size of the stack

public CharStack(int size) {

maxSize = size;

arr = new char[maxSize];

top = -1;

}

public void push(char element) {

if (top == maxSize - 1) {

System.out.println("Stack Overflow");

} else {

arr[++top] = element;

}

}

public char pop() {

if (top == -1) {

System.out.println("Stack Underflow");

return '\0'; // Return null character if stack is empty

} else {

return arr[top--]; // Return the top element and decrement top

}

}

public boolean isEmpty() {

return top == -1;

}

}

public class ReverseStringUsingStack {

// Method to reverse a string using stack

public static String reverseString(String input) {

CharStack stack = new CharStack(input.length()); // Create a custom stack

// Push all characters of the input string onto the stack

for (int i = 0; i < input.length(); i++) {

stack.push(input.charAt(i));

}

// Pop characters from the stack and build the reversed string

StringBuilder reversed = new StringBuilder();

while (!stack.isEmpty()) {

reversed.append(stack.pop());

}

return reversed.toString(); // Return the reversed string

}

// Main method to test the program

public static void main(String[] args) {

// Test Case 1

String input1 = "hello";

System.out.println("Test Case 1:");

System.out.println("Input: " + input1);

System.out.println("Output: " + reverseString(input1));

// Test Case 2

String input2 = "world";

System.out.println("\nTest Case 2:");

System.out.println("Input: " + input2);

System.out.println("Output: " + reverseString(input2));

}

}

**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

Ans=

import java.util.Stack;

public class Test {

static int evaluatePostfix(String exp)

{

Stack<Integer> stack = new Stack<>();

// Scan all characters one by one

for (int i = 0; i < exp.length(); i++) {

char c = exp.charAt(i);

// If the scanned character is an operand

// (number here), push it to the stack.

if (Character.isDigit(c))

stack.push(c - '0');

// If the scanned character is an operator, pop

// two elements from stack apply the operator

else {

int val1 = stack.pop();

int val2 = stack.pop();

switch (c) {

case '+':

stack.push(val2 + val1);

break;

case '-':

stack.push(val2 - val1);

break;

case '/':

stack.push(val2 / val1);

break;

case '\*':

stack.push(val2 \* val1);

break;

}

}

}

return stack.pop();

}

public static void main(String[] args)

{

String exp = "231\*+9-";

System.out.println("postfix evaluation: "

+ evaluatePostfix(exp));

}

}

**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]

Ans=

class CircularQueue {

int[] arr;

int front;

int rear;

int maxSize;

int size;

// Constructor to initialize the queue

public CircularQueue(int size) {

this.maxSize = size;

arr = new int[maxSize];

front = 0;

rear = -1;

this.size = 0;

}

// Method to add an element to the rear of the queue (Enqueue operation)

public void enqueue(int element) {

if (size == maxSize) {

System.out.println("Queue Overflow");

} else {

rear = (rear + 1) % maxSize; // Increment rear in a circular manner

arr[rear] = element;

size++;

}

}

// Method to remove and return the front element from the queue (Dequeue operation)

public int dequeue() {

if (size == 0) {

System.out.println("Queue Underflow");

return -1; // Return -1 if the queue is empty

} else {

int dequeuedElement = arr[front];

front = (front + 1) % maxSize; // Move front in a circular manner

size--;

return dequeuedElement;

}

}

// Method to return the current elements in the queue as a string

public String displayQueue() {

if (size == 0) {

return "[]";

}

StringBuilder queueString = new StringBuilder();

queueString.append("[");

for (int i = 0; i < size; i++) {

queueString.append(arr[(front + i) % maxSize]);

if (i < size - 1) {

queueString.append(", ");

}

}

queueString.append("]");

return queueString.toString();

}

}

// Main class to test the circular queue implementation

public class CircularQueueUsingArray {

// Main method to run test cases

public static void main(String[] args) {

System.out.println("Test Case 1:");

CircularQueue queue1 = new CircularQueue(5);

queue1.enqueue(4);

queue1.enqueue(5);

queue1.enqueue(6);

queue1.enqueue(7);

int dequeued1 = queue1.dequeue();

queue1.enqueue(8);

System.out.println("Queue: " + queue1.displayQueue());

System.out.println("Dequeued element: " + dequeued1);

System.out.println("Test Case 2:");

CircularQueue queue2 = new CircularQueue(5);

queue2.enqueue(1);

queue2.enqueue(2);

queue2.enqueue(3);

queue2.enqueue(4);

int dequeued2\_1 = queue2.dequeue();

int dequeued2\_2 = queue2.dequeue();

queue2.enqueue(5);

System.out.println("Queue: " + queue2.displayQueue());

System.out.println("Dequeued elements: " + dequeued2\_1 + ", " + dequeued2\_2);

}

}

**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

Ans =

class MyStack {

int[] arr

int top;

int maxSize;

public MyStack(int size) {

this.maxSize = size;

arr = new int[maxSize];

top = -1; // Indicates stack is empty

}

public void push(int element) {

if (top + 1 < maxSize) {

arr[++top] = element; // Increment top and add element

} else {

System.out.println("Stack Overflow");

}

}

public int pop() {

if (top >= 0) {

return arr[top--]; // Return the top element and decrement top

} else {

System.out.println("Stack Underflow");

return -1; // Return -1 to indicate stack is empty

}

}

public boolean isEmpty() {

return top == -1;

}

}

class QueueUsingTwoStacks {

private MyStack stack1; // Stack for enqueue operations

private MyStack stack2; // Stack for dequeue operations

public QueueUsingTwoStacks(int size) {

stack1 = new MyStack(size);

stack2 = new MyStack(size);

}

public void enqueue(int element) {

stack1.push(element); // Push the element onto stack1

}

public int dequeue() {

if (stack2.isEmpty()) { // If stack2 is empty, move elements from stack1 to stack2

while (!stack1.isEmpty()) {

stack2.push(stack1.pop());

}

}

if (stack2.isEmpty()) { // If stack2 is still empty, the queue is empty

System.out.println("Queue Underflow");

return -1; // Return -1 to indicate queue is empty

}

return stack2.pop(); // Pop the top element from stack2

}

// Method to return the current elements in the queue as a string

public String displayQueue() {

StringBuilder queueString = new StringBuilder();

queueString.append("[");

// Elements in stack2 (FIFO order)

for (int i = 0; i <= stack2.top; i++) {

queueString.append(stack2.arr[i]);

if (i < stack2.top) {

queueString.append(", ");

}

}

// Elements in stack1 (LIFO order) need to be displayed in reverse

for (int i = stack1.top; i >= 0; i--) {

queueString.append(stack1.arr[i]);

if (i > 0) {

queueString.append(", ");

}

}

queueString.append("]");

return queueString.toString();

}

}

// Main class to test the queue implementation using two custom stacks

public class QueueUsingTwoStacksMain {

// Main method to run test cases

public static void main(String[] args) {

// Test Case 1

System.out.println("Test Case 1:");

QueueUsingTwoStacks queue1 = new QueueUsingTwoStacks(5);

queue1.enqueue(3);

queue1.enqueue(7);

int dequeued1 = queue1.dequeue();

System.out.println("Queue: " + queue1.displayQueue());

System.out.println("Dequeued element: " + dequeued1);

// Test Case 2

System.out.println("\nTest Case 2:");

QueueUsingTwoStacks queue2 = new QueueUsingTwoStacks(5);

queue2.enqueue(10);

queue2.enqueue(20);

int dequeued2\_1 = queue2.dequeue();

int dequeued2\_2 = queue2.dequeue();

System.out.println("Queue: " + queue2.displayQueue());

System.out.println("Dequeued elements: " + dequeued2\_1 + ", " + dequeued2\_2);

}

}