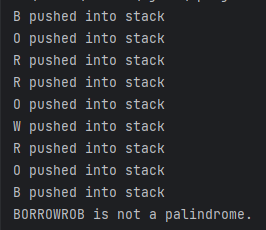
DS-LAB 7

22K-5195

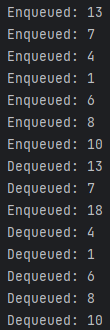
Q1.

public class Task1 {  
 public static void main(String[] args) {  
 String INPString = "BORROWROB";  
 MyStack stack = new MyStack();  
  
 for (int i = 0; i < INPString.length(); i++) {  
 stack.pushStack(INPString.charAt(i));  
 }  
  
 String reversedString = "";  
 while (!stack.isStackEmpty()) {  
 reversedString += stack.popStack();  
 }  
  
 if (INPString.equals(reversedString)) {  
 System.*out*.println(INPString + " is a palindrome.");  
 } else {  
 System.*out*.println(INPString + " is not a palindrome.");  
 }  
 }  
}  
class MyStack {  
 int maxSize = 10;  
 int top;  
 char stackArray[];  
  
 MyStack() {  
 top = -1;  
 stackArray = new char[maxSize];  
 }  
 char popStack() {  
 if (top < 0) {  
 System.*out*.println("Stack Underflow");  
 return '\0';  
 } else {  
 char x = stackArray[top--];  
 return x;  
 }  
 }  
  
  
 boolean isStackEmpty() {  
 return (top < 0);  
 }  
   
 boolean pushStack(char x) {  
 if (top >= (maxSize - 1)) {  
 System.*out*.println("Stack Overflow");  
 return false;  
 } else {  
 stackArray[++top] = x;  
 System.*out*.println(x + " pushed into stack");  
 return true;  
 }  
 }  
}



Q2.

public class Task2 {  
 public static void main(String[] args) {  
 Queue CustomerID = new Queue(10);  
  
  
 CustomerID.enqueue(13);  
 CustomerID.enqueue(7);  
 CustomerID.enqueue(4);  
 CustomerID.enqueue(1);  
 CustomerID.enqueue(6);  
 CustomerID.enqueue(8);  
 CustomerID.enqueue(10);  
  
  
 CustomerID.dequeue();  
 CustomerID.dequeue();  
  
  
 CustomerID.enqueue(18);  
  
 CustomerID.dequeue();  
 CustomerID.dequeue();  
 CustomerID.dequeue();  
 CustomerID.dequeue();  
 CustomerID.dequeue();  
 }  
}  
class Queue {  
 int front, rear, size;  
 int capacity;  
 int[] array;  
  
 public Queue(int capacity) {  
 this.capacity = capacity;  
 this.front = this.size = 0;  
 this.rear = capacity - 1;  
 this.array = new int[this.capacity];  
 }  
  
 boolean isFull() {  
 return size == capacity;  
 }  
  
 boolean isEmpty() {  
 return size == 0;  
 }  
 int dequeue() {  
 if (isEmpty()) {  
 System.*out*.println("Queue is empty. Cannot dequeue.");  
 return -1;  
 } else {  
 int item = array[front];  
 front = (front + 1) % capacity;  
 size--;  
 System.*out*.println("Dequeued: " + item);  
 return item;  
 }  
 }  
 void enqueue(int item) {  
 if (isFull()) {  
 System.*out*.println("Queue is full. Cannot enqueue " + item);  
 } else {  
 rear = (rear + 1) % capacity;  
 array[rear] = item;  
 size++;  
 System.*out*.println("Enqueued: " + item);  
 }  
 }  
  
  
}



Q3.

public class Task3 {  
 public static int evaluateExp(String Exp) {  
 Stack stack = new Stack();  
  
 for (int i = 0; i < Exp.length(); i++) {  
 char c = Exp.charAt(i);  
  
 if (Character.*isDigit*(c)) {  
 int num = 0;  
 while (i < Exp.length() && Character.*isDigit*(Exp.charAt(i))) {  
 num = num \* 10 + (Exp.charAt(i) - '0');  
 i++;  
 }  
 i--;  
  
 stack.push(num);  
 } else if (c == '+' || c == '-' || c == '\*' || c == '/') {  
  
 int operand2 = stack.pop();  
 int operand1 = stack.pop();  
  
 if (c == '+') {  
 stack.push(operand1 + operand2);  
 } else if (c == '-') {  
 stack.push(operand1 - operand2);  
 } else if (c == '\*') {  
 stack.push(operand1 \* operand2);  
 } else if (c == '/') {  
 stack.push(operand1 / operand2);  
 }  
 } else if (c == '(') {  
  
 stack.push(c);  
 } else if (c == ')') {  
  
 int result = 0;  
 while (!stack.isEmpty() && stack.peek() != '(') {  
 result = stack.pop();  
 }  
 stack.pop();  
  
 stack.push(result);  
 }  
 }  
  
 return stack.peek();  
 }  
  
 public static void main(String[] args) {  
 String Exp = "12+13-5\*(0.5+0.5)+1";  
 int result = *evaluateExp*(Exp);  
 System.*out*.println("Final Output: " + result);  
 }  
}  
class StackNode {  
 int data;  
 StackNode next;  
  
 public StackNode(int data) {  
 this.data = data;  
 this.next = null;  
 }  
}  
  
class Stack {  
 StackNode top;  
  
 public boolean isEmpty() {  
 return top == null;  
 }  
  
 public void push(int data) {  
 StackNode newNode = new StackNode(data);  
 if (top == null) {  
 top = newNode;  
 } else {  
 newNode.next = top;  
 top = newNode;  
 }  
 }  
 public int peek() {  
 if (isEmpty()) {  
 return -1;  
 }  
 return top.data;  
 }  
  
 public int pop() {  
 if (isEmpty()) {  
 return -1;  
 }  
 int data = top.data;  
 top = top.next;  
 return data;  
 }  
  
  
}



Q4.

public class Task4 {  
  
 private static int precedence(char operator) {  
 switch (operator) {  
 case '+':  
 case '-':  
 return 1;  
 case '\*':  
 case '/':  
 return 2;  
 }  
 return 0;  
 }  
 public static String Conversion(String infix) {  
 Queue output = new Queue();  
 String postfix = "";  
  
 for (int i = 0; i < infix.length(); i++) {  
 char currCh = infix.charAt(i);  
  
 if (Character.*isLetterOrDigit*(currCh)) {  
 postfix += currCh;  
 } else if (currCh == '(') {  
 output.enqueue(currCh);  
 } else if (currCh == ')') {  
 while (!output.isEmpty() && output.front.data != '(') {  
 postfix += output.dequeue();  
 }  
 if (!output.isEmpty()) {  
 output.dequeue();  
 }  
 } else {  
 while (!output.isEmpty() && *precedence*(currCh) <= *precedence*(output.front.data)) {  
 postfix += output.dequeue();  
 }  
 output.enqueue(currCh);  
 }  
 }  
  
 while (!output.isEmpty()) {  
 postfix += output.dequeue();  
 }  
  
 return postfix;  
 }  
  
 public static void main(String[] args) {  
 String infixExp = "((a+b)\*c)-d";  
 String postfixExp = *Conversion*(infixExp);  
 System.*out*.println("Infix Expression: " + infixExp);  
 System.*out*.println("Postfix Expression: " + postfixExp);  
 }  
}  
class Node {  
 char data;  
 Node next;  
  
 Node(char data) {  
 this.data = data;  
 this.next = null;  
 }  
}  
  
class Queue {  
 Node front, rear;  
  
 Queue() {  
 front = rear = null;  
 }  
  
 char dequeue() {  
 if (front == null) {  
 throw new IllegalStateException("Queue is empty");  
 }  
 char item = front.data;  
 front = front.next;  
 if (front == null) {  
 rear = null;  
 }  
 return item;  
 }  
 void enqueue(char item) {  
 Node newNode = new Node(item);  
 if (rear == null) {  
 front = rear = newNode;  
 } else {  
 rear.next = newNode;  
 rear = newNode;  
 }  
 }  
 boolean isEmpty() {  
 return front == null;  
 }  
}



Q5.

public class Task5 {  
 public static void main(String[] args) {  
 PriorityQueue CloudComputing = new PriorityQueue();  
 CloudComputing.enqueue(new Task(1, 20, 30, 512, 2));  
 CloudComputing.enqueue(new Task(2, 15, 20, 256, 1));  
 CloudComputing.enqueue(new Task(3, 25, 40, 768, 3));  
 CloudComputing.enqueue(new Task(4, 18, 25, 384, 2));  
  
 int numVMs = 3;  
 int availCPU = numVMs \* 60;  
 int availMemory = numVMs \* 1024;  
 int availStorage = numVMs \* 5;  
  
 while (!CloudComputing.isEmpty()) {  
 Task task = CloudComputing.dequeue();  
 if (task.cpuUsage <= availCPU && task.memory <= availMemory && task.storage <= availStorage) {  
  
 availCPU -= task.cpuUsage;  
 availMemory -= task.memory;  
 availStorage -= task.storage;  
 System.*out*.println("Task " + task.taskID + " executed on a Virtual Machine.");  
 } else {  
  
 System.*out*.println("Task " + task.taskID + " is waiting for resources.");  
 CloudComputing.enqueue(task);  
 }  
 }  
 }  
}  
  
class PriorityQueue {  
 Node front, rear;  
  
 PriorityQueue() {  
 front = rear = null;  
 }  
  
 Task dequeue() {  
 if (front == null) {  
 return null;  
 }  
 Task task = front.data;  
 front = front.next;  
 return task;  
 }  
 void enqueue(Task task) {  
 Node newNode = new Node(task);  
 if (rear == null || task.priority < front.data.priority) {  
 newNode.next = front;  
 front = newNode;  
 if (rear == null) {  
 rear = front;  
 }  
 } else {  
 Node curr = front;  
 while (curr.next != null && curr.next.data.priority <= task.priority) {  
 curr = curr.next;  
 }  
 newNode.next = curr.next;  
 curr.next = newNode;  
 if (newNode.next == null) {  
 rear = newNode;  
 }  
 }  
 }  
 boolean isEmpty() {  
 return front == null;  
 }  
}  
  
class Node {  
 Task data;  
 Node next;  
  
 Node(Task data) {  
 this.data = data;  
 this.next = null;  
 }  
}  
class Task {  
 int taskID;  
 int execuTime;  
 int cpuUsage;  
 int memory;  
 int storage;  
 int priority;  
  
 Task(int taskID, int execuTime, int cpuUsage, int memory, int storage) {  
 this.taskID = taskID;  
 this.execuTime = execuTime;  
 this.cpuUsage = cpuUsage;  
 this.memory = memory;  
 this.storage = storage;  
 this.priority = computePriority();  
 }  
  
 int computePriority() {  
 return execuTime + (cpuUsage + memory + storage);  
 }  
}

