Tejas Katkade

Brackets are tall punctuation marks used in matched pairs within text, to set apart or interject other text. Brackets refer to different types of brackets in different parts of the world and in different contexts.

Write a program which reads a String, which consists of alphabets [a-z, A-Z] and 3 types of brackets listed below:

1. Parentheses - ()
2. Square brackets - []
3. Braces or Curly brackets - {}

And determine whether every open bracket has a matching close bracket. If any open/close bracket doesn't have a matching close/open bracket or any extra open/close bracket then it is to be treated as invalid string.

Following are 3 examples of valid string:

- (the[is]{valid})
- {the(is[valid])}
- (this)(is)(valid)

Following are 4 examples of invalid string:

- (the[is]{invalid))
- (the[is]{invalid}}
- (this](is}{invalid)
- [this]{is}(invalid))

- (the[is][invalid])

New Tab | Full Screen | Java | 16px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java

```java
import java.util.Scanner;

class Stack {
    int[] s;
    int size;
    int top = -1;
    Stack() {
         s = new int[40];
    }

    public boolean isEmpty() {

        return top == -1;
    }

    public void push(int data) {
        s[++top] = data;
    }

    public int pop() {
        return s[top--];
    }

    public int peek() {
        return s[top];
    }
}

public class BracketMatch {
```

Ln: 1, Ch: 0, | Total Ln: 89

Compile | Self Assesment | Submit to Grader

- (the[is][invalid])

New Tab | Full Screen | Java | 15px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java

```java
        public static boolean checkParenthesis(String str) {
            Stack stack = new Stack();
            int n = str.length();
            for (int i=0; i<n; i++) {
                 if( str.charAt(i) == '[' || str.charAt(i) == '(' || str.charAt(i) == '{' ) {
                        stack.push(str.charAt(i));
                 }
                 else if(   str.charAt(i) >= 'a' && str.charAt(i) <= 'z' ||
                             str.charAt(i) >= 'A' && str.charAt(i) <= 'Z'    ){

                 }
                 else{
                     if(stack.isEmpty()){
                         return false;
                     }
                     if(   (str.charAt(i) == ']' && (char)stack.peek() == '[' ) ||
                           (str.charAt(i) == ')' && (char)stack.peek() == '(' ) ||
                           (str.charAt(i) == '}' && (char)stack.peek() == '{' )
                     ){
                           stack.pop();
                     }else{
                         return false;
                     }
                 }
            }
            if(stack.isEmpty()) {
             return true;
            }
            return false;
        }
```

Ln: 54, Ch: 24, | Total Ln: 78

Compile | Self Assesment | Submit to Grader

New Tab | Full Screen | Java | 15px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java

```java
                    if(   (str.charAt(i) == ']' && (char)stack.peek() == '[' ) ||
                          (str.charAt(i) == ')' && (char)stack.peek() == '(' ) ||
                          (str.charAt(i) == '}' && (char)stack.peek() == '{' )
                    ){
                          stack.pop();
                    }else{
                        return false;
                    }
                }
            }
            if(stack.isEmpty()) {
             return true;
            }
            return false;
        }

        public static void main(String[] args){
            Scanner sc = new Scanner(System.in);
            int test = sc.nextInt();
            sc.nextLine();
            for(int i=0; i<test; i++) {

                if(checkParenthesis(sc.nextLine())){
                    System.out.println("TRUE");
                }else {
                    System.out.println("FALSE");
                }
            }
        }
}
```

Ln: 70, Ch: 17, | Total Ln: 75

Compile | Self Assesment | Submit to Grader

Parikshak Team
parikshak[at]cdac[dot]in

Last updated:04-Oct-2024 16:21

Write a program to create **Link-List** and different operations on it. The operation descriptions along with their code is given in the table below.

No	Operation Code	Operation Description
1	**AB**	Add the node at the beginning of the list.
2	**PR**	Print the link-list from head to end of the list, each node separated by space and terminated by a newline.
3	**AE**	Add the node at the end of the list.
4	**AMA**	Add the node at the middle after a particular node.
5	**AMB**	Add the node at the middle before a particular node.
6	**DN**	Delete the node.
7	**DNA**	Delete the node after a particular node.

Your program should take command as given above, corresponding input and perform the operations till the exit command ("EXIT") is given. Assume that 'node_info' is a single integer 0 < node_info < 100

<u>Note</u>

- Partial grading is enabled for this test. i.e. You will get partial marks for correct output for each input. Also each input is designed to test one or more of 'operations'
- So it is advisable to implement the functions (operations) incrementally and keep submitting the program it for grading once you have implemented and tested an operation.

New Tab | Full Screen | Java | 13px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java | Untitled | SimpleLinkedList.java

```java
import java.util.Scanner;
class SimpleLinkedList {

    Node head;

    static class Node {
        int data;
        Node next;

        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }
    public Node createNode(int data) {
        return new Node(data);
    }

    public void addB(int data) {
        Node newNode = createNode(data);
        if(head == null){
            head = newNode;
            return;
        }

        newNode.next = head;
        head = newNode;
    }

    public void addE(int data) {
        Node newNode = createNode(data);
        if(head == null){
            head = newNode;
            return;
        }
```

Ln: 30, Ch: 40, | Total Ln: 217

Close Test

New Tab | Full Screen | Java | 13px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java | Untitled | SimpleLinkedList.java

```java
    public void addE(int data) {
        Node newNode = createNode(data);
        if(head == null){
            head = newNode;
            return;
        }
        Node temp = head;

        while(temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
    }

    public void addA(int key, int data) {
        Node newNode = createNode(data);

        if(head == null) {
            return;
        }

        Node temp = head;
        while(temp != null && temp.data != key){
            temp = temp.next;
        }
        if(temp != null) {
            if(temp.next != null) {
                newNode.next = temp.next;
            }
            temp.next = newNode;
        }
    }


    public void addB(int key, int data) {
```

Ln: 30, Ch: 40, | Total Ln: 217

Close Test

New Tab | Full Screen | Java | 13px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java | Untitled | SimpleLinkedList.java

```java

    public void addB(int key, int data) {
        Node newNode = createNode(data);
        if(head == null) {
            return;
        }
        Node temp = head;
        Node prev = null;
        while(temp != null && temp.data != key){
            prev = temp;
            temp = temp.next;
        }
        if(temp != null) {
            if(temp == head) {
                newNode.next = head;
                head = newNode;
            }else {
                newNode.next = prev.next;
                prev.next = newNode;
            }
        }
    }
    public void deleteNA(int key){
        if(head == null) {
            return;
        }
        Node temp = head;
        while(temp != null && temp.data != key){
            temp = temp.next;
        }
        if(temp.next != null) {
            temp.next = temp.next.next;
        }
    }
    public void deleteNB(int key){
```

Ln: 94, Ch: 9, | Total Ln: 212

[redacted]

Close Test

New Tab | Full Screen | Java | 13px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java | Untitled | SimpleLinkedList.java

```java
    public void deleteNB(int key){

        if(head == null) {
            return;
        }

        Node temp = head;
        Node prev = null;
        while(temp.next != null && temp.next.data != key){
            prev = temp;
            temp = temp.next;
        }

        if(temp.next != null) {

            if(temp == head) {
                head = head.next;
            }else {
                prev.next = prev.next.next;
            }
        }
    }
    public void deleteN(int key){

        if(head == null) {
            return;
        }

        Node temp = head;
        if(temp.data == key) {
            head = head.next;
        }

        while(temp.next != null && temp.next.data != key){
            temp = temp.next;
```

Ln: 94, Ch: 9, | Total Ln: 212

[redacted]

Close Test

New Tab | Full Screen | Java | 13px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java | Untitled | SimpleLinkedList.java

```java
    public void deleteN(int key){

        if(head == null) {
            return;
        }

        Node temp = head;
        if(temp.data == key) {
            head = head.next;
        }

        while(temp.next != null && temp.next.data != key){
            temp = temp.next;
        }

        if(temp.next != null) {

            temp.next = temp.next.next;
        }

    }
    public void print(){
        if(head == null) {
            return;
        }

        Node temp = head;
        while(temp != null) {
            System.out.print(temp.data+" ");
            temp = temp.next;
        }
        System.out.println();
    }

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

Ln: 94, Ch: 9, | Total Ln: 212

[redacted]

Close Test

New Tab | Full Screen | Java | 13px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java | Untitled | SimpleLinkedList.java

```java
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        SimpleLinkedList ll = new SimpleLinkedList();
       while(true){
            String str = sc.next();
            switch(str){
                case "AB":
                    {
                        int data = sc.nextInt();
                        sc.nextLine();
                        ll.addB(data);
                    }
                    break;
                case "AE":
                    {
                        int data = sc.nextInt();
                        sc.nextLine();
                        ll.addE(data);
                    }
                    break;
                case "AMA":
                    {
                        int key = sc.nextInt();
                        int data = sc.nextInt();
                        sc.nextLine();
                        ll.addA(key,data);
                    }
                    break;
                case "AMB":
                    {
                        int key = sc.nextInt();
                        int data = sc.nextInt();
                        sc.nextLine();
                        ll.addB(key,data);
                    }
```

Ln: 94, Ch: 9, | Total Ln: 212

[redacted]

Close Test

New Tab | Full Screen | Java | 13px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java | Untitled | SimpleLinkedList.java

```java
                    int key = sc.nextInt();
                    int data = sc.nextInt();
                    sc.nextLine();
                    ll.addA(key,data);
                }
                break;
            case "AMB":
                {
                    int key = sc.nextInt();
                    int data = sc.nextInt();
                    sc.nextLine();
                    ll.addB(key,data);
                }
                break;
            case "PR":
                ll.print();
                break;
            case "DN":
                int data = sc.nextInt();
                sc.nextLine();
                ll.deleteN(data);
                break;
            case "DNA":
                {
                    int key = sc.nextInt();
                    sc.nextLine();
                    ll.deleteNA(key);
                }
                break;
            case "EXIT":
                return;
        }
    }
}
```

Ln: 206, Ch: 9, | Total Ln: 210

[redacted]

Close Test

Write a Java program to create a binary search tree, where a left child node is always less than its parent node and a right child node is always greater than the parent node. Create a tree by taking input of positive integers. Once the tree is created search a particular value in the tree and print the search path starting from root. Assume that the keys in all nodes are unique.

```
        20
       /  \
    15     27
   /      /  \
 12     24    29
```

fig1.

In the Fig 1. tree search path as following for given node

20 => Root

24 => Root R L

25 => Not found

12 => Root L L

Note: R is for right**, L** is for Left

प्रगत संगणन विकास केंद्र
CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING

Welcome 240840320119

Time Left: 01:46:12

PARIKSHAK LET'S SOLVE IT

New Tab | Full Screen | Java | 15px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java | Untitled | SimpleLinkedList.java | TreePath.java

```java
import java.util.Scanner;

class BST{
    Node root;
    static class Node{
        Node left;
        int data;
        Node right;

        Node(int data) {
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }

    public Node insert(Node root, int data) {
        if(root == null) {
            root = new Node(data);
            return root;
        }

        if(data < root.data) {
            root.left = insert(root.left, data);
        }else{
            root.right = insert(root.right, data);
        }
        return root;
    }

```

प्रगत संगणन विकास केंद्र
CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING

Welcome 240840320119

Time Left: 01:45:22

PARIKSHAK LET'S SOLVE IT

New Tab | Full Screen | Java | 15px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java | Untitled | SimpleLinkedList.java | TreePath.java

```java
    public void search(Node root, int data, String str) {
        if(root == null) {
            System.out.println("Not Found");
            return;
        }
        if(root.data == data){
            System.out.println(str);
            return;
        }

        if(data < root.data) {
            str = str +"L ";
            search(root.left, data, str);
        }else{
            str = str +"R ";
            search(root.right, data, str);
        }
    }
    public void inOrder(Node root){
        if(root == null){
            return ;
        }

        inOrder(root.left);
        System.out.print(root.data);
        inOrder(root.right);
    }
}
public class TreePath{
```

Time Left: 01:44:20

PARIKSHAK LET'S SOLVE IT

New Tab | Full Screen | Java | 15px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java | Untitled | SimpleLinkedList.java | TreePath.java

```java
        System.out.print(root.data);
        inOrder(root.right);
    }
}
public class TreePath{
    public static void main(String[] args ) {
    Scanner sc = new Scanner(System.in);
        int[] arr = new int[100];
        BST bst = new BST();
        int input = 0;
        int m =0;
        while(input != -1){
            input = sc.nextInt();
            arr[m++] = input;
        }
        int j = 1;
        bst.root = bst.insert(bst.root,arr[0]);
        while(arr[j] != -1){
            bst.insert(bst.root,arr[j++]);
        }
        int size  = sc.nextInt();
        int[] a = new int[size];
        for(int i = 0; i<size; i++){
            a[i] = sc.nextInt();
        }
        for(int i = 0; i<size; i++){
            bst.search(bst.root, a[i],"Root ");
        }
    }
}
```

Ln: 81, Ch: 9, | Total Ln: 85

Implement a circular queue in c++/java, to support "***enque***" and "***deque***" operation of a given size. Your program should support following commands as given below:

- ***enque* N** // insert N in the Queue, if queue is full print the output as "**FULL**'
- **deque** // Remove the first elements from the Queue and print the value if the Queue is empty print "**EMPTY**"
- **exit** // exit the program

assume the input will be always a positive integer.

New Tab | Full Screen | Java | 15px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java | Untitled | SimpleLinkedList.java | TreePath.java | CircularQ.java

```java
import java.util.Scanner;
public class CircularQ{
    int[] CQ;
    int size;
    int front;
    int rear;
    CircularQ(int size) {
        this.size = size;
        CQ = new int[size];
        front = -1;
        rear = -1;
    }
    public boolean isEmpty()
    {
        return front == -1;
    }
    public boolean isFull(){
        return front == (rear+1)%size;
    }
    public void enque(int data) {
        if(isFull()) {
            System.out.println("FULL");
            return;
        }
        if(front == -1){
            front = 0;
        }
        rear = (rear+1)%size;
        CQ[rear] = data;
    }
```

Ln: 0, Ch: 25, | Total Ln: 73

[redacted]

Close Test

New Tab | Full Screen | Java | 15px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java | Untitled | SimpleLinkedList.java | TreePath.java | CircularQ.java

```java
    public void deque() {
        if(isEmpty()) {
            System.out.println("EMPTY");
            return;
        }
        System.out.println(CQ[front]);
        if(front == rear) {
            front = -1;
            rear = -1;
        }else{
            front = (front+1)%size;
        }
    }
     public static void main(String[] args ){

        Scanner sc = new Scanner(System.in);
        int size = sc.nextInt();
        CircularQ cq = new CircularQ(size);
        sc.nextLine();
        while(true){
            String str = sc.next();
            switch(str){
                case "enque":
                    int data = sc.nextInt();
                    sc.nextLine();
                    cq.enque(data);
                    break;
                case "deque":
                    cq.deque();
                    break;
```

Ln: 51, Ch: 24, | Total Ln: 67

[redacted]

Close Test

New Tab | Full Screen | Java | 15px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java | Untitled | SimpleLinkedList.java | TreePath.java | CircularQ.java

```java
            front = -1;
            rear = -1;
        }else{
            front = (front+1)%size;
        }
    }
     public static void main(String[] args ){

        Scanner sc = new Scanner(System.in);
        int size = sc.nextInt();
        CircularQ cq = new CircularQ(size);
        sc.nextLine();
        while(true){
            String str = sc.next();
            switch(str){
                case "enque":
                    int data = sc.nextInt();
                    sc.nextLine();
                    cq.enque(data);
                    break;
                case "deque":
                    cq.deque();
                    break;
                case "exit":
                    return;
            }
        }
    }

}
```

Ln: 51, Ch: 24, | Total Ln: 67

[redacted]

Close Test

- Input: "({[()]})"
  Output: Balanced

- Input: "([)]"
  Output: Not Balanced

New Tab | Full Screen | Java | 14px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java | Untitled | SimpleLinkedList.java | TreePath.java | CircularQ.java | Parenthesis.java

```java
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

class Stack {
    int[] s;
    int size;
    int top = -1;
    Stack() {
         s = new int[40];
    }
    public boolean isEmpty() {
        return top == -1;
    }
    public void push(int data) {
        s[++top] = data;
    }
    public int pop() {
        return s[top--];
    }
    public int peek() {
        return s[top];
    }
}
public class Parenthesis {
       public static boolean checkParenthesis(String str) {
           Stack stack = new Stack();
           int n = str.length();
           for (int i=0; i<n; i++) {

               if( str.charAt(i) == '[' || str.charAt(i) == '(' || str.charAt(i) == '{' )
               {
                   stack.push(str.charAt(i));
               }else{
```

Ln: 19, Ch: 5, | Total Ln: 64

[redacted]

Close Test

New Tab | Full Screen | Java | 14px | Eclipse | Word Wrap | Shortcut

Untitled | BracketMatch.java | Untitled | SimpleLinkedList.java | TreePath.java | CircularQ.java | Parenthesis.java

```java
               if( str.charAt(i) == '[' || str.charAt(i) == '(' || str.charAt(i) == '{' )
               {
                   stack.push(str.charAt(i));
               }else{

                  if(stack.isEmpty()) {
                      return false;
                  }

                   if(   (str.charAt(i) == ']' && (char)stack.peek() == '[' ) ||
                         (str.charAt(i) == ')' && (char)stack.peek() == '(' ) ||
                         (str.charAt(i) == '}' && (char)stack.peek() == '{' )
                   ){
                       stack.pop();
                   }else{

                       return false;
                   }
               }
           }
           if(stack.isEmpty()) {
            return true;
           }
           return false;
       }
       public static void main(String[] args) throws IOException{
           BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
               if(checkParenthesis(br.readLine())){
                   System.out.println("Balanced");
               }else {
                   System.out.println("Not Balanced");
               }
       }
}
```

Ln: 57, Ch: 33, | Total Ln: 64

[redacted]

Close Test

Implement a stack using array in c++/java, to support "push" and "pop" operation of a given size. Your program should support following commands as given below:

- push N // insert N in the stack, if stack is full print the output as "Stack Overflow"
- pop // Remove the last elements from the Stack and print the value and if the stack is empty print "Stack Underflow"
- exit // exit the program

assume the input will be always a positive integer.

Implement a stack using array in c++/java, to support "push" and "pop"

```java
import java.util.Scanner;

public class StackDemo {

    int[] s;
    int size;
    int top = -1;

    StackDemo() {
        this(5);
    }
    StackDemo(int size) {
        this.size = size;
         s = new int[size];
    }
    public boolean isEmpty() {
        return top == -1;
    }
    public boolean isFull() {
        return top == size -1;
    }
    public void push(int data) {
        if(isFull()) {
            System.out.println("Stack Overflow");
            return;
        }
        s[++top] = data;
    }

```

Ln: 26, Ch: 5, | Total Ln: 60

Implement a stack using array in c++/java, to support "push" and "pop"

```java
    public void pop() {
        if(isEmpty()) {
            System.out.println("Stack Underflow");
            return;
        }
        System.out.println(s[top--]);
    }
    public static void main(String[] args ){
        Scanner sc = new Scanner(System.in);
        int size = sc.nextInt();
        StackDemo s = new StackDemo(size);
        sc.nextLine();
        while(true){
            String str = sc.next();
            switch(str){
                case "push":
                    int data = sc.nextInt();
                    sc.nextLine();
                    s.push(data);
                    break;
                case "pop":
                    s.pop();
                    break;
                case "exit":
                    return;
            }
        }
    }
}
```

Ln: 56, Ch: 4, | Total Ln: 58