| **Part A**  **Name:- Aryan Srivastava**  **Roll No:- A073**  **Subject:- Data Structures and Algorithms**  **Program: B Tech/MBA Tech CE 2nd Year** |
| --- |
| **Aim:**  To study and implement concept of Stack data structure |
| **Prerequisite:** C++ Programming |
| **Outcome:** To implement parenthesis matching using stack. |
| * **Theory:** Stack is a linear data structure which stores the elements in an ordered manner. * The elements in a stack are added and removed only from one end which is called top. * The policy is LIFO, the element that was inserted last is the first one to be taken out.   Operations-   1. Push(element)- inserts an element at top of the stack 2. Pop() – removes the topmost element 3. Peek()- returns the topmost element without removing 4. isEmpty() – checks if stack is empty 5. isFull()- checks if stack is full   **Parenthesis matching using stack:**  Stacks can be used to check the validity of parentheses in any algebraic expression.  Algebraic expression is valid if for every open bracket there is a corresponding closing bracket.  For example, the expression (A+B} is invalid but an expression {A + (B – C)} is valid.  Algorithm  Parenthesischeck( string)  {  valid= true;  s= the empty stack;  while ( entire string is not read)  {  symb=next symbol of the string;  if (symb == ‘(‘ || symb == ‘{‘|| symb == ‘[‘)  push (symb)  else if (symb == ‘)‘ || symb == ‘}‘|| symb == ‘]‘)  {  i= pop ();  if ( i is not the matching opener of symb)  valid=false  } *//end else*  } *//end while*  if (stack s not empty)  valid=false  if (valid) print (“the string is valid”);  else print (“the string is not valid”);  } |
| **Procedure:**   1. Open CodeBlock editor or visual studio editor and write the code in C++. 2. Complile and run the code |
| **Instructions:**   1. Copy code & paste in code section and output of Part B. |
|  |

| **Part B** |
| --- |
| **Code:**  **Bracket.java**  **public class Bracket {**  **Stack stack;**  **public Bracket(int capacity) {**  **this.stack = new Stack(capacity);**  **}**  **public boolean areBracketsMatched(String s) {**  **char[] chars = s.toCharArray();**  **for (int i = 0; i < chars.length; i++) {**  **char c = chars[i];**    **if (c == '(' || c == '{' || c == '[') {**  **stack.push(c);**  **stack.display();**  **}**  **else if (c == ')') {**  **if (stack.isEmpty() || stack.pop() != '(') {**  **return false;**  **}**  **stack.display();**  **} else if (c == '}') {**  **if (stack.isEmpty() || stack.pop() != '{') {**  **return false;**  **}**  **stack.display();**  **} else if (c == ']') {**  **if (stack.isEmpty() || stack.pop() != '[') {**  **return false;**  **}**  **stack.display();**  **}**  **}**    **return stack.isEmpty();**  **}**  **public static void main(String[] args) {**  **Bracket bracketChecker = new Bracket(100);**  **String input = "{[(]}";**  **boolean result = bracketChecker.areBracketsMatched(input);**  **System.out.println(result);**  **}**  **}**  **Stack.java**  **public class Stack {**  **private char[] stackArray;**  **private int top;**  **private int capacity;**  **public Stack(int capacity) {**  **this.capacity = capacity;**  **this.stackArray = new char[capacity];**  **this.top = -1;**  **}**  **public void push(char element) {**  **if (top == capacity - 1) {**  **System.out.print("Stack is full");**  **} else {**  **stackArray[++top] = element;**  **}**  **}**  **public char pop() {**  **if (top == -1) {**  **return ' ';**  **} else {**  **return stackArray[top--];**  **}**  **}**  **public boolean isEmpty() {**  **return top == -1;**  **}**  **public void display() {**  **for (int i = 0; i <= top; i++) {**  **System.out.print(stackArray[i] + " ");**  **}**  **System.out.println();**  **}**  **}** |
| **Output:** |
| **Observation & Learning:**  learnt about the stack operations and iterating through a stack |
| **Answer Following Question**  Simulate the parenthesis checker algorithm for each of the following strings by showing the contents of the stack at each point-   1. {A + (B – C)}   the number signifies the char index  stack: 0→ { 1→ {  2→ {  3→ {(  4→ {(  5→ {(  6→ {(  7→ {  8→  since empty stack, brackets match   1. (A + B)) \* C   the number signifies the char index  stack: 0→ ( 1→ (  2→ (  3→ (  4→ false |
| **Conclusion:**  We successfully implemented Java program for parenthesis check using stack |