**Experiment 3**

(PART B: TO BE COMPLETED BY STUDENTS)

**(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)**

|  |  |
| --- | --- |
| Roll No. C013 | Name: Ashmit Jain |
| Program : BTI | Division: B |
| Batch: B1 | Date of Experiment: 8/9/2024 |
| Date of Submission: 8/9/2024 | Grade : |

* 1. **Tasks given in PART A to be completed here**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**class Stack {**

**private:**

**int size;**

**int top;**

**char\* stackArray;**

**public:**

**// Constructor**

**Stack(int maxSize) {**

**size = maxSize;**

**top = -1;**

**stackArray = new char[size];**

**}**

**// Destructor**

**~Stack() {**

**delete[] stackArray;**

**cout << "Stack has been destroyed" << endl;**

**}**

**// Check if the stack is empty**

**bool isEmpty() {**

**return top == -1;**

**}**

**// Check if the stack is full**

**bool isFull() {**

**return top == size - 1;**

**}**

**// Push an element onto the stack**

**void push(char ch) {**

**if (isFull()) {**

**cout << "Stack Overflow\n";**

**} else {**

**stackArray[++top] = ch;**

**cout << "Element pushed: " << ch << endl;**

**}**

**}**

**// Pop the topmost element from the stack**

**void pop() {**

**if (isEmpty()) {**

**cout << "Stack Underflow\n";**

**} else {**

**cout << "Element popped: " << stackArray[top--] << endl;**

**}**

**}**

**// Peek at the topmost element without popping it**

**char peek() {**

**if (isEmpty()) {**

**cout << "Stack is empty\n";**

**return '\0';**

**} else {**

**cout << "Topmost element: " << stackArray[top] << endl;**

**return stackArray[top];**

**}**

**}**

**};**

**// Function to check if the braces in the string are balanced**

**bool balancedBraces(string str) {**

**Stack stk(str.length());**

**for (char ch : str) {**

**if (ch == '(' || ch == '{' || ch == '[') {**

**stk.push(ch);**

**} else if (ch == ')' || ch == '}' || ch == ']') {**

**if (stk.isEmpty()) {**

**cout << "Unbalanced Braces" << endl;**

**return false;**

**}**

**char topElement = stk.peek();**

**if ((ch == ')' && topElement == '(') ||**

**(ch == '}' && topElement == '{') ||**

**(ch == ']' && topElement == '[')) {**

**stk.pop();**

**} else {**

**cout << "Unbalanced Braces" << endl;**

**return false;**

**}**

**}**

**}**

**if (stk.isEmpty()) {**

**cout << "Balanced Braces" << endl;**

**return true;**

**} else {**

**cout << "Unbalanced Braces" << endl;**

**return false;**

**}**

**}**

**int main() {**

**string str;**

**cout << "Enter the string: ";**

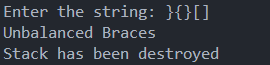
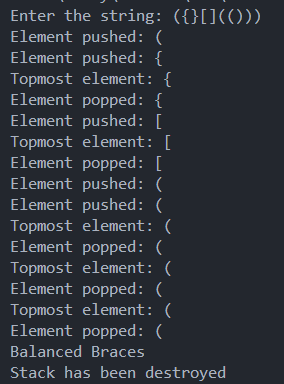
**cin >> str;**

**balancedBraces(str);**

**return 0;**

**}**

* 1. **Output /Observations**

****

### Observation:

###### Class Structure and Methods:

###### The Stack class is implemented with basic stack operations: push, pop, peek, isEmpty, and isFull.

###### The constructor dynamically allocates memory for the stack array based on the length of the input string.

###### The balancedBraces function is responsible for checking if the braces in a string are balanced using the stack.

###### Balanced Brace Check Logic:

###### The string is traversed character by character.

###### Opening braces ((, {, [) are pushed onto the stack.

###### For each closing brace (), }, ]), the stack is checked to see if it has the corresponding opening brace at the top. If it does, that brace is popped from the stack.

###### If at any point the closing brace does not match the opening brace or the stack is empty when it shouldn't be, the string is deemed unbalanced.

###### At the end of the traversal, if the stack is empty, the braces are balanced; otherwise, they are unbalanced.

###### Output Messages:

###### The program prints whether the braces are balanced or not, based on the operations performed during the string traversal.

### Conclusion:

###### The code effectively demonstrates how to use a stack data structure to solve the problem of checking balanced braces in a string. By pushing opening braces onto the stack and popping them when corresponding closing braces are encountered, the stack helps ensure that every opening brace has a matching closing brace in the correct order.

###### The separation of the Stack class and the balancedBraces function provides a clear and modular approach to solving this problem. The program handles both underflow (attempting to pop from an empty stack) and overflow (attempting to push onto a full stack) scenarios gracefully, although in this specific context of brace checking, overflow is unlikely to occur given the string's length determines the stack size.

###### This approach can be expanded to handle more complex scenarios involving other types of delimiters or more intricate validation rules.

###### 4o

###### 