Project Title: Bracket Parenthesis Checker

Table of Contents

1.Introduction:

Balanced parentheses are essential in programming to ensure proper syntax and logical execution of expressions. This project focuses on validating whether an input expression contains correctly nested and matched brackets (round, square, and curly). It showcases how stack data structures can be applied in real-world scenarios such as compiler design and expression validation. This project focuses on validating whether an input expression contains correctly nested and matched brackets (round (), square [], and curly {}).

The core idea is to use a **stack data structure**, where opening brackets are pushed onto the stack and, upon encountering a closing bracket, the program checks whether it matches the most recent opening bracket. If all brackets are matched correctly and the stack is empty at the end, the expression is valid.

This concept is widely applied in real-world scenarios such as:

- **Compiler Design**: Ensuring that source code is syntactically correct before converting it into machine-level instructions.
- **Expression Parsing**: Validating mathematical expressions or logical statements in calculators and interpreters.
- **Code Editors & IDEs**: Automatically highlighting mismatched parentheses or suggesting corrections to developers.

- **XML/HTML Validation**: Ensuring tags are properly nested and closed in structured documents.

2.Team Members:

S. No.	Name	Roll Number	Responsibility
1	Vijay Kumar	35	Input handling & bracket validation logic
2	Bharath	25	Stackimplementation(push,pop,peek empty)
3	Veera	33	Algorithm design (parenthesis matching)
4	yadeedya	21	Error Detection and Debugging
5	Karthik	19	File handling (optional)
6	vinay	11	Testing & documentation

3.Objective:

The main objective is to create a C program capable of: 1. Accepting expressions from users as input. 2. Utilizing stack operations to validate balanced parentheses. 3. Handling multiple and nested brackets efficiently.

4.Problem Statement:

Checking parentheses manually is error-prone, especially for lengthy and complex expressions with multiple levels of nesting. This project addresses this by automating the validation process using stack-based algorithms, ensuring accuracy and efficiency.

5.System Requirements:

Software Requirements: -

Turbo C++ / GCC Compiler - Any Text Editor (VS Code, Code::Blocks)

Hardware Requirements: -

Minimum 2GB RAM - 1.0 GHz Processor - Keyboard & Monitor

6.Data Structures Used:

Stack (implemented using arrays): Used for pushing and popping brackets during validation

- The primary data structure used in this project is a **Stack**, implemented with arrays.
- A stack follows the **LIFO (Last In, First Out)** principle, which is ideal for checking balanced brackets.
- **Push Operation:** When an opening bracket (, {, or [is encountered, it is **pushed** onto the stack.
- **Pop Operation:** When a closing bracket), }, or] is found, the program **pops** the top element from the stack and checks if it matches the corresponding opening bracket.
- If at the end of traversal the stack is empty, the expression is balanced; otherwise, it is unbalanced.

7.Modules and Description:

Module	Description
Input Module	Accepts an expression from the user
Validation Module	Implements stack operations to check brackets
Display Module	Shows whether the expression is balanced or not
Error Handling Module	Displays error for mismatched/unbalanced brackets

8.Algorithms Used:

The algorithm follows a stack-based approach:

1. Traverse the expression character by character.
2.Push opening brackets onto the stack.
3.On encountering a closing bracket, check the top of the stack: - If it matches, pop it. - If not, expression is invalid.
4. After traversal, if stack is empty → expression is balanced; else → unbalanced.

9.Sample I/O:

Example 1:
Input: { [(a+b) * c] + d }
Output: Expression is balanced

Example 2: Input: (a+b) * c
Output: Expression is NOT balanced

10.Test Cases:

Test Case ID	Input Expression	Expected Output
TC_01	(a+b)	Balanced
TC_02	{a+b*(c-d)}	Balanced
TC_03	[a+b*(c-d)]}	NOT Balanced
TC_04	((a+b)	NOT Balanced
TC_05	Empty Input	Invalid Expression

11.Future Enhancements:

-> Extend functionality to validate HTML/XML tags.

->Provide detailed error messages including mismatch position.

->Develop a GUI-based interface for better usability.

12.Conclusion:

The project demonstrates how stack operations are useful in solving real-world problems like syntax checking. It also strengthens understanding of fundamental data structures and algorithms commonly used in compilers and interpreters.

13.References:

1. Let Us C by Yashavant Kanetkar

2. www.geeksforgeeks.org

3. TutorialsPoint – C Programming Tutorials

4. Visual Guide to Stack Applications in C