Project Report: Simple Calculator Using Data Structures

Abstract

This project presents the design and development of a Simple Calculator application using data structures, primarily stacks. The calculator is capable of performing basic arithmetic operations such as addition, subtraction, multiplication, and division, and supports the evaluation of infix mathematical expressions by converting them to postfix (Reverse Polish Notation) format. The implementation emphasizes how foundational data structures can be applied to real-world computational problems, providing accuracy, clarity, and efficient computation.

1. Introduction

In the digital age, calculators are essential tools for performing mathematical operations, both in academic and professional settings. This project demonstrates how basic data structures like stacks can be used to build a fully functional calculator that evaluates user input expressions correctly and efficiently. The calculator accepts input in infix notation (e.g., 3 + 4 * 2), converts it to postfix, and evaluates the result using a stack-based algorithm.

2. Objectives

• To design and implement a calculator that supports basic arithmetic operations.

• To convert infix expressions to postfix notation using a stack.

• To evaluate postfix expressions using stack operations.

• To understand and apply the concept of operator precedence and associativity.

3. Algorithms and Data Structures Used

3.1 Stack (Primary Data Structure)

A stack is a linear data structure that follows the LIFO (Last In, First Out) principle. It is widely used in parsing and evaluating mathematical expressions.

3.2 Expression Conversion: Infix to Postfix

The algorithm processes each character of the expression, appending operands to output and using a stack for operators. Parentheses are handled with stack pushes and pops.

3.3 Postfix Expression Evaluation

Operands are pushed onto the stack. Operators pop two operands, apply the operation, and push the result back. The final value on the stack is the answer.

4. Implementation Details

Supported operations include addition (+), subtraction (-), multiplication (*), and division (/). The complexity of expression parsing and evaluation is linear with respect to the number of tokens.

5. Advantages of Using Stack in Calculator

- Efficient handling of operator precedence and associativity.
- Simple resolution of nested parentheses.
- Fast and accurate expression evaluation.
- Clear separation of parsing and computation.

6. Sample Execution

Input (Infix Expression): (3 + 5) * 2 - 4 / 2

Converted Postfix: 3 5 + 2 * 4 2 / -

Final Output: 14

7. Working Mechanism of the Calculator

The calculator functions by first taking a user-input expression in infix format. This expression is parsed character by character to distinguish between operands, operators, and parentheses.

The stack plays a crucial role in maintaining the order of operations. Operators are pushed to the stack only when their precedence is higher than the operators currently on top. This helps maintain proper evaluation order.

For example, multiplication and division have higher precedence than addition and subtraction. If an expression contains both, the calculator ensures the correct operation order without explicitly needing nested conditions.

Once the infix expression is fully traversed and converted into postfix form, evaluation becomes straightforward. Each operand is pushed to the operand stack, and whenever an operator appears, the top two values are popped, evaluated, and the result is pushed back onto the stack.

8. Real-life Applications of Stack-based Calculators

1. **Scientific Calculators**: Many advanced calculators use stack-based methods for evaluating complex expressions, especially those involving functions like sin, cos, tan, etc.

2. **Programming Language Compilers**: Compilers use stacks for syntax parsing and expression evaluation.

3. **Expression Solvers in Mobile Apps**: Apps that evaluate or graph mathematical expressions often use postfix evaluation for speed.

4. **Spreadsheet Software**: Programs like MS Excel or Google Sheets evaluate cell formulas using stack-based approaches.

9. Limitations of the Current Model

While the current calculator handles basic arithmetic, it does not support:

- Floating-point arithmetic with high precision

- Trigonometric or logarithmic functions

- Variable support (e.g., 'x + y')

- Error handling for malformed expressions

- Multi-digit numbers with complex formatting (like 12.45 or -23.9)

Addressing these limitations would make the calculator much more robust and ready for real-world applications.

10. Conclusion

The Simple Calculator project illustrates the application of stack data structures in a practical and meaningful way. It effectively handles arithmetic operations and complex expressions while ensuring accurate evaluation through infix-to-postfix conversion and stack-based computation.

11. Future Enhancements

• Add support for decimal numbers and floating-point operations.

• Implement modulus, exponentiation, and square root functions.

• Develop a graphical user interface (GUI).

• Add error handling for invalid inputs and division by zero.

• Extend to scientific calculator features (log, sin, cos, etc.).

12. References

• Data Structures and Algorithms by Alfred Aho, Jeffrey Ullman

• GeeksforGeeks: Stack Data Structure and Expression Evaluation

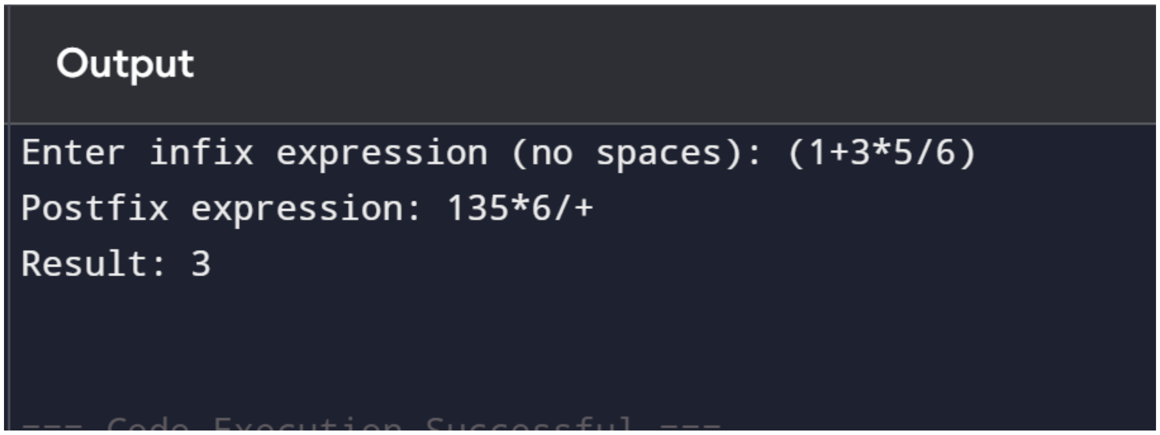
• https://en.wikipedia.org/wiki/Reverse_Polish_notation

•https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.html

13. Github Links:

https://github.com/vishwesh7086/simple-calculator-minor-project-Data-Structure

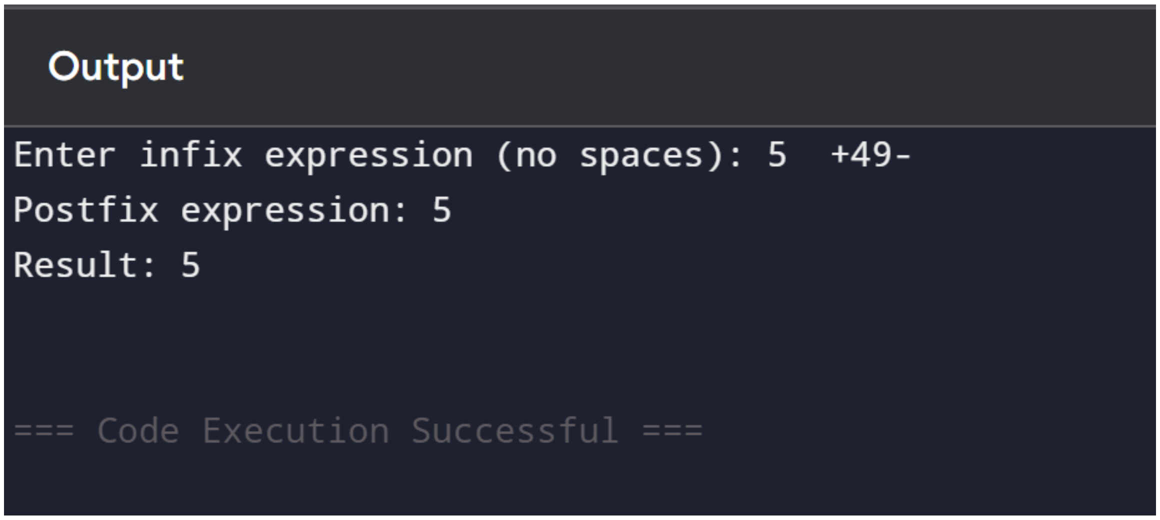
14. Execution Screenshots:



Output

```
Enter infix expression (no spaces): (1+3*5/6)
Postfix expression: 135*6/+
Result: 3


=== Code Execution Successful ===
```



Output

```
Enter infix expression (no spaces): 5  +49-
Postfix expression: 5
Result: 5


=== Code Execution Successful ===
```