**VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY**

**INTERNATIONAL UNIVERSITY**



**ALGORITHMS & DATATRUCTURES (IT013IU)**

Course by

**Dr. Vi Chi Thanh and MSc. Thai Trung Tin**

|  |  |
| --- | --- |
| **Name** | **ID** |
| Lê Quang Dũng | ITDSIU23031 |
| Bùi Lê Anh Khoa | ITDSIU23009 |

**Table of Contents**

Abstract

1. Introduction
2. Idea for the Report: The Polish Notation (Pháp Ký Ba Lan)
3. Objectives
4. Program Structure and Design
5. Expression Evaluation Logic
6. Graphical User Interface (GUI)
7. Undo Feature
8. Key Algorithms
9. Infix to Postfix Conversion
10. Postfix Evaluation
11. Undo Implementation
12. User Interface Design
13. Data Structures

7.1 Stack

1. Time Complexity Analysis

8.1 Infix to Postfix Conversion

8.2 Postfix Evaluation

8.3 Undo Feature

1. Challenges and Solutions

9.1 Parsing Complex Expressions

9.2 Undo Feature

1. Testing and Validation
2. Results and Discussion

12.1 Strengths

12.2 Limitations

1. Conclusion and Future Work
2. References

**Developer team**

|  |  |
| --- | --- |
| **Name** | **Job** |
| Lê Quang Dũng | Designed and implemented the infix to postfix conversion algorithm. Conducted testing and validation of the program. Prepared visually engaging slides explaining the project overview, key features, algorithms, and results |
| Bùi Lê Anh Khoa | Worked on the Undo feature and the GUI design. Developed the postfix evaluation logic and integrated it with the main program. Wrote the report, documenting all aspects of the project, including the introduction, algorithm explanations, testing results, and future enhancements. |

**Abstract**

The report describes a Complete Calculator application built in Java. The program has GUI implemented in Swing and can evalulate infix notation, convert to postfix notation and calculate results. The report details the program’s architecture, core algorithms, user interface, and new features such as the Undo mechanism. It ends with an assessment of how the program is working — and how it could work better. The report also includes details on the algorithms, time complexity and data structures used in the program

**1. Introduction**

A calculator is one of the most basic useful apps in computing. Project: Complete Calculator in Java, to Parse, Evaluate, and give output in Infix Mathematical Expression. The application aimed to mimic your average calculator, all the while demonstrating more complex programming concepts such as mathematics evaluations, stacks and the graphical user interface (GUI). Adding the ability to undo the most recent action and handling expression errors improves both usability and robustness.

**2. Idea for the Report: The Polish Notation (Pháp Ký Ba Lan)**

This report considers the principles of Polish Notation that was introduced by Polish mathematician Jan Lukasiewicz. To Polish Notation, the concept of parenthesis is irrelevant, since one can prefix or postfix the operators to the respective operands. This method of structuring enables easy computation of expressions, particularly in a computing environment.

In this project, the implementation is based on the conversion of infix to postfix, which is then done using the Shunting-Yard algorithm. Postfix notation makes expression evaluation easier since there are no problems of operator precedence, as every operator is processed in the order it was written. The use of RPN is also the main feature of a calculator, as it makes the whole process more efficient and easier to work with.

**Polish Notation (Pháp Ký Ba Lan) in Depth**

Polish Notation, introduced by Jan Lukasiewicz, eliminates ambiguity in mathematical expressions by standardizing the order of operations. There are two types:

* **Prefix (Polish) Notation:** Operators precede operands (example: + 3 5).
* **Postfix (Reverse Polish) Notation:** Operators follow operands (example 3 5 +).

**Example Conversion:**

Given the infix expression:  
(3 + 5) \* (2 - 1)

**Step-by-Step Conversion to Postfix:**

1. Start with the innermost parentheses.
   * Convert (3 + 5) → 3 5 +.
   * Convert (2 - 1) → 2 1 -.
2. Apply the \* operator:  
   Result → 3 5 + 2 1 - \*.

**Postfix Expression:** 3 5 + 2 1 - \*

**Advantages of RPN in Evaluation**

Using a stack, the evaluation becomes straightforward:

1. Push operands onto the stack.
2. When an operator is encountered, pop the required operands, compute the result, and push it back onto the stack.

**Evaluation Steps for 3 5 + 2 1 - \*:**

1. Push 3, 5 → Stack: [3, 5]
2. Encounter +: Pop 5, 3; Compute 3 + 5 = 8; Push 8 → Stack: [8]
3. Push 2, 1 → Stack: [8, 2, 1]
4. Encounter -: Pop 1, 2; Compute 2 - 1 = 1; Push 1 → Stack: [8, 1]
5. Encounter \*: Pop 1, 8; Compute 8 \* 1 = 8; Push 8 → Stack: [8]

**Result:** 8

**Comparison of Notations**

|  |  |  |  |
| --- | --- | --- | --- |
| **Notation** | **Example** | **Description** | **Advantages** |
| Infix | (3 + 5) \* 2 | Standard human-readable notation. | Easy for humans to write. |
| Prefix | \* + 3 5 2 | Operators come before operands. | Compact and eliminates ambiguity. |
| Postfix | 3 5 + 2 \* | Operators come after operands. | Simplifies computational parsing. |

By integrating this methodology, the Complete Calculator showcases the practical application of Polish Notation in solving real-world computational problems. The implementation not only demonstrates the power of algorithmic thinking but also highlights the elegance of using stacks to manage intermediate operations.

**3. Objectives**

The primary goals of the project include:

1. Implementing a calculator capable of evaluating infix expressions using the postfix conversion algorithm.
2. Creating a user-friendly GUI with Swing.
3. Enabling additional features such as undoing the last input and handling invalid expressions gracefully.
4. Ensuring modular and maintainable code for future extensibility.

**4. Program Structure and Design**

The calculator program is divided into three main components:

**4.1 Expression Evaluation Logic**

The evaluation of mathematical expressions follows these steps:

1. **Conversion of Infix to Postfix**: The program uses the Shunting-Yard algorithm to rearrange an infix expression into postfix notation. This ensures proper operator precedence and allows easier evaluation.
2. **Postfix Evaluation**: A stack-based approach is used to evaluate the postfix expression by processing operators and operands sequentially.

**4.2 Graphical User Interface (GUI)**

The GUI is implemented using Java Swing components:

* A JTextField for user input.
* A JTextArea for displaying results and postfix conversion.
* A JPanel with buttons for numbers, operators, and functions (e.g., Clear, Undo, Equals).

**4.3 Undo Feature**

The undo functionality is achieved by maintaining a stack of input states. Users can revert to the previous state of the expression with a single click.

**5. Key Algorithms**

**5.1 Infix to Postfix Conversion**

The program converts infix expressions to postfix notation using the following logic:

1. Traverse the expression character by character.
2. Push operands directly to the output.
3. Push operators to a stack, ensuring precedence rules are followed.
4. Pop operators from the stack to the output when encountering a closing parenthesis or a lower-precedence operator.

**5.2 Postfix Evaluation**

To evaluate the postfix expression:

1. Push operands onto a stack.
2. When an operator is encountered, pop the top two elements from the stack, apply the operator, and push the result back onto the stack.
3. The final result is obtained when the stack contains a single element.

**5.3 Undo Implementation**

The Undo feature uses a Stack<String> to store previous states of the input field. Each modification to the input is saved, allowing the user to revert to the last saved state efficiently.

**6. User Interface Design**

The GUI was designed with usability in mind. It consists of three main sections:

1. **Input Field**: A JTextField for users to input expressions. It is right-aligned for a calculator-like feel.
2. **Output Area**: A JTextArea displays the postfix notation and the evaluation result.
3. **Button Panel**: A JPanel containing buttons for digits, operators, and functions arranged in a grid layout.

**Button Layout**

|  |  |  |  |
| --- | --- | --- | --- |
| 7 | 8 | 9 | + |
| 4 | 5 | 6 | - |
| 1 | 2 | 3 | \* |
| Undo | 0 | = | / |
| C |  |  |  |

**7. Data Structures**

**7.1 Stack**

The stack is the primary data structure used in the program. It is utilized in the following areas:

1. **Infix to Postfix Conversion**: To temporarily store operators and parentheses.
2. **Postfix Evaluation**: To hold operands and intermediate results during evaluation.
3. **Undo Feature**: To maintain a history of input states for reverting changes.

**8. Time Complexity Analysis**

**8.1 Infix to Postfix Conversion**

The time complexity of this algorithm is **O(n)**, where n is the length of the expression. Each character is processed once, and stack operations (push and pop) are constant time.

**8.2 Postfix Evaluation**

The evaluation of a postfix expression also takes **O(n)** time, as each operand and operator is processed exactly once.

**8.3 Undo Feature**

The undo operation involves pushing or popping from a stack, which is **O(1)** in time complexity.

**Overall Complexity**

For evaluating an expression (infix to postfix conversion + postfix evaluation), the total time complexity is **O(n)**.

**9. Challenges and Solutions**

**9.1 Parsing Complex Expressions**

* **Challenge**: Handling operator precedence and parentheses.
* **Solution**: The Shunting-Yard algorithm ensures correct parsing.

**9.2 Undo Feature**

* **Challenge**: Implementing state management for user inputs.
* **Solution**: A Stack<String> is used to save and restore previous states efficiently.

**10. Testing and Validation**

The program was tested with various inputs to ensure correctness:

1. Simple expressions: 3+5, 10-2.
2. Expressions with precedence: 3+5\*2, (10-2)/4.
3. Invalid inputs: 3++5, 10/(2-2).

All tests produced correct results or appropriate error messages.

**11. Results and Discussion**

The Complete Calculator meets its design objectives:

* Correctly evaluates mathematical expressions.
* Provides a user-friendly interface with essential functions.
* Includes additional features like Undo and error handling.

**11.1 Strengths**

1. Modular code design for easier maintenance.
2. Robust handling of invalid inputs.
3. Extensible GUI with features like Undo.

**11.2 Limitations**

1. Does not support advanced functions like trigonometry or logarithms.
2. Poor GUI

**12. Conclusion and Future Work**

The Complete Calculator successfully demonstrates the integration of GUI design and algorithmic problem-solving in Java. It provides a robust and user-friendly solution for evaluating mathematical expressions. Future improvements could include:

1. Adding advanced mathematical functions (e.g., sin, cos, tan).
2. Enhancing the UI with themes or better layouts.