DSA Calculator: Expression Evaluator & Converter with Trie-Based Autocomplete

# 1. INTRODUCTION

## 1.1 PURPOSE

The purpose of this project is to create a comprehensive DSA-based calculator in Java that supports conversion and evaluation of mathematical expressions. It includes infix, postfix, and prefix conversions, postfix evaluation with variables, expression tree generation, step-by-step evaluation, syntax error detection, and intelligent autocomplete using a Trie. This tool aims to help students and developers visualize and learn core DSA implementations through a file-based Java application.

## 1.2 DOCUMENT CONVENTIONS

The application is developed using Java syntax conventions. All logic is written using Java standard libraries, avoiding external packages. Inputs are read from text files and outputs are written to text files. Expressions and variables follow common mathematical and programming notation.

## 1.3 INTENDED AUDIENCE AND PROJECT MANUAL

This documentation is designed for academic evaluators, students, and developers interested in learning the application of Data Structures and Algorithms (DSA) in expression evaluation and symbolic computation. The manual explains system architecture, logic, and features for understanding and extending the project.

## 1.4 PROJECT SCOPE

The scope includes:  
\* Infix ↔ Postfix ↔ Prefix and vice versa conversions  
\* Postfix, Prefix, Infix evaluation with variable input  
\* Bitwise operation evaluation  
\* Expression tree construction and traversal  
\* Step-by-step evaluation output  
\* Trie-based autocomplete suggestions  
\* Error detection with correction suggestions  
\* History tracking and replay functionality  
\* File-based I/O for all user interactions

# 2. OVERALL DESCRIPTION

## 2.1 PRODUCT PERSPECTIVE

The DSA Calculator is a standalone Java application with no external dependencies. It serves both as a learning tool and a powerful utility for expression conversion and evaluation using core data structures implemented from scratch.

## 2.2 USER CLASS AND CHARACTERISTICS

\* Learner/User: Uses the calculator to test, convert, and evaluate expressions, understand DSA workflows.  
\* Developer/Contributor: Modifies or extends the codebase for educational or project submission purposes.

## 2.3 DESIGN AND IMPLEMENTATION CONSTRAINTS

\* No external libraries or frameworks  
\* All DSA implementations (Trie, Stack, Tree, etc.) are custom-written  
\* File-based input/output instead of console  
\* Uses primitive data types and arrays where applicable

# 3. SYSTEM FEATURES

## 3.1 FEATURE-TO-DSA LOGIC MAPPING

|  |  |  |
| --- | --- | --- |
| Feature | DSA Logic Used | Data Types / Structure Used |
| Infix-Postfix/Prefix Conversion | Stack | char[], String[], char[][] |
| Postfix Evaluation with Variables | Stack + Variable Map | int[], String[], HashMap |
| Bitwise Expression Evaluation | Stack | int[], char[] |
| Expression Tree | Binary Tree | Custom Node class using arrays |
| Autocomplete | Trie with frequency ranking | char[][], int[], boolean[] |
| Step-by-step Output | Recursive Logging | String[], int[] |
| History and Replay | Stack or Doubly Linked List | String[], int[], char[] |
| Error Detection & Suggestions | Validation Tree + Parser | String, custom error class |

## 3.2 SYSTEM FUNCTIONALITY

\* Convert between Infix, Prefix, Postfix expressions  
\* Evaluate postfix/bitwise expressions with variables  
\* Autocomplete partially typed operators/functions  
\* Show step-by-step evaluation of expressions  
\* Detect and suggest corrections for invalid expressions  
\* Build and display expression trees with traversals  
\* Track and replay previous expressions via history  
\* Input and output handled through files

## 3.3 NONFUNCTIONAL REQUIREMENTS

\* Platform-independent (Java SE 8+)  
\* Efficient runtime for large expressions  
\* Easily extendable modular design  
\* File-based human-readable input and output

# 4. EXTERNAL INTERFACE REQUIREMENTS

## 4.1 USER INTERFACES

Text files serve as the primary interface. Users input expressions and commands into a file (e.g., input.txt), and results are written to another file (e.g., output.txt).

## 4.2 HARDWARE INTERFACES

Runs on any standard machine with Java runtime installed (JDK 8 or later).

## 4.3 SOFTWARE INTERFACES

\* Java SE 8+  
\* Text editor and command-line interface

## 4.4 COMMUNICATION INTERFACES

None; this is an offline, standalone application.

# 5. REFERENCES

\* Java SE Documentation – https://docs.oracle.com/javase/8/  
\* GeeksForGeeks Expression Conversion Guide – https://www.geeksforgeeks.org/expression-conversion/  
\* Stack & Trie DSA Tutorials – https://www.tutorialspoint.com/data\_structures\_algorithms/index.htm  
\* Java Stack, Recursion, and Tree Basics – https://www.w3schools.com/java/