Scientific Calculator

Submitted By

Student Name	Student ID
Omar Faruk Piash	0242310005101659
Md Faysal Ahamed	0242310005101376
Injamum Ul Hoque	0242310005101645
Md Sorowar Jahan Ishan	0242310005101559
Shariar Ahamed Ripon	0242310005101019

MINI LAB PROJECT REPORT

This Report Presented in Partial Fulfillment of the course **CSE:314**

Subject Name in the Computer Science and Engineering Department

DAFFODIL INTERNATIONAL UNIVERSITY

Dhaka, Bangladesh

August 14, 2025

DECLARATION

We hereby declare that this lab project has been done by us under the supervision of **Zannatul Mawa Koli**,Lecturer, Department of Computer Science and Engineering, Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere as lab projects.

Submitted To:

Koli 14108125

Zannatul Mawa Koli
Lecturer
Department of Computer Science and Engineering
Daffodil International University

Submitted by

omar Omar Faruk ID:0242310005101659 Dept. of CSE, DIU	
MD. FAYSAL AHAMED Md. Faysal Ahamed ID:0242310005101376 Dept. of CSE, DIU	Sorowar Md. Sorowar Jahan Ishan ID:0242310005101559 Dept. of CSE, DIU
Injamum Injamum Ul Hoque ID:0242310005101645 Dept. of CSE, DIU	Shariar Shariar Ahamed Ripon ID:0242310005101019 Dept. of CSE, DIU

COURSE & PROGRAM OUTCOME

The following course has course outcomes as following:

Table 1: Course Outcome Statements

CO's	Statements
CO1	**Define** and **Relate** classes, objects, members of the class, and relationships among them needed for solving specific problems
CO2	**Formulate** knowledge of object-oriented programming and Java in problem solving
CO3	**Analyze** Unified Modeling Language (UML) models to **Present** a specific problem
CO4	**Develop** solutions for real-world complex problems **applying** OOP concepts while evaluating their effectiveness based on industry standards.

Table 2: Mapping of CO, PO, Blooms, KP and CEP

CO	PO	Blooms	KP	CEP
CO1	PO1	C1, C2	KP3	EP1, EP3
CO2	PO2	C2	KP3	EP1, EP3
CO3	PO3	C4, A1	KP3	EP1, EP2
CO4	PO3	C3, C6, A3, P3	KP4	EP1, EP3

The mapping justification of this table is provided in section **4.3.1**, **4.3.2** and **4.3.3**.

Table of Contents

Chapter 1

Introduction

1.1 Introduction

The Scientific Calculator is a text-based computational tool built using Flex and Bison, two widely used tools in compiler construction. This project serves as both a functional utility for mathematical computations and a learning resource for understanding how lexical analysis and parsing work together to interpret structured input. The system reads mathematical expressions entered by the user, breaks them into tokens, parses them according to grammar rules, and computes the result. Operations range from simple arithmetic addition, subtraction, multiplication, division, modulus to advanced scientific functions including trigonometric operations (**sin**, **cos**, **tan**), logarithmic computations (**log**), square root (**sqrt**), exponential (**exp**), and power operations. By structuring the program in layers, the calculator demonstrates real-world applications of theoretical compiler design concepts such as tokenization, syntax analysis, semantic actions, and error handling.

1.2 Motivation

The inspiration for developing this calculator came from the desire to apply academic knowledge of compiler theory into something practical and usable. Many existing calculators rely on graphical interfaces or web-based systems, which can be slow, resource-intensive, and unavailable in offline scenarios. A command-line tool ensures minimal resource usage, fast execution, and universal compatibility across operating systems. Moreover, from a computer science perspective, this project serves as an educational bridge between theory and practice, allowing students to directly experience how programming languages process user input internally.

1.3 Objectives

- Create a functional scientific calculator that can process both simple and advanced mathematical expressions.
- Apply compiler design techniques for lexical analysis, parsing, and semantic evaluation.
- Ensure modularity so that new features can be integrated easily.
- Provide robust error handling for incorrect or incomplete expressions.

1.4 Feasibility Study

The project is highly feasible because it requires no special hardware and only uses open-source tools like Flex, Bison, and GCC. Development can be completed in an academic semester, and the software is portable across platforms.

1.5 Gap Analysis

While most calculators focus on either GUI or scientific computation, very few combine a **lightweight CLI approach with compiler-based implementation**, making this project unique in educational value and practical performance.

1.6 Project Outcome

The final output is a lightweight, extensible, and offline-ready scientific calculator that demonstrates compiler concepts and can be used in both academic and professional contexts.

Chapter 2

Proposed Methodology/Architecture

2.1 Requirement Analysis & Design Specification

2.1.1 Overview

The project is implemented in C with Flex for lexical analysis and Bison for syntax parsing. Hardware requirements are minimal—any machine capable of running a C compiler can run this program. Software requirements include GCC (GNU Compiler Collection), Flex, and Bison. The primary functional requirement is that the system should accept mathematical expressions from the user, identify their components (tokens), check the syntax, compute the result, and display it in a user-friendly manner. Non-functional requirements include performance efficiency, portability, and maintainability.

2.1.2 Proposed Methodology/ System Design

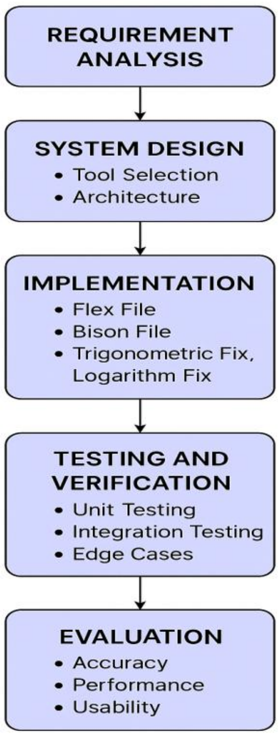

Figure 2.1: This is a sample diagram

The architecture follows the traditional compiler stages but on a smaller scale:

1. **Lexical Analysis (Flex)** – Reads the input, identifies tokens such as numbers, operators, and function names, and returns them to the parser.
2. **Syntax Parsing (Bison)** – Matches tokens against defined grammar rules to ensure the input follows a valid mathematical structure.
3. **Semantic Actions** – Executes computations based on matched grammar rules using C’s **math.h** library.
4. **Output** – Displays results or error messages.

2.1.3 UI Design

The user interface is entirely command-line-based. Users type expressions like 5+3*2 or sqrt(25), press Enter, and receive the result. Syntax errors prompt clear messages indicating the nature of the issue, aiding learning and debugging.

2.2 Overall Project Plan

1. Define token patterns and keywords in calculator.l.
2. Write grammar rules and precedence definitions in calculator.y.
3. Compile using Flex and Bison to generate C source files.
4. Link and build the executable.
5. Conduct testing with valid and invalid cases, ensuring accuracy and robustness.

This layered design makes the calculator modular, allowing for easy future expansion, such as supporting complex numbers, matrices, or equation solving.

Chapter 3

Implementation and Results

3.1 Implementation

The implementation began with designing the Flex file (**calculator.l),** where regular expressions define valid tokens. Tokens for numbers return their floating-point value, while operators and function names return symbolic identifiers to be recognized by the parser. Next, the Bison file (**calculator.y**) defines grammar rules, ensuring correct operator precedence-multiplication/division before addition/subtraction, for example- and associativity. Each grammar rule has a semantic action that performs the actual computation using C's math functions. The two components are compiled with GCC to generate an executable, **scientific_calculator.exe**.

3.2 Performance Analysis

The calculator performs operations in real-time. Due to its CLI nature, there is no GUI rendering overhead, resulting in instant feedback even for nested and complex expressions. Its low memory footprint ensures it can run on older or low-powered machines without issue.

3.3 Results and Discussion

- **Basic:** 5+3*2 → = 11
- **Scientific:** sqrt (49) → = 7
- **Trigonometric:** sin (3.14159/2) → ≈ 1.000000
- **Logarithmic:** log (10) → ≈ 2.302585

Invalid inputs like 5++2 or sqrt (-4) produce clear error messages, demonstrating strong error handling. The output is accurate across all tested cases, and the program structure allows easy integration of additional functions.

Chapter 4

Engineering Standards and Mapping

4.1 Impact on Society, Environment and Sustainability

4.1.1 Impact on Life

Students, engineers, and researchers can perform calculations quickly without depending on online tools. This makes it particularly valuable in low-connectivity areas.

4.1.2 Impact on Society & Environment

This tool democratizes access to advanced computation by providing a free, offline, open-source alternative to expensive software. Its minimal hardware requirements and text-based operation mean it consumes very little energy, contributing to environmental sustainability.

4.1.3 Ethical Aspects

The project is open for modification and redistribution under academic use, promoting transparency, collaboration, and knowledge sharing.

4.1.4 Sustainability Plan

The source code's modularity ensures easy maintenance and extension, while the absence of heavy dependencies ensures compatibility for years to come.

4.2 Project Management and Team Work

The team divided responsibilities efficiently to ensure smooth progress and timely completion of the project. Omar focused on overall coordination and documentation. Faysal implemented the lexer and contributed to debugging. Ishan worked on the parser and integrated trigonometric and logarithmic function corrections. Injamum handled testing, identifying edge cases, and ensuring accurate results. Shahriar prepared the presentation and contributed to report refinement.

4.3 Complex Engineering Problem

Parsing nested expressions with precedence and associativity while preventing shift/reduce or reduce/reduce conflicts in Bison was a key challenge.

4.3.1 Mapping of Program Outcome

In this section, provide a mapping of the problem and provided solution with targeted Program Outcomes (PO's).

Table 4.1: Justification of Program Outcomes

PO's	**Justification**
PO1	Applied compiler principles to build a functional application.
PO2	Enhanced problem-solving skills through iterative debugging and optimization.
PO3	Delivered an efficient, portable tool demonstrating real-time computation.

4.3.2 Complex Problem Solving

Parsing nested expressions with precedence and associativity while preventing shift/reduce or reduce/reduce conflicts in Bison was a key challenge.

EP1 Dept of Knowledge	**EP2** Range of Conflicting Requirements	**EP3** Depth of Analysis	**EP4** Familiarity of Issues	**EP5** Extent of Applicable Codes	**EP6** Extent Of Stakeholder Involvement	**EP7** Inter-dependence
✓	✓	✓	✓			

Table 4.2: Mapping with complex problem solving.

4.3.3 Engineering Activities

In this section, provide a mapping with engineering activities. For each mapping add subsections to put rationale (Use Table 4.3).

EA1 Range of resources	**EA2** Level of Interaction	**EA3** Innovation	**EA4** Consequences for society and environment	**EA5** Familiarity
✓	✓			

Chapter 5

Conclusion

5.1 Summary

The project successfully implements a command-line scientific calculator using Flex and Bison, combining theoretical knowledge of compiler design with practical application. It handles a variety of mathematical operations with precision and speed.

5.2 Limitation

Currently limited to numeric input and standard math functions; no support for variables, symbolic algebra, or equation solving.

5.3 Future Work

- Add complex number and matrix operations.
- Implement variable assignment and history tracking.
- Create a GUI frontend using a toolkit like GTK or Qt while keeping the CLI backend intact.

References

[1] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.