Bangla Programming Language Compiler

Submitted By

Student Name	Student ID
Jubayer Hossain	0242220005101790
Nazia Sultana Marjan	0242220005101802
Ismail Hossain Sifat	0242220005101757
Anisa Azad Oishy	0242220005101789
Al Nahian Habib	0242220005101808

LAB PROJECT REPORT

This Report Presented in Partial Fulfillment of the course

CSE332: Compiler Design Lab in the Department of

Computer Science and Engineering

DAFFODIL INTERNATIONAL UNIVERSITY

Dhaka, Bangladesh

11/12/2025

DECLARATION

We hereby declare that this lab project has been done by us under the supervision of **Mushfiqur Rahman Chowdhury**, **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:

_____

Mushfiqur Rahman Chowdhury
Lecturer
Department of Computer Science and Engineering
Daffodil International University

Submited by

Jubayer Hossain ID: 0242220005101790 Dept. of CSE	
Nazia Sultana Marjan ID: 0242220005101802 Dept. of CSE	Ismail Hossain Sifat ID: 0242220005101757 Dept. of CSE
Anisa Azad Oishy ID: 0242220005101789 Dept. of CSE	Al Nahian Habib ID: 0242220005101808 Dept. of CSE

Table of Contents

1. Introduction

Compiler design plays a vital role in the field of computer science, enabling the translation of high-level programming languages into machine-understandable instructions.

This project aims to design and implement a custom programming language written entirely using Bangla keywords, making programming more approachable for Bangla-speaking beginners. The project includes the development of a lexer (Flex) and parser (Bison) capable of interpreting Bangla syntax, performing type checking, executing statements, and displaying meaningful error messages. This Bangla Compiler supports integer, float, and string variables, expressions, loops, conditionals, I/O statements, and robust error handling.

1.1 Overview of the Bangla Programming Language Compiler

The Bangla Programming Language Compiler is a simplified compiler designed to translate Bangla-based programming instructions into executable C-based operations. The aim of this project is to make programming more intuitive for native Bangla speakers by allowing them to write logical instructions using familiar Bangla keywords such as dhoro, dekhao, jodi, cholo, jonno, and many more.

The compiler currently supports variable declaration, input handling, output generation, arithmetic expression evaluation, conditional statements, loops, and semantic error detection. It is built using lexical analysis (Flex), syntax analysis (Bison), and C-based interpretation logic.

1.2 Background of the Project

Bangla is one of the most widely spoken languages globally, yet programming tools and languages rarely support Bangla syntax. Many new learners struggle with English-based programming languages due to unfamiliar keywords and symbols.

Compiler Design is a fundamental topic in computer science, and implementing a custom compiler helps students understand the internal workings of programming languages—how code is scanned, parsed, and executed.

This project was developed as part of the Compiler Design course and aims to combine education with innovation by building a compiler using Bangla instructions.

1.3 Scope of the Work

The scope of the project includes:

- Designing a custom syntax using Bangla keywords
- Implementing a lexical analyzer to recognize tokens
- Implementing a parser to validate grammar rules
- Adding semantic analysis to detect type errors
- Supporting integer, float, and string data types
- Executing valid programs and producing correct output
- Handling comments, loops, conditions, and input commands

The compiler does not yet support functions, arrays, user-defined types, or advanced optimization, but these may be added in future improvements.

2. Motivation & Objectives

2.1 Motivation

The primary motivation behind developing this project is to reduce the language barrier for beginners in programming. By enabling students to write programs in their own mother tongue, learning becomes more intuitive and enjoyable. Additionally, building a compiler deepened our understanding of how real programming languages work internally.

2.2 Objectives

The project has been carried out to achieve the following objectives:

- To design a simple Bangla-keyword-based programming language
- To implement lexical analysis to recognize identifiers, keywords, strings, and numbers
- To develop a parser using Bison to validate language grammar
- To support variable declarations using integer, float, and string types
- To detect semantic type errors properly and generate meaningful error messages
- To allow loops, conditional statements, and expressions in Bangla syntax
- To create a compiler that is easy for beginners and Bangla speakers

3. Proposed Methodology & System Design

3.1 System Overview

The Bangla Programming Language Compiler converts Bangla-styled source code into executable behavior using a structured compilation process.

The system is built using Flex (for lexical analysis) and Bison (for syntax analysis), along with C code for runtime execution.

The workflow includes:

- Taking Bangla keywords and statements as input
- Tokenizing meaningful units (identifiers, numbers, strings, operators)
- Checking grammar structure using the parser
- Detecting semantic errors
- Executing valid statements sequentially

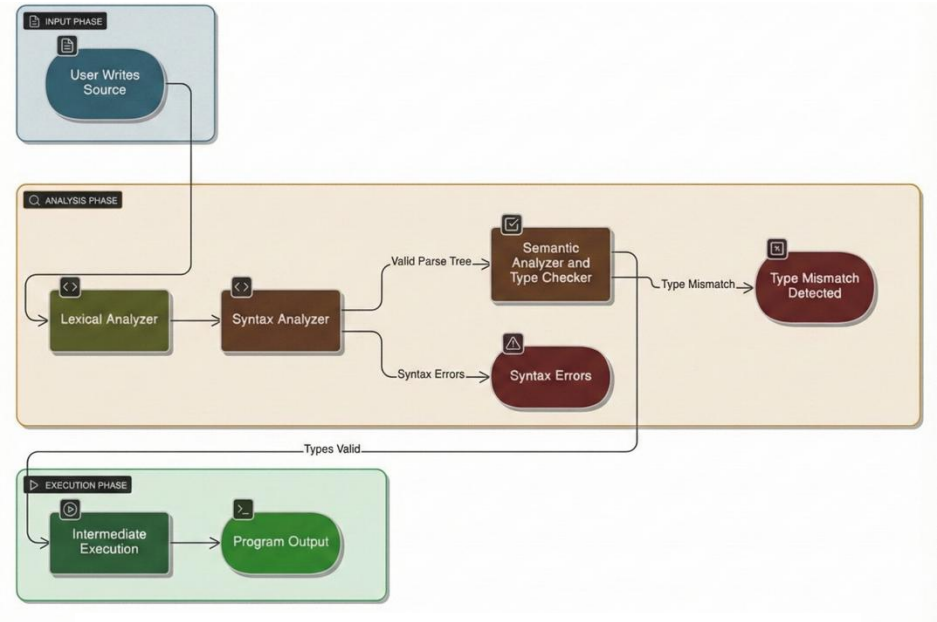


Figure: Workflow Diagram

The compiler ensures simple Bangla-language-based programming for beginners while maintaining the behavior of traditional procedural languages.

3.2 Architecture of the Compiler

The compiler architecture consists of the following major components:

- **Source Code (.ruby)** – User-written Bangla program
- **Lexical Analyzer (Flex)** – Converts sequences of characters into tokens
- **Parser (Bison)** – Checks grammar rules and builds an Abstract Syntax Tree (AST)
- **Semantic Analyzer** – Detects type mismatches and invalid assignments
- **Execution Engine** – Executes statements in order and prints outputs
- **Error Handler** – Displays meaningful Bangla error messages

The system works as a mini-compiler with support for variables, expressions, conditions, loops, and type-checking.

3.3 Lexical Analyzer (Lexer) Design

The lexer reads the input program character-by-character and identifies meaningful tokens.

Key responsibilities include:

- Recognizing Bangla keywords like (dhoro, dekhao, jodi, tahole, shes, cholo, jonno)
- Recognizing identifiers (variable names)
- Reading numeric values (integers and floats)
- Reading string literals
- Skipping whitespace
- Supporting single-line comments using # and //
- Passing token types to the parser

The lexer ensures that invalid symbols are caught early and reported clearly.

3.4 Syntax Analyzer (Parser) Design

The parser ensures that the structure of the Bangla program follows the defined grammar.

Using Bison grammar rules, the parser:

- Builds AST nodes for expressions and statements
- Validates assignment formats
- Confirms correct order of keywords (e.g., IF … THEN … END)
- Supports nested statements, loops, and conditions

The parser also triggers error messages when invalid syntax is detected, including:

- Missing END
- Wrong ordering of keywords
- Unexpected tokens
- Incomplete statements

3.5 Semantic Checking and Error Handling

After syntax verification, semantic analysis ensures that the meaning of statements is valid.

This includes:

- **Type Checking**

Integer cannot be assigned a float (dhoro int a = 3.3)

Float cannot be assigned a string (dhoro float b = "compiler")

String cannot be assigned an integer without quotes (dhoro string name = 5)

- **Variable Initialization Checking**

Detect writing or printing a variable before assignment

- **Loop Condition Validation**

Ensures FOR and CHOLO loops have proper limits

- **Accumulating Multiple Errors**

Instead of stopping at the first error, the compiler collects all errors and displays them at the end.

3.6 System Design Diagram

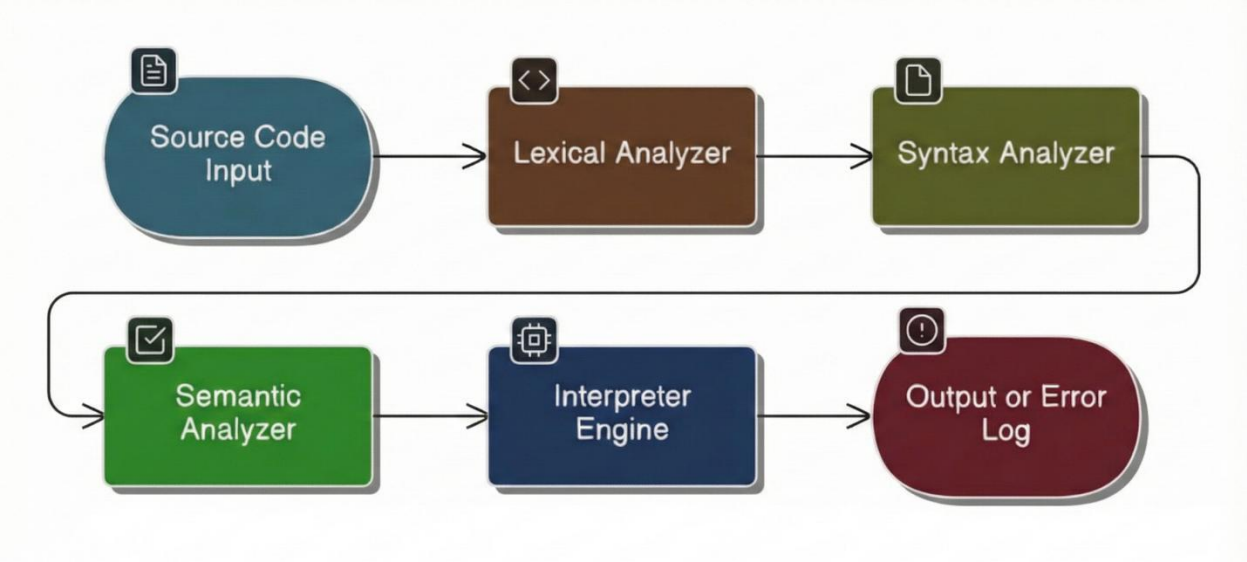


Figure: System Design Diagram

4. Implementation & Testing Result

4.1 Implementation Details

The compiler was implemented using:

- Flex for token generation
- Bison for grammar rules
- C for executing statements and managing variable storage

Each variable is stored in a table containing type, value, and string representation.

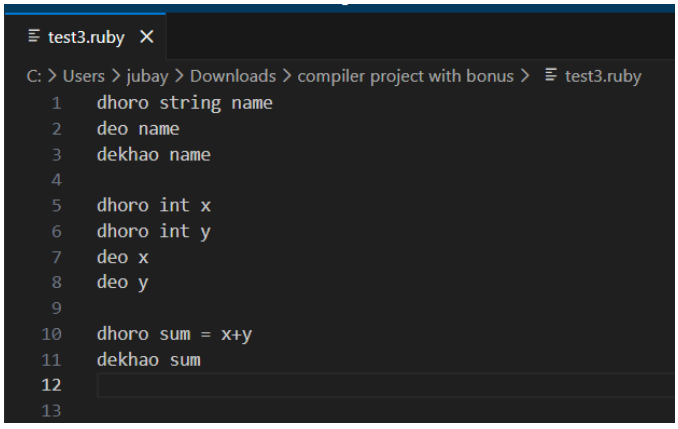
Loops and conditions are implemented using AST traversal and runtime evaluation.

New features implemented:

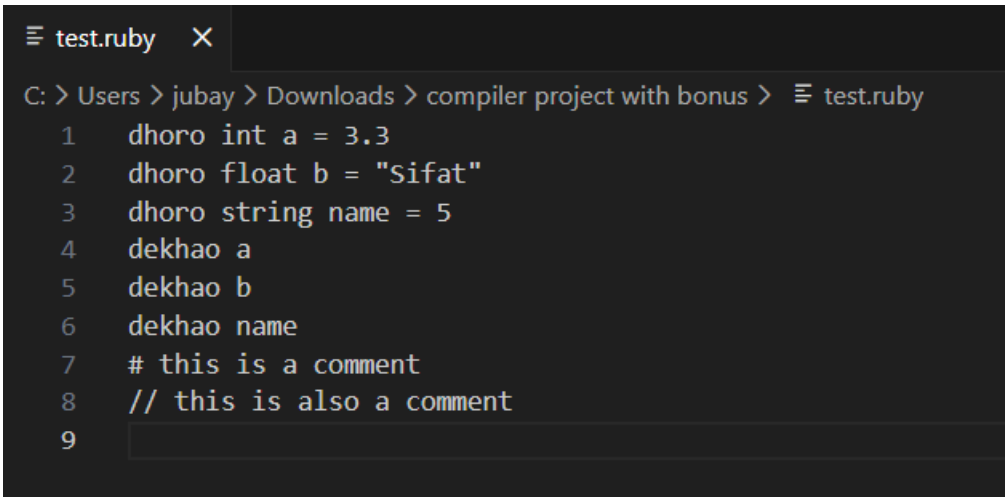
- Type-safe declarations (int, float, string)
- String-number distinction
- FOR loop (cholo ... jonno ... shes)
- Error accumulation
- Comment support

4.2 Test Cases:

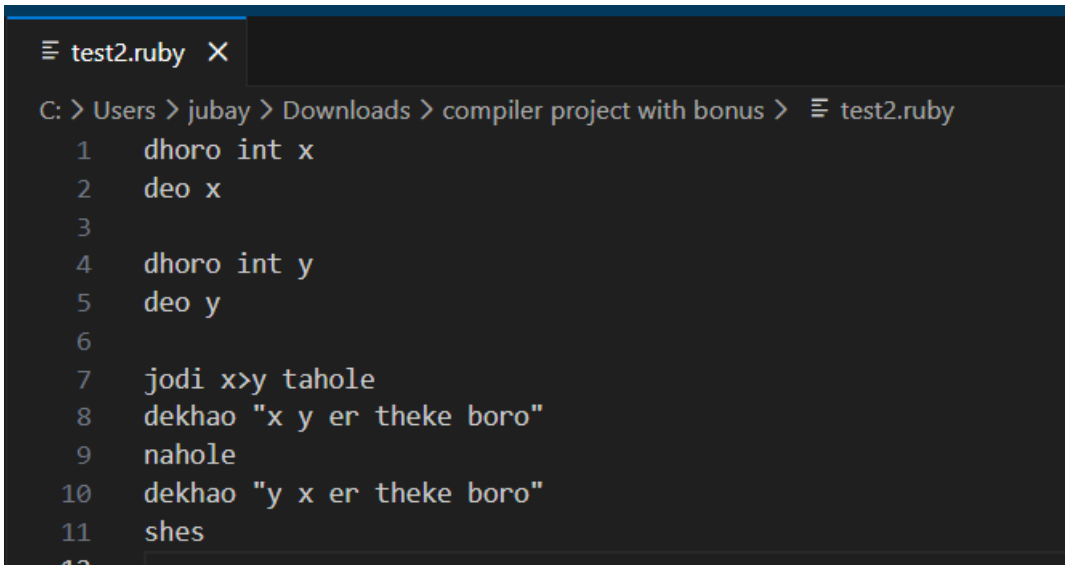
1. **Valid Input:**



2. **Multiple Errors:**



3. **Conditional Statements:**



4. **Loop**

test1.ruby

C: > Users > jubay > Downloads > compiler project with bonus > test1.ruby

```
1   dhoro int i = 0
2
3   cholo i jonno 5
4   dekhao i
5   shes
```

4.3 Output Screenshot:

1. Valid Output:

```
C:\Users\jubay\Downloads\compiler project with bonus>bangla.exe test3.ruby
Enter value for name: Jubayer
Jubayer
Enter value for x: 25
Enter value for y: 25
50
```

2. Multiple Errors Output:

```
C:\Users\jubay\Downloads\compiler project with bonus>bangla.exe test.ruby
Type error (line 9): cannot assign non-integer value 3.300000 to int variable 'a'
Type error (line 9): cannot assign string value to numeric variable 'b'
Type error (line 9): cannot assign numeric value to string variable 'name'
0
0
0

Program finished with 3 type error(s).
```

3. **Conditional Statement Output:**

```
C:\Users\jubay\Downloads\compiler project with bonus>bangla.exe test2.ruby
Enter value for x: 4
Enter value for y: 5
y x er theke boro
```

4. **Loop Output:**

```
C:\Users\jubay\Downloads\compiler project with bonus>bangla.exe test1.ruby
0
1
2
3
4
```

5. Discussion of Outcomes and Limitations

5.1 Achievements

- Successfully implemented a functional Bangla programming language compiler
- Added multitype variable support (int, float, string)
- Developed meaningful semantic error messages (type mismatch, invalid assignment)
- Enabled loop structures, conditionals, and print/input operations
- Provided support for comments to improve readability
- Created a working demonstration of compiler design principles taught in coursework

5.2 Limitations

- The compiler currently works as an interpreter, not a machine-code generator
- Complex data structures (arrays, functions, OOP) are not supported
- Floating-point precision is basic; advanced arithmetic is limited
- Error recovery is minimal; the compiler stops after encountering certain parsing errors
- No GUI interface - the compiler runs from a command line

5.3 Possible Improvements

- Adding function support with parameters and return types
- Introducing arrays, lists, and custom data structures
- Implementing error recovery to continue parsing after detecting errors
- Building a graphical IDE with syntax highlighting
- Extending Bangla keywords to support additional programming constructs
- Allowing compilation to C code or bytecode instead of direct interpretation

6. Conclusion

The development of the Bangla Programming Language Compiler has demonstrated how compiler design principles can be applied to build a customized, domain-specific language. The primary goal of this project was to create a simple compiler that supports Bangla-keyword-based instructions while performing lexical analysis, syntax checking, and semantic error detection. Throughout the implementation, features such as variable declaration, input/output operations, conditional statements, looping constructs, string handling, and meaningful runtime/type error reporting were successfully achieved. The inclusion of robust semantic checks-such as type mismatch detection for integers, floats, and strings-significantly improved the reliability of the language. Additionally, the system's ability to identify and report multiple errors in a single execution showcases a substantial enhancement over basic compiler models. By designing the lexer, parser, and interpreter components carefully, the project demonstrates the core workflow of a functional compiler-from tokenization to parsing, semantic checking, and execution. While the compiler fulfills its intended objectives, certain limitations remain—such as the lack of advanced features like function definitions, arrays, operator precedence handling beyond basics, and optimized intermediate code representation. These present opportunities for future extensions.

Overall, the project successfully illustrates the complete lifecycle of building a small but practical compiler and provides a foundation for future development in language processing and compiler engineering.

7. References

1. Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D.
   Compilers: Principles, Techniques, and Tools (2nd Edition). Pearson, 2006.
   (Also known as the "Dragon Book")
2. Flex & Bison Documentation
   Official GNU Manuals for Lexical Analysis and Parsing
   https://www.gnu.org/software/bison/manual/
   https://westes.github.io/flex/manual/
3. Dennis M. Ritchie & Brian W. Kernighan
   The C Programming Language (2nd Edition). Prentice Hall, 1988.
4. Online Compiler Design Lecture Notes
   Various university courses and tutorials explaining phases of compilation, semantic analysis, and interpreter implementation.
5. GitHub Community Examples
   Open-source projects related to custom language compilers using Flex/Bison.
6. StackOverflow Developer Discussions
   Helpful insights on resolving parsing conflicts, error handling, and semantic checking implementation.