Compiler Design Project Report

Custom Programming Language Implementation Using Flex & Bison

Submitted by:

Name:Sayma Mushsharat

Roll:2007036

Department of Computer Science & Engineering

Khulna University of Engineering & Technology

Supervised by:

Nazia Jahan Khan Chowdhury

- ❑ Assistant Professor
- ❑ Department of Computer Science & Engineering
- ❑ Khulna University of Engineering & Technology

Dipannita Biswas

- ❑ Lecturer
- ❑ Department of Computer Science & Engineering
- ❑ Khulna University of Engineering & Technology

Submission Date:14-01-2025

Table of Contents

1. Introduction

This project implements a custom programming language compiler using Flex (Lexical Analyzer) and Bison (Parser). The language design focuses on readability and includes features for mathematical operations, control structures, and basic programming constructs.

1.1 Project Objectives

- Develop a custom programming language
- Implement lexical analysis using Flex
- Create syntax analysis using Bison
- Support mathematical and logical operations
- Handle variable declarations and operations
- Implement control structures

2. Project Overview

2.1 Tools Used

- Flex (Fast Lexical Analyzer Generator)

- Bison (Parser Generator)
- GCC (GNU Compiler Collection)
- C Programming Language

2.2 System Architecture

```
[Source Code] → [Lexical Analyzer (Flex)] → [Parser (Bison)] → [Output]
```

3. Language Features

3.1 Basic Elements

1. **Comments**
    a. Single line: //comment
    b. Multi-line: /*comment*/
2. **Data Types**
    a. intg: Integer type
    b. float: Floating-point type
    c. charac: Character type
    d. long: Long integer type
3. **Variable Naming**
    a. Format: Var_[name]?
    b. Example: Var_count?

3.2 Operators

1. **Arithmetic**
    a. jog: Addition
    b. biyog: Subtraction
    c. gun: Multiplication
    d. vag: Division
    e. mod: Modulus

f. power: Exponentiation

2. **Comparison**
    a. eq: Equal to
    b. neq: Not equal
    c. Gt: Greater than
    d. Lt: Less than
    e. Geq: Greater than or equal
    f. Leq: Less than or equal
3. **Logical**
    a. &&: AND
    b. ||: OR
    c. !!: NOT

4. Implementation Details

4.1 Lexical Analysis (Flex Implementation)

lex

```
%{
    #include "bison.tab.h"
    #include<stdio.h>
    #include<string.h>
    #include<math.h>
    #include<stdlib.h>
%}

/* Token Definitions */
single_line_comment [/][/].*
multiple_line_comment [/][*][A-Za-z0-9.
\n]*[*][/]
User_Datatype "intg"|"float"|"charac"|"long"
```

```
variable "Var_"[a-zA-Z]([a-zA-Z0-9])*[?]
logical_operator "&&"|"||"|"!!"
digit [0-9]
IDENTIFIER [a-zA-Z]([a-zA-Z0-9])*

%%
/* Token Rules */
{single_line_comment} { /* Handle comments */ }
"intg"    { return INT; }
"float"   { return FLOAT; }
"isit"    { return IF; }
"forl"    { return FOR; }
%%
```

4.2 Syntax Analysis (Bison Implementation)

C:

```
%{
    #include<stdio.h>
    #include<string.h>
    #include <math.h>
    int yyparse();
    int yylex();
    int yyerror();
%}

%union {
    float fvalue;
    int number;
    char string[1009];
```

```
}

/* Token Declarations */
%token <number> NUM
%token <string> VAR
%type <string> statement
%type <number> expression

%%
/* Grammar Rules */
start: '#' import LT HEADER GT codestart;
statement: if_stmt | expression SC | loop_stmt;
expression: NUM | VAR | expression operator expression;
%%
```

5. Sample Programs

5.1 Mathematical Operations

Copy

```
#IMPORT Lt stdio.h
Gt start()
Begin intg Var_x?,
 Var_y?;;
 Var_x? := 10;;
Var_y? := 20;;
show(Var_x? jog Var_y?);;
sin(90);; cos(45);; End
```

Output:

```
add data x 1
add data y 2
value of expression: 30
sin(90) is 1.00000
cos(45) is 0.70739
```

5.2 Control Structures

```
#IMPORT Lt stdio.h Gt
start()
Begin
    intg Var_num?;;
    Var_num? := 5;;
    isit (Var_num? Gt 3)
    Begin
        show(1);;
    End
    or
    Begin
        show(0);;
    End
End
```

6. Compilation and Execution

6.1 Building the Compiler

bash

```
# Generate parser
bison -d app.y
```

```
# Generate lexical analyzer
flex app.l

# Compile
gcc app.tab.c lex.yy.c -o compiler -lm
```

6.2 Running Programs

bash

```
# Interactive mode
./compiler

# File mode
./compiler < input.txt
```

6.3 Error Handling

The compiler implements error handling for:

- Undefined variables
- Type mismatches
- Syntax errors
- Division by zero
- Variable redeclaration

7. Conclusion

7.1 Achievements

- Successfully implemented a custom programming language
- Created a working compiler using Flex and Bison
- Implemented comprehensive mathematical operations
- Developed control structures and variable management

- Added support for various data types and operators

7.2 Future Enhancements

1. Add support for:
    a. Arrays and strings
    b. User-defined functions
    c. More data types
    d. File I/O operations
2. Improve error handling and reporting
3. Implement optimization techniques
4. Add support for modules and libraries

8. References

1. Flex (lexical analyzer generator) documentation
    a. Source: https://github.com/westes/flex
    b. Version used: 2.6.4
2. Bison/Yacc documentation
    a. GNU Bison Manual
    b. Version used: 3.8.2
3. Compiler Construction: Principles and Practice
    a. Author: Kenneth C. Louden
    b. Publisher: Course Technology
    c. Year: 1997
4. LEX & YACC TUTORIAL
    a. Author: Tom Niemann
    b. Source: epaperpress.com/lexandyacc