

Bangladesh Agricultural University

Department of Computer Science and Mathematics

CSM 3222: Compiler Lab

Lab Assignment 4

Parser Implementation using ANTLR4 and Bison

Submitted By:

Name: Ihfaz Hakim Adnan

Student ID:2209014

Level-3, Semester-2

Submitted To:

Md. Saifuddin

Lecturer, Department of Computer Science and Mathematics

Bangladesh Agricultural University

Date of Submission:29.1.26

Contents

1 Task 1: ANTLR4 Parser and Parse Tree Visualization	2
1.1 Objective	2
1.2 Requirements	2
1.3 Installation and Setup	2
1.4 Implementation	2
1.5 Input and Output	3
1.6 Working Principles	4
2 Task 2: Flex + Bison Parser	5
2.1 Objective	5
2.2 Requirements	5
2.3 Installation and Setup	5
2.4 Implementation	5
2.5 Input and Output	8
2.6 Working Principles	9

1 Task 1: ANTLR4 Parser and Parse Tree Visualization

1.1 Objective

To generate a lexer and parser using ANTLR4 for a context-free grammar, validate input strings, and visualize parse trees.

1.2 Requirements

- ANTLR4 (version 4.13.1)
- Java JDK (version 8 or higher)

1.3 Installation and Setup

Prerequisites:

- Verify Java installation: `java --version`

Installing ANTLR4:

1. Download `antlr-4.13.1-complete.jar` from <https://www.antlr.org/>
2. Save it to `C:\ANTLR4`
3. Open Environment Variables and add new System Variable:
 - Name: `ANTLR_JAR`
 - Value: `C:\ANTLR4\antlr-4.13.1-complete.jar`
4. Add `C:\ANTLR4` to Path
5. Create file `C:\ANTLR4\antlr.bat` with content:

```
@echo off
java -jar "%ANTLR_JAR%" %*
```

1.4 Implementation

GitHub Repository: <https://github.com/Ihfaz07/Compiler-lab>
Grammar File (`Expr.g4`):

```
grammar Expr;

// Parser rules
expr:   expr op=('*' | '/') expr      # MulDiv
     |   expr op=('+' | '-') expr      # AddSub
     |   expr '^' expr                # Power
     |   '(' expr ')'
     |   ID
     |   NUM
```

```

;

// Lexer rules
ID : [a-zA-Z]+ ;
NUM : [0-9]+ ('.' [0-9]+)? ;
WS : [ \t\r\n]+ -> skip ;

```

Compilation and Execution:

```

# Generate lexer and parser
java -jar C:\ANTLR4\antlr-4.13.1-complete.jar Expr.g4

# Compile Java files
javac -cp ".;C:\ANTLR4\antlr-4.13.1-complete.jar" *.java

# Run parser with GUI visualization
java -cp ".;C:\ANTLR4\antlr-4.13.1-complete.jar"
    org.antlr.v4.gui.TestRig Expr expr -gui input.txt

# For tree output (without GUI)
java -cp ".;C:\ANTLR4\antlr-4.13.1-complete.jar"
    org.antlr.v4.gui.TestRig Expr expr -tree input.txt

```

1.5 Input and Output

Valid Input (input1.txt):

```
b ^ 5 * 100 + a * ( b * 0.97 - c )
```

Output for Valid Input:

```
(expr (expr (expr b) ^ (expr 5)) * (expr 100)) +
(expr (expr a) * (expr ( (expr (expr (expr b) * (expr 0.97))
- (expr c)) )))
```

Parse Tree Visualization:

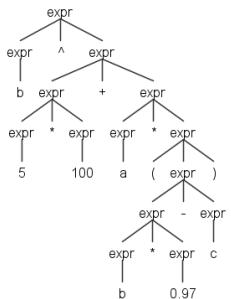


Figure: Parse tree for valid input

Invalid Input (input2.txt):

```
a + * b
```

Output for Invalid Input:

```
line 1:4 mismatched input '*' expecting {'*', '/', '+',  
'-', '^', ID, NUM, '('}
```

1.6 Working Principles

ANTLR4 generates a top-down recursive descent parser from the grammar specification. The lexer tokenizes the input stream, and the parser constructs a parse tree by matching tokens against grammar rules. The parser uses LL(*) parsing with infinite lookahead, enabling it to handle left-recursive grammars automatically. For each rule, ANTLR4 generates visitor and listener interfaces for tree traversal. The `-gui` flag invokes a Java Swing-based visualizer that displays the parse tree structure hierarchically. Valid inputs produce complete parse trees, while invalid inputs trigger syntax error messages indicating the position and nature of the error.

2 Task 2: Flex + Bison Parser

2.1 Objective

To implement a parser using Flex for lexical analysis and Bison for syntax analysis, validating input strings against a context-free grammar.

2.2 Requirements

- Flex (version 2.6 or higher)
- Bison (version 3.x or higher)
- GCC compiler
- Make utility

2.3 Installation and Setup

Installing Flex and Bison on Windows:

1. Download and install MinGW-w64 from <https://www.mingw-w64.org/>
2. Add MinGW bin directory to System PATH (e.g., C:\mingw64\bin)
3. Install Flex and Bison using package manager or download pre-compiled binaries
4. Verify installation:

```
flex --version
bison --version
gcc --version
```

2.4 Implementation

GitHub Repository: <https://github.com/Ihfaz07/Compiler-lab>

Lexer File (lexer.l):

```
%{
#include "parser.tab.h"
%}

%%
"if"        { return IF; }
"else"      { return ELSE; }
[a-zA-Z_][a-zA-Z0-9_]* { return ID; }
[0-9]+       { return NUM; }
"<"         { return LT; }
"=="        { return ASSIGN; }
"+"
```

```
"("      { return LPAREN; }
")"      { return RPAREN; }
"{"      { return LBRACE; }
"}"      { return RBRACE; }
";"      { return SEMI; }
[ \t\n]+  { /* skip whitespace */ }
.        { return yytext[0]; }
%%
int yywrap() { return 1; }
```

Parser File (parser.y):

```
%{  
#include <stdio.h>  
#include <stdlib.h>  
extern int yylex();  
void yyerror(const char *s);  
%}  
  
%token IF ELSE ID NUM LT ASSIGN PLUS MINUS DIV  
%token LPAREN RPAREN LBRACE RBRACE SEMI  
  
%%  
program: statement_list  
        ;  
  
statement_list: statement  
            | statement_list statement  
            ;  
  
statement: assignment SEMI  
        | if_statement  
        ;  
  
assignment: ID ASSIGN expr  
        ;  
  
if_statement: IF LPAREN expr RPAREN LBRACE statement_list RBRACE  
            | IF LPAREN expr RPAREN LBRACE statement_list RBRACE  
            | ELSE LBRACE statement_list RBRACE  
            ;  
  
expr: ID  
    | NUM  
    | expr PLUS expr  
    | expr MINUS expr  
    | expr DIV expr  
    | expr LT expr  
    ;  
  
%%  
  
void yyerror(const char *s) {  
    fprintf(stderr, "Syntax error: %s\n", s);  
}  
  
int main(int argc, char **argv) {  
    if (argc > 1) {  
        FILE *file = fopen(argv[1], "r");  
    }  
}
```

```

        if (file) {
            yyin = file;
        }
    }
    if (yparse() == 0) {
        printf("Parsing successful!\n");
    }
    return 0;
}

```

Compilation and Execution:

```

# Generate parser and lexer
bison -d parser.y
flex lexer.l

# Compile
gcc parser.tab.c lex.yy.c -o parser

# Run with input
./parser input.txt

```

2.5 Input and Output

Valid Input (input1.txt):

```

x = y / 5;
if ( a ) {
    if( a < b ) {
        m = 5;
    } else {
        m = 10;
    }
} else {
    x = y + z;
}

```

Output for Valid Input:

Parsing successful!

Invalid Input (input2.txt):

```

x = y / ;
if ( a ) {
    m = 5
}

```

Output for Invalid Input:

Syntax error: syntax error

2.6 Working Principles

Flex and Bison implement a two-stage parsing process. Flex performs lexical analysis by reading the input character stream and generating tokens based on regular expression patterns. These tokens are passed to the Bison-generated parser, which performs syntax analysis using an LALR(1) bottom-up parsing algorithm. Bison builds a parse table from the grammar rules and uses a shift-reduce mechanism to validate input against the grammar. When all input is successfully reduced to the start symbol, parsing succeeds. Syntax errors occur when the parser encounters unexpected tokens that don't match any production rule. The `yyerror` function handles error reporting, while the `main` function orchestrates the parsing process.