Khulna University of Engineering & Technology

Department of Computer Science & Engineering

Course No : CSE 3112

Course Title : Compiler Design Laboratory

Project Name : Simple Compiler using Bison and Flex

Submission By

Ahsan Habib

Roll : 1807064

Objective:

After doing this project, we will be able to know

- About Flex and Bison.
- About token and how to declare rules against token.
- How to declare CFG (context free grammar) for different grammar like if else pattern, loop and so on.
- About different patterns and how they work.
- How to create different and new semantic and synthetic rules for the compiler.
- About shift and reduce policy of a compiler.
- About top down and bottom up parser and how they work.

Introduction:

A compiler is a special program that processes statements written in a particular programming language and turns them into machine language or "code" that a computer's processor uses. Typically, a programmer writes language statements in a language such as Pascal or C one line at a time using an editor. The file that is created contains what are called the source statements. The programmer then runs the appropriate language compiler, specifying the name of the file that contains the source statements.

Flex:

Flex (fast lexical analyzer generator) is a free and open-source software alternative to lex. It is a computer program that generates lexical analyzers (also known as "scanners" or "lexers"). An input file describing the lexical analyzer to be generated named lex.l is written in lex language. The lex compiler transforms lex.l to C program, in a file that is always named lex.yy.c. The C compiler compiles the lex.yy.c file into an executable file called a.out. The output file a.out takes a stream of input characters and produces a stream of tokens.

```
/* definitions */
....
%%
/* rules */
....
```

%%

/* auxiliary routines */

….

Bison:

GNU Bison, commonly known as Bison, is a parser generator that is part of the GNU Project. Bison reads a specification of a context-free language, warns about any parsing ambiguities, and generates a parser (either in C, C++, or Java) that reads sequences of tokens and decides whether the sequence conforms to the syntax specified by the grammar. Bison command is a replacement for the yacc. It is a parser generator similar to yacc. Input files should follow the yacc convention of ending in .y format.

/* definitions */

....

%%

/* rules */

....

%%

/* auxiliary routines */

….

Run The program in terminal

1. bison -d 1807064.y
2. flex 1807064.l
3. gcc lex.yy.c 1807064.tab.c -o x
4. x

Procedure:

1.The code is divided into two part flex file (.l) and bison file (.y) .

2.Input expression check the lex (.y) file and if the expression satisfies the rule then it check the CFG into the bison file .

3.it's a bottom up parser and the parser construct the parse tree .firstly ,matches the leaves node with the rules and if the CFG matches then it gradually goes to the root .

Token

A **token** is the smallest element(character) of a computer language program that is meaningful to the **compiler**. The parser has to recognize these as **tokens**: identifiers, keywords, literals, operators, punctuators, and other separators.

My compiler tokens

NUM, VAR ,IF, ELSE, ARRAY ,MAIN ,INT, FLOAT ,CHAR ,START, END ,FOR, WHILE, PRINTFUNCTION, SIN, COS ,TAN, LOG ,CASE, DEFAULT, SWITCH

CFG

Context-free grammars (CFGs) are used to describe context-free languages. A context-free grammar is a set of recursive rules used to generate patterns of strings. A context-free grammar can describe all regular languages and more, but they cannot describe *all* possible languages.

My compiler CFGs

```
program: MAIN ':' START line END

line:       /* NULL */
            | line statement
            ;

statement: ';'
            | declaration ';'
            | expression ';'
            |VAR '=' expression ';'
            | WHILE '(' NUM '<' NUM ')' START statement END
            | IF '(' expression ')' START expression ';' END %prec IFX
            | IF '(' expression ')' START expression ';' END ELSE START expression ';' END
            | PRINTFUNCTION '(' expression ')' ';'
            | ARRAY TYPE VAR '(' NUM ')' ';'
            | SWITCH '(' NUM ')' START SWITCHCASE END
            | FOR '(' NUM ',' NUM ',' NUM ')' START statement END


declaration : TYPE ID1

TYPE : INT
             | FLOAT
             | CHAR
ID1 : ID1 ',' VAR
            |VAR
SWITCHCASE: casegrammer
             |casegrammer defaultgrammer
 casegrammer: /*empty*/
             | casegrammer casenumber
 casenumber: CASE NUM ':' expression ';'
 defaultgrammer: DEFAULT ':' expression ';'



expression: NUM
            | VAR
```

| expression '+' expression

| expression '-' expression

| expression '*' expression

| expression '/' expression

| expression '%' expression

| expression '^' expression

| expression '<' expression

| expression '>' expression

| '(' expression ')'

| SIN expression

| COS expression

| TAN expression

| LOG expression

Features of this compiler

1.header file

2. Main function

3.Comments

4.Variable declaration

5. IF ELSE Block

6.Variable assignment

7. Array Declaration

8. For loop

9. While loop

10. Print function

11. Switch Case

12.Mathematical Expression

Addition, Subtraction, Multiplication, Division, Power, Log () Operation, Sin () operation, Tan () operation, Cos () operation.

Discussion:

The input code is parsed using a bottom-up parser in this compiler. Because it is only built with flex and bison, this compiler is unable to provide original functionality such as if-else, loop, and switch case features. However, when creating code in this compiler-specific style, header declaration is not required but if we need we can use header file. The float variable always returns a value in the double data type, which is a compiler requirement. Any variable's string value is not stored by this compiler. With certain modifications, the code format supported by this compiler is similar to that of the C language. This compiler is error-free while working with the stated CFG format.

Conclusion:

Every programming language has required the use of a compiler. Designing a new language without a solid understanding of how a compiler works may be a challenging endeavor. Several issues were encountered during the design phase of this compiler, such as loop, if-else, switch case functions not working as they should owing to bison limitations, character and string variable values not being stored properly, and so on. In the end, some of these issues were resolved, and given the constraints, this compiler performs admirably.

References:

- https://whatis.techtarget.com/definition/compiler
- Principles of Compiler Design By Alfred V.Aho & J.D Ullman