Lab Report on Project

Course No: CSE 3212

Course Name: Compiler Design Laboratory

Submitted To:

Dola Das Lecturer Department of Computer Science and Engineering Khulna University of Engineering and Technology, Khulna	Md. Ahsan Habib Nayan Lecturer Department of Computer Science and Engineering Khulna University of Engineering and Technology, Khulna

Submitted By:

Md. Mahfuzul Haque Gazi
Roll: 1707003

Year: 3rd 2nd Term

Department of Computer Science and Engineering
Khulna University of Engineering & Technology, Khulna

Submission Date: June 15, 2021

Flex and Bison

Flex: Flex is a fast lexical analyzer generator. Lexical analysis is the process where input string is divided into meaningful units such as variables, constants, keywords, operators, punctuation etc. These units are also called as tokens. To identify each token, there must be grammar that defines itself. Flex takes a program written in a combination of Flex and C, and it writes out a file – 'lex.yy.c' that holds a definition of function yylex(), with the following prototype.

int yylex(void)

Each time yylex is called, it returns the next token reading a '.l' file. A flex program has the following structure:

{ definitions }

%%

{ rules }

%%

{ user subroutines }

Bison: Bison, also known as GNU Bison, is a general purpose parser generator that converts a grammar description for an LALR contest-free grammar (CFG) into a C program to parse that generator. It is compatible with yacc (yet another compiler compiler). All properly-written Yacc grammars ought to work with Bison with no change. A bison file is written in the following format:

%{
C declarations (types, variables, functions, preprocessor commands)

%}

Bison declarations (grammar symbols, operator precedence decl., attribute data type)

%%

grammar rules

%%

additional C code goes here

Flex and Bison are tools for building programs that handle structured input. They were originally tools for building compilers, but they have proven to be useful in many other areas. Lex and yacc are tools used to generate lexical analyzers and parsers. In this project, bison takes the tokens generated from flex and grammar rules written in the '.y' file and generates a syntax tree. In the '.y' file, the grammar rules are written in BNF form. YACC(Yet Another Compiler Compiler) is used to generate a '.h' file and '.c' file from the '.y' file.

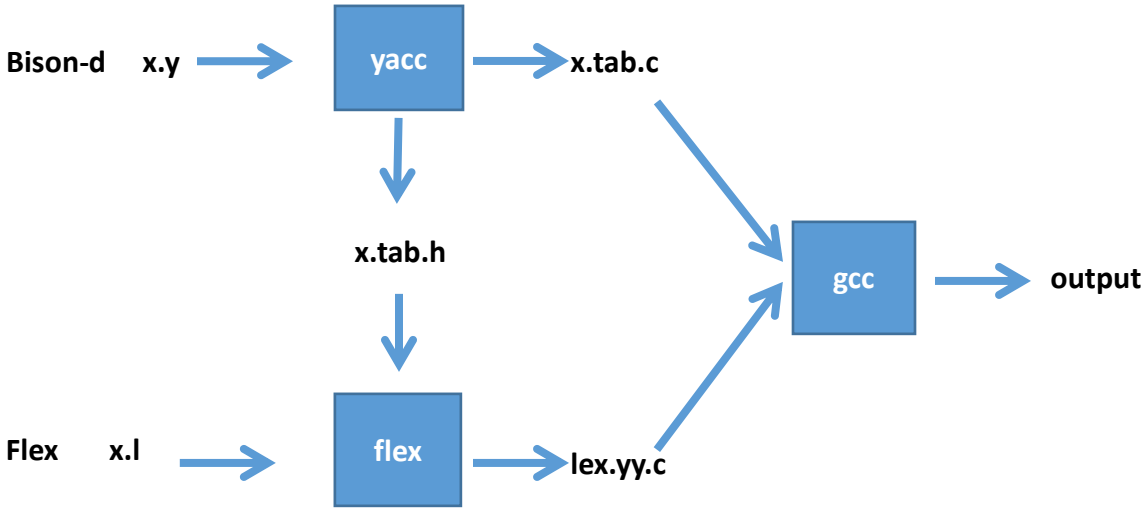

Figure 1: Flex and Bison workflow

Run the program in terminal:

- bison -d project.y
- flex project.l
- gcc project.tab.c lex.yy.c –o app
- app

Tokens:

Serial No	Token	Input String	Description
1.	MAIN	main	Defines main function

2.	begin	begin	Defines the starting of a function
3.	end	end	Defines the end of a function
4.	INT	inumber	Specifies the variable type as integer number Example: inumber a;
5.	IF	if	Similar to if() in C. Example: if(a<b)
6.	FI	fi	Specify end of if statement. Example: if(a<b) a = b; fi
7.	ELSE	else	Similar as else statement in C
8.	OUT	display()	Prints the value of expression given between parenthesis. Example: display(a+b) ;
9.	SWITCH	switch	Similar to switch() in C
10.	CASE	case	Similar to case in C
11.	DEFAULT	default	Similar to default in C
12.	LOOP	loop	Defines a loop statement. Example: loop(a:b) statement; done [where loop continues from a to b, here a<b]
13.	POOL	pool	Defines a different loop statement. Example: pool(b:a) statement; done [where loop continues from b to a, here b>a]
14.	DONE	done	Defines end of a loop statement
15.	POW	**	Functions as pow() in c. For example pow(a, b) is written as: a ** b;
16.	EQ	==	Similar to == operator in C
17.	GE	>=	Similar to >= operator in C
18.	LE	<=	Similar to <= operator in C
19.	NE	!=	Similar to != operator in C
20.	varname	[a-zA-Z]+	Denotes variable name like C

21.	num	[-]?[0-9]+	Represents any integer number, both positive and negative.

CFG used in this compiler:

program:

MAIN begin cstatement end

;

cstatement: /*null*/

| cstatement statement

;

statement:';'

| expression ';'

| declaration ';'

| varname '=' expression ';'

| IF '(' expression ')' expression ';' FI

| IF '(' expression ')' expression ';' FI ELSE expression ';' FI

| IF '(' expression ')' expression ';' FI ELSE IF '(' expression ')' expression ';' FI ELSE

expression ';' FI

;

expression ';' FI

| OUT '(' expression ')' ';'

| SWITCH '(' varname ')' begin B end

```
        |LOOP '(' varname ':' varname ')' statement DONE
        |LOOP '(' varname ':' varname ':' expression')' statement DONE
        |POOL '(' varname ':' varname ')' statement DONE
        |POOL '(' varname ':' varname ':' expression')' statement DONE
        ;

B   : C
        | C D
        ;
C   : C '+' C
        | CASE num ':' expression ';'
        ;
D   : DEFAULT ':' expression ';'
declaration: TYPE ID1
        ;
TYPE: INT
        ;
ID1:    ID1 ',' varname
        | varname
        | ID1 ',' varname '=' expression
        |varname '=' expression
        ;
expression:
```

num

| varname

| expression '+' expression

| expression '-' expression

| expression '*' expression

| expression '/' expression

| '(' expression ')

| expression '<' expression

| expression '>' expression

| expression POW expression

| expression '%' expression

| expression EQ expression

| expression LE expression

| expression GE expression

| expression NE expression

;

Precedence:

- left '<' '>' EQ LE GE NE
- left '+' '-'
- left '/' '*'
- left POW '%'

Features in this compiler:

1. Header file
2. Main function
3. Single line comment
4. Multiple line comment
5. Single/Multiple Character Variable declaration
6. IF ELSE Conditions
7. Variable assignment
8. Loops
9. Print(Display) function
10. Switch Case
11. Mathematical Expression :
    a. Addition
    b. Subtraction
    c. Multiplication
    d. Division
    e. Power
    f. Modulus

Conclusion:

In this project, a simple compiler is implemented. It can perform most of the simple arithmetic and logical operations as performed in C. The input program is taken from a text file and outputs are simply displayed in terminal. However, this compiler can not handle switch case, instead it performs all statements in cases and default. Only integer values can be assigned in a variable, but double values can be stored after division operation.

References:

- LEX & YACC TUTORIAL by Tom Niemann
- https://www.oreilly.com/library/view/flex-bison/9780596805418/ch01.html

- https://en.wikipedia.org/wiki/GNU_Bison
- https://tomassetti.me/why-you-should-not-use-flex-yacc-and-bison