KHULNA UNIVERSITY OF ENGINEERING & TECHNOLOGY

Department of Computer Science and Engineering

Report on CSE3212 PROJECT

Course Title: Compiler Design Laboratory

Topic: Simple Compiler using Flex and Bison

Date of Submission: June 15, 2021

Submitted by:

Rakib Hasan

Roll: **1707014**

3rd Year 2nd Semester

Department of Computer Science and Engineering

Khulna University of Engineering & Technology (KUET),
Khulna-9203

TABLE OF CONTENTS

Topics: **Page no:**

Introduction:

In computing, a compiler is a computer program that translates computer code written in one programming language into another language. The name “Compiler” is used for programs that translate source code from a high level programming language to a lower level language to create executable program. A compiler is likely to perform many or all of the following operations: preprocessing, lexical analysis, parsing, semantic analysis, intermediate code generation, code optimization, code generation. Compilers implement these operations in phases that promote efficient design and correct transformation of source code to target code.

Flex:

For dividing the input into meaningful units such as variable, keywords, constants, operators, punctuations, etc. we need a tool called “Flex”. This tokenization step is called “Lexical Analysis”. Flex is a tool for generating scanners. Flex source is a table of regular expressions and corresponding program fragments. It generates lex.yy.c file which defines a routine yylex(). The required file for Flex is saved with “.l” extension. Structure of a file that is compatible with Flex is as follows:

```
{Definitions}
%%
{Rules}
%%
{User subroutine}
```

Bison:

Parsing is an essential stage for designing a compiler. In this stage, Context Free Grammars are used to parse input commands of a program using a parse tree. Bison is a powerful and free version of “yacc” which has the full form “yet another compiler compiler”, a UNIX parser. It is a general purpose parser generator that converts a grammar description for an LALR(1) context free grammar into a C program to parse the grammar. Bison is compatible with yacc. All properly written yacc grammars are ought to work with bison with no change. It interfaces with scanners generated by Flex. Bison input file is saved with a “.y” extension. Bison generates two files compiling a input file those are: inputfile_name.tab.h and inputfile_name.tab.c. File with “.h” extension is used in Flex input file to connect both file and the later one is used as a parser scanner. Input file structure of a bison file is as follows:

%}

C declarations

%}

Bison Declarations

%%

Grammar Rules Declarations in BNF form(CFG)

%%

Additional C codes

Procedure:

1. The code is divided into a flex file (.l) and a bison file (.y).
2. Using the bison file code is drawn from a text file (.txt).
3. After that input expressions are matched with the rules of flex file (.l) and upon satisfaction of the matching step, the CFG of the bison file (.y) is checked for a match.
4. It is a bottom up parser and the parser construct the parse tree. At first, matches the leaves node with the rules and if the CFG matches then it gradually goes to the root.

Features in this compiler:

- Header declaration
- Body declaration

- Variable declaration (integer, character, float)
- Variable value assignment
- Arithmetic operations (+ , - , * , /)
- If condition
- If-else condition
- For loop
- Switch-case
- Print Function

Tokens used in this Compiler:

The following table represents the tokens and meaning of these tokens used in this compiler design process.

Serial No	Token	Input string	Definition of Token
1	VoidMain	MAIN	Defines the start of the function similar as main() in C
2	Int	int	Specify the variable of integer type
3	Float	float	Specify the variable of float type
4	Char	char	Specify the variable of char type
5	Num	[0-9]	Any number
6	Vr	[a-z]	Denotes variable
7	Sf	Sf	Denotes start of parentheses
8	Ef	Ef	Denotes end of parentheses
9	Ss	Ss	Denotes starting curly braces
10	Es	Es	Denotes ending curly braces
11	Com	COMA	Similar to ‘,’ (comma) in C

12	Semi	SEMI	Similar to ‘;’ (semicolon) in C
13	Jug	(+)	Denotes the plus operation
14	Biyog	(-)	Denotes the minus operation
15	Gun	(*)	Denotes the multiplication operation
16	Vaag	(/)	Denotes the division operation
17	Vgses	(%)	Denotes the modulus operation
18	Soman	(=)	Denotes the equal operation
19	Bigger	BORO	Represent greater than logical operation
20	Smaller	CHOTO	Represent smaller than logical operation
21	If	JODI	Similar to if() in C
22	Else	NAHOLE	Similar to else in C
23	LOOP	LOOP	Similar to for loop in C
24	PF	PF	Prints the variable value
25	SWITCH	SWITCH	Similar to switch() in C
26	CASE	CASE	Similar to case in C
27	DEFAULT	DEFAULT	Similar to default in C
28	BREAK	BREAK	Similar to break in C
29	Colon	(:)	Similar to ‘:’ (colon) in C

Table 1.1: Table of used tokens and meanings of tokens.

Context Free Grammar(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. The production rule of a CFG is of the form,
A → b where A is a non-terminal and b is a terminal.

CFGs used in this compiler:

```
program: VoidMain Sf Ef   Ss  ccode  Es


ccode:
            | ccode statement
            | ccode cdecl
            ;
cdecl : Type ID1 Semi


Type: Int
            | Float
            | Char
            ;
ID1: ID1 Com Vr
```

```
statement: Semi
           | exp Semi
           | Vr Soman exp Semi
           | If_code
           | If_else_code
           | LOOP_code
           | PF
           |Switch_code


Switch_code:
  SWITCH Sf Vr Ef Ss B Es


B   :   C
           | C   D
C : C jug C
           | CASE Num Colon exp Semi BREAK Semi
D : DEFAULT Colon exp Semi BREAK Semi



exp   : Num
           | Vr
           | exp Jug exp
           | exp Biyog exp
           | exp Gun exp
           | exp Vaag exp
           | exp  Vgses exp
           | exp Smaller exp
           | exp Bigger exp
```

Terminal commands to run the program:

1. bison –d cp.y
2. flex cp.l
3. gcc lex.yy.c cp.tab.c –o ex
4. ex

Discussion:

This compiler uses a bottom up parser to parse the input code. This compiler is unable to provide original functionality of if else, loop, switch case features as it is only build using flex and bison. However , header declaration is not mandatory while writing a code in this compiler specific format. This compiler doesn’t allow to initialize multiple character variable.This only support lowercase English alphabet to declare variable. Also , this compiler doesn’t support min max function like C . This compiler doesn’t store string value of any variable. The code format that is supported by this compiler is close to that of C language with some modification. This compiler works perfectly with the declared CFG format without any error.

Conclusion:

Compiler has been an essential part of every programming language. Without a sound knowledge about how a compiler works, designing a new language can be very difficult task. Several difficulties are faced during design period of this compiler such as loop, if else, switch case functions not working as it should be due to limitation of bison, character and string variable value isn't storing properly, etc. At the end, some of this problems have been fixed and considering the limitations this compiler works just fine.

References:

1. blueshark-14/Compiler-Project-Beginner-level-1707014 (github.com)