KHULNA UNIVERSITY OF ENGINEENING AND TECHNOLOGY

CSE 3212

Compiler Design Laboratory

Submitted To

DOLA DAS
Lecturer
Department of Computer Science and Engineering (CSE)
Khulna University of Engineering and Technology (KUET)

Md. AHSAN HABIB NAYAN
Lecturer
Department of Computer Science and Engineering (CSE)
Khulna University of Engineering and Technology (KUET)

Submitted By

SHAMEEM AHAMMED

Roll: 1707117

Year: 3rd Semester: 2nd

Department of Computer Science and Engineering (CSE)

Khulna University of Engineering and Technology (KUET)

Date of Submission: 15 June,2021

Objective:

The main purposes of this lab are given below:

- To gather knowledge about compiler
- To learn bison and flex
- To create token from a language
- To create a new language of our own

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. We normally write our programming statements in a particular language to an editor then the source code is run to appropriate language compiler. Compiler makes the source code file to an executable file.

Flex and Bison:

Flex is a tool that generates token. It also does the job of pattern matching. .It reads the input stream and produces the source code as output through implementing the lexical analyzer in the C program. The extension of flex file is '.l'. It creates a C file named lex.yy.c after execution. A Flex file contains yylex() function which starts scanning and called yywrap(). When the scanning is oven then yywrap() returns and the scanning is finished.

Bison is a parser generator. The flex file provides token or scanners to a bison file and the bison file generates a syntax tree. The file extension for a bison file is '.y' (yacc). YACC- Yet Another Compiler Compiler is used to generate a '.h' and '.c' file. The '.h' file contains the definitions of the tokens the '.l' file

returns. The function “yyparse()” is called by yacc which calls the yylex() function to get the tokens. Bison stores the parsed tokens in two stacks: parse stack, and value stack. In value stack the values of the parsed tokens are assigned automatically starting from ASCII value 258. The parsed tree generated by Bison is calculated from bottom to top.

The following diagram illustrates the workflow of flex and bison:

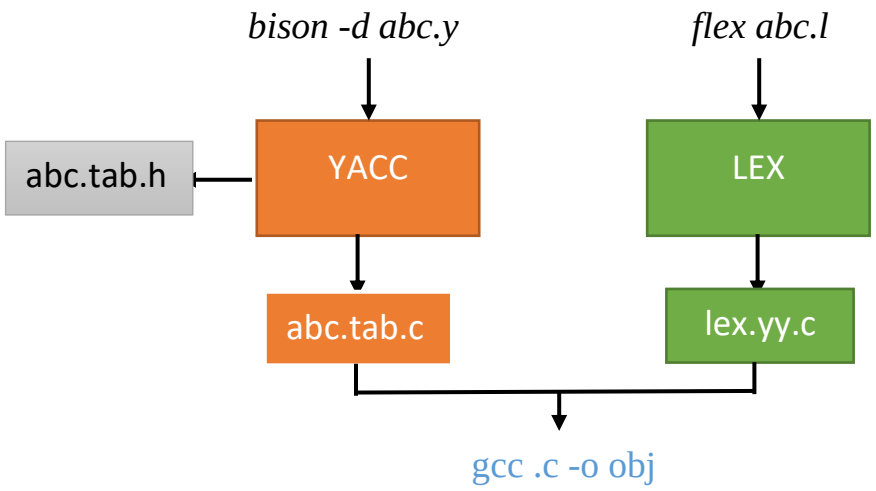

Fig 1: Workflow of Flex and Bison.

Running Bison and Flex Program:

The command prompt script for running bison and flex program is given step by step:

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

PROJECT

In this lab, we were assigned to design our own unique programming language using flex and bison. To complete my task, I have designed a source code which is my own language, a lexical analyzer and a syntax analyzer.

A source code with my own keywords has been written. It is passed to the lexical analyzer. The lexical analyzer program matches the syntax and generates token. Then in the syntax analyzer, there are grammar rules, data structure and other functionalities are defined. Yacc reads the grammar and generate C code for a syntax analyzer or parser. The syntax analyzer uses grammar rules that allow it to analyze tokens from the lexical analyzer and create a syntax tree.

My language has two parts. They are:

- Header Section
- Main Function

Header Section includes all header files. The syntax for including header file is:

!include header_file_name. Example: !include string, !include math.h

Before starting to our main function let’s introduce some basic signs with their tokens and examples:

Sign	Token	Syntax	Example
+ (add)	ADD	number or variable + number or variable;	4+5; a+3; a+b;
- (sub)	SUB	number or variable – number or variable;	7-2; a-b; 8-a;
* (mul)	MUL	number or variable * number or variable;	7*9; a*b; a*5;
/ (div)	DIV	number or variable / number or variable;	6/3; a/b; c/8;
% (mod)	MOD	number or variable % number or variable;	7%2; a%b; 34%a;
> (greater than)	>	number or variable > number or variable;	a>b; 4>2; 6>c;
< (less than)	<	number or variable < number or variable;	a<b; 89<90; c<92;
= (assign)	=	variable = expression	a=8+9; b=3; c=45%3;
!/	COMMENT	single line comment	!/this is comment
!!/	MC	multiple line comment	!!/ multiple line comment
br (break)	BREAK	break statement	break;

Main Function:

Normally a program starts with main function. In my program, it will also start from the main function. The syntax and token are given below with example:

Function Name	Token	Syntax	Example

suru	MAIN	suru:
		{
		statements;
		}
			suru:
			{
			int a,b;
			a=6;
			}

Inside the main function the following features are added:

- Data type
- Variable declaration
- Assign values to variables
- Input
- Output function (printing values)
- Single line comment
- Multiple line comment
- If
- Else
- Forward for loop
- Backward loop
- While loop
- Arithmetic operation
- Switch
- Increment
- Decrement
- Square Root function
- Prime number function
- GCD function

- LCM function
- Fibonacci series function

Data Types & Variable Declaration:

The data type of my program is given below:

Data Type	Token	Syntax	Example (Variable Declaration)
int (for integer)	INT	int variable, variable name [a-z] and [A-Z]	int a,b,c;
float (for float)	FLOAT	float variable, variable name [a-z] and [A-Z]	float a,B;
char (for character)	CHAR	char variable, variable name [a-z] and [A-Z]	char a,B,c;

Assign Values to the Variables:

Syntax: int a=7; a=8;

Input Value for Variables:

Name of function	Token	Syntax	Example
cin (for inputting data)	CIN	cin(variable,value)	cin(a,67); cin(c,-78);

Printing Variables/Values:

Name of function	Token	Syntax	Example

display (for inputting data)	SHOW	display(varibales);
		display(values);
		display(expression);
			display(a);
			display(89);
			display(8+9);

Comments:

There are single line and multiple line comment for my program.

!/ single line comment

!!/ multiple line comment

Arithmetic Operations:

Syntax: a+b; a-b; 7+b; 4/6; 78%3;

If-Else:

Name of function	Token	Syntax	Example

condition (work as if)	IF	if(variable or number or expression > or < variable or number or expression)
		{
		statements;
		}
			if(a>b)
			if(6>8)
			if(a<9)
			{
			a=4+b;
			}

or (work as else)	ELSE	or
		{
		statements;
		}
			or
			{
			a+b;
			a=9+3;
			}

Forward For Loop:

Name of function	Token	Syntax	Example

loop	LOOP	loop(variable or number < or > variable or number)
		{
		statements;
		}
			loop(a>b)
			loop(1<10)
			{
			a=9+7;
			}

Backward For Loop:

Name of function	Token	Syntax	Example

rloop	RLOOP	rloop(variable or number < or > variable or number)
		{
		statements;
		}
			rloop(a>b)
			rloop(1<10)
			{
			a=9+7;
			}

While Loop:

Name of function	Token	Syntax	Example

jotokkhon	WHILE	jotokkhon(variable or number < or > variable or number)
		{
		statements;
		}
			jotokkhon(a>b)
			jotokkhon(1<10)
			{
			a=9+7;
			}

Switch-Case:

Name of function	Token	Syntax	Example

switch	SWITCH	switch(variable or number)
		{
		cases;
		}
			switch(a)
			switch(6)
			{
			}

cs	CASE	cs case_number: statements; break;	cs 1: a+b; br;(break)
dflt	DEFAULT	dflt: statements; break;	dflt: a+7; br;

Increment-Decrement:

Name of function	Token	Syntax	Example
inc (for increment)	INC	inc(variable, unit)	inc(a,9); inc(b,-9);
dec (for decrement)	DEC	dec(variable, unit)	dec(b,-9); dec(a,7);

Some Functions:

Name of function	Token	Syntax	Example

root (for square root)	ROOT	root(variable);
		root(expression)
		root(number)
			root(a);
			root(9);
			root(16+9);

prime (for finding the number is prime or not)	PRIME	prime(variable);
		prime(expression)
		prime(number)
			prime(a);
			prime(7);
			prime(a+8);

lcm (least common multiple)	LCM	lcm(variable or number or expression, variable or number or expression);	lcm(a,b);
			lcm(7,9);
			lcm(3+3,9);

gcd (greatest common divisor)	GCD	gcd(variable or number or expression, variable or number or expression);	gcd(a,b);
			gcd(1,5);
			gcd(34,54);

fib (Fibonacci series)	FIB	fib(variable or number or expression)	fib(a);
			fib(4);
			fib(2+5);

A Simple Input and Output to My program:

Input	Output

!include math
!include string.h
 	
suru:
{
  int a,b,c,d;
  a=8;
  b=10;
  c=11;
  prime(a);
  gcd(b,c);
  lcm(a,c);
  cin(d , 12);
  display(d);
  root(4);
  fib(6);
 	
  !/if without else
 	
  condition(c>b)
  {
  c-b;
  }
  !/if with else
 	
  condition(b>a)
  {
  b-a;
  }
  or
  {
  a-b;
  }
  !/display function
!/loop
	Header file include
	Header file include
	
	This is a Declaration
	Value of the variable= 8
	Value of the variable= 10
	Value of the variable= 11
	8 is not prime
	Gcd of 10 and 11 = 1
	Lcm of 8 and 11 = 88
	input done. Value =12
	Printing Value 12
	Value of root=2.000000
	The fibonacci series is= 0 1 1 2 3 5
	//This is comment
	
	value of expression in condition: 1
	//This is comment
	
	value of expression in condition: 2
	//This is comment
	
	//This is comment

```
  loop (a<c)
  {
    7-8;
  }

  rloop (b>a)
  {
     100/5;
        }

  switch ( a )
        {
        cs 1 :
                a + b; br;
        +
        cs 2 :
                b + c; br;
        +
        cs 3 :
                a + c; br;
        dflt :
                c + 4; br;
        }

        jotokkhon (6>3)
        {
          c=c+1;
         }

        inc (d,3);

        display(d);
        dec (d,2);
        display(d);
        display(4*9);
}
```

```
The value of expression= -1
value of the loop: 8 expression value: -1
value of the loop: 9 expression value: -1
value of the loop: 10 expression value: -1
value of the loop: 11 expression value: -1
The value of expression= 20
value of the loop: 10 expression value: 20
value of the loop: 9 expression value: 20
value of the loop: 8 expression value: 20

Case 1 executed and the expression value is=18
Case 2 executed and the expression value is=21
Case 3 executed and the expression value is=19
Default Case executed and the expression
value=15

Value of the variable= 12
value of the while loop: 6
value of the while loop: 5
value of the while loop: 4
value of the while loop: 3

Printing Value 15
Printing Value 13
Printing Value 36
```

Discussion and Conclusion:

This lab was very interesting to all of us as we are now able to create our own language and compiler. So far, we code in normally C/C++/java/python but now we are able to run our own program. A project which we performed, make our basic clear. We make our program in educational purposes. If we want to make a language for public use, we need to put more effort on it. A vast area of script is needed. Our designed program may not contain all the features of a language. Though there were some difficulties for all in this pandemic situation to attain lab, limitation of internet and so on, hope our knowledge on compiler is now very sharp by performing this laboratory.

References:

1. Flex, version 2.5 by Vern Paxson

2. https://whatis.techtarget.com/definition.