REPORT

CSE 3212

Course Name: Compiler Design and Laboratory.

Submitted to:

Dola Das

Lecturer,

Department of Computer Science and Engineering,

Khulna University of Engineering and Technology.

Ahsan Habib Nayan

Lecturer,

Department of Computer Science and Engineering,

Khulna University of Engineering and Technology.

Submitted by:

Sajidul Islam

Roll: 1707010

Department of Computer Science and Engineering,

Khulna University of Engineering and Technology.

Submission Date: *June 15 , 2021*

Introduction:

FLEX

Flex is A fast Lexical Analyzer Generator. It Dividing the input into meaningful units. For a C program the units are variables, constants, keywords, operators, punctuation etc. These units also called as tokens. Flex takes a program written in a combination of Flex and C, and it writes out a file (called lex.yy.c) that holds a definition of function yylex().

BISON

Bison is a general-purpose parser generator that converts a grammar description for an LALR(1) context-free grammar into a C program to parse that grammar. Bison is upward compatible with Yacc: all properly-written Yacc grammars ought to work with Bison with no change.Interfaces with scanner generated by Flex.Scanner called as a subroutine when parser needs the next token.

.

COMPILER OVERVIEW

This manual compiler consists of two major components. Which are Tokens and Context free grammar (CFG). Their description is given below:

Tokens:

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 separator.

My Custom Tokens are:

- *NUM* : This token is returned to parser when it matches any numbers.
- *VAR* : This token is returned to parser when it matches any characters.
- *IF :* This token is returned to parser when it matches “IF”.
- *ELSE :* This token is returned to parser when it matches “FI”.
- *MAIN :* This token is returned to parser when it matches “body”.
- *INT :* This token is returned to parser when it matches “INTEGER”.
- *FLOAT :* This token is returned to parser when it matches “FRACTION”.
- *CHAR :* This token is returned to parser when it matches “CHAR”.
- *START :* This token is returned to parser when it matches “((”.
- *END :* This token is returned to parser when it matches “))”.
- *SWITCH :* This token is returned to parser when it matches “SWITCH”.

- *CASE :* This token is returned to parser when it matches “CASE”.
- *DEFAULT :* This token is returned to parser when it matches “DEFAULT”.
- *BREAK :* This token is returned to parser when it matches “BREAK”.
- *FOR :* This token is returned to parser when it matches “LOOP”.
- *PF :* This token is returned to parser when it matches “out”.
- *SIN :* This token is returned to parser when it matches “SIN”.
- *COS :* This token is returned to parser when it matches “COS”.
- *TAN :* This token is returned to parser when it matches “TAN”.
- *LOG :* This token is returned to parser when it matched “LOG”.
- *LOG10:* This token is returned to parser when it matched “LOG10”.
- *PLUS :* This token is returned to parser when it matched “add”.
- *MINUS :* This token is returned to parser when it matched “sub”.
- *MULTI :* This token is returned to parser when it matched “mul”.
- *DIV :* This token is returned to parser when it matched “div”.
- *FACTIORIAL :* This token is returned to parser when it matched “FACT”.
- *POW :* This token is returned to parser when it matched “pow”.
- *ODDEVEN :* This token is returned to parser when it matches “isEVEN”.
- *PRIME :* This token is returned to parser when it matches “isPRIME”.
- *FIBBO :* This token is returned to parser when it matches “FIBB”.

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.

My custom CFG is:

program: MAIN ':' START new END

;

new: /* NULL */

| new statement

;

statement:

| declaration ';

| expression ';

| VAR '=' expression ';'

| FOR '(' NUM '<' NUM ')' START statement END

| IF '(' expression ')' START expression ';' END

| PF '(' expression ')' ';'

declaration : TYPE ID1

;

TYPE : INT

| FLOAT

| CHAR

```
    ;

expression: NUM
          | VAR
          | expression 'PLUS' expression
          | expression 'MINUS' expression
          | expression 'MULTIPLY' expression
          | expression 'DIV' expression
          | expression '%' expression
          | expression 'pow' expression
          | expression '<' expression
          | expression '>' expression
          | '(' expression ')'
          | SIN expression
          | COS expression
          | TAN expression
          | LOG expression
          | FACTIORIAL expression
          | ODDEVEN expression
          | PRIME expression
          | FIBBO expression
```

Features:

Main Function:

Every thing should be inside the main function other wise the program will not execute.it is followed by : and “((“ “))” .

Valid Syntax:

```
 body :
 ((
  statement ;
));
```

Arithmetic operators :

Operator Sign	**Operation**	**Syntax**
+	Addition	3 add 5
-	Subtraction	7 sub 6
*	Multiplication	2 mul 4
/	Division	8 div 2
^	X to the power of n	5 pow 3

Assignment & Conditional operators

Operator sign	**Operation**	**Syntax**
<	Less than	3 < 5
>	Greater than	7 > 6
<=	Less than or equal to	2 <= 4
>=	Greater than or equal to	8 >= 2

Data types:

There are 3 data types available in this compiler.

TYPES	Syntax
Integer : any numbers	Integer a,b,c;
Character : any characters	Char a,b,c;
Float : decimal point number	Frac a,b,c

Headers:

Headers are available in this compiler. It starts with insert and ends with .h file extension.

Valid syntax : insert math.h

Comments:

A single line comment starts with the symbol '#' and after this symbol any letter or digits or special character in the particular line will be ignored by the compiler

Valid Syntax: # This is a single line comment

Multiline comment syntax: #! This is a multi line comment !#

If - else

The structure of if else is quite similar with that of the C programming language. The keyword if is used with parenthesis that takes expressions and upon evaluating the expression if it gives a valid result

then it enters the following curly braces, otherwise it moves on to the next set of curly braces to perform further instructions.

Valid Syntax: IF (4 < 5) { #executing if block }
FI { #executing else block }

Loop:

A loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

The syntax of loop in this language is –

```
LOOP ( init; condition) ((
  statement(s);
))
```

Switch-Case:

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.

Syntax

The syntax for a **switch** statement is as follows

```
SWITCH(expression) ((
```

```
CASE constant-expression  :
  statement(s);
  BREAK; /* optional */
CASE constant-expression  :
  statement(s);
  BREAK; /* optional */
/* you can have any number of case statements */
DEFAULT: /* Optional */
statement(s);))
```

Printing newline

We are to use the NL(); keyword to add a newline to our code. Here NL refers to newline.

Assigning values in a variable

We can initialize a value inside a variable while we declare it. We can also provide expressions to initialize a variable. Previously initialized variables can also be copied to another variable using the assignment operator. For example, the following styles are considered valid:

INTEGER a = 30, p = 4 add 5 ; etc .

Built in Functions:

My custom Compiler have some built in functions :

isEven(arg): Returns if a function is even or odd.

isPrime(arg): Returns if the number is prime or not.

FIBB(arg) : print the Fibbonacci series up to the argument provided.

Fact(arg) : print the factorial of the argument provided.

Trigonometric Functions:

SIN(arg)

COS(arg)

TAN(arg)

LOG(arg)

LOG10(arg)

These functions returns trigonometric values.

Code Execution:

Type this in the command line to run the code:

bison –d “filename”.y

flex “L filename”.l

gcc lex.yy.c “filename”.tab.c –o “anyname”

“anyname”