Lab Report

Course Name: Compiler Design Laboratory
Course No: CSE 3212

Submitted to:

- Dola Das
  Lecturer,
  Department of Computer Science and Engineering,
  KUET

- Md. Ahsan Habib Nayan
  Lecturer,
  Department of Computer Science and Engineering,
  KUET

Submitted By:

Name: Sazia Rahman
Roll: 1707012
Department of Computer Science and Engineering
KUET

Submission Date: 15 – 06 - 2021

Introduction:

Flex: Flex is a tool for generating programs that recognize lexical patterns in text. Flex takes a program written in a combination of Lex and C language and then divide the input into meaningful units called token and it writes out a file (called lex.yy.c) that holds a definition of function yylex(). Each time yylex() is called it returns the next token and after it is finished reading tokens it call yywrap() function . The scanning of the input file finishes when yywrap() function returns 1 which means end of file. Lex file is saved with an extension l. The file pattern is given below

```
{definitions }
%%
{ rules }
%%
{user subroutines}
```

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. It takes a specification of a syntax as input, and produces a procedure for recognizing that language as output. Bison is compatible with yacc and the input file for bison is saved with the extension y. Bison generates two output file one is a (filename.tab.c) file and another is (filename.tab.h) file which is used as the header in lex file. Format of bison input file is given below

```
%{ C declarations (types, variables, functions, preprocessor commands)   %}
/* Bison declarations (grammar symbols, operator precedence decl., attribute data type) */
%%
/* grammar rules go here */
%%
/* additional C code goes here */
```

DESCRIPTION: Flex divides input into meaningful token. Bison takes this token and find relations and does operations related these token. The token that I have used in this compiler is given below:

MAIN = This token is returned when “function main()” pattern is found.\

START = This token is returned when “start” pattern is found.

END = This token is returned when “end” pattern is found.

IF = This token is returned when “iff” pattern is found.

ELIF = This token is returned when “elif” pattern is found.

ELSE = This token is returned when “else” pattern is found.

SWITCH = This token is returned when “switch” pattern is found.

CASE = This token is returned when “case” pattern is found.

DEF = This token is returned when “default” pattern is found.

BREAK = This token is returned when “break” pattern is found.

FOR = This token is returned when “for” pattern is found.

WHILE = This token is returned when “while” pattern is found.

evenodd = This token is returned when “evenodd” pattern is found.

Factorial = This token is returned when “factorial” pattern is found.

prime = This token is returned when “prime” pattern is found.

sin = This token is returned when “sin” pattern is found.

cos = This token is returned when “cos” pattern is found.

tan = This token is returned when “tan” pattern is found.

log = This token is returned when “log” pattern is found.

SQRT = This token is returned when “squaroot” pattern is found.

power = This token is returned when “power” pattern is found.

gcd = This token is returned when “gcd” pattern is found.

INT = This token is returned when “integer” pattern is found.

FLOAT = This token is returned when “float” pattern is found.

CHAR = This token is returned when “character” pattern is found.

SHOW = This token is returned when “show” pattern is found.

COND = This token is returned when “condition” pattern is found.

INCREMENT = This token is returned when “inc” pattern is found.

DECREMENT = This token is returned when “dec” pattern is found.

GT = This token is returned when “gt” pattern is found.

LT = This token is returned when “lt” pattern is found.

EQ = This token is returned when “eq” pattern is found.

NEQ = This token is returned when “neq” pattern is found.

NUMBERI = This token is returned when [-+]?{digit}+ pattern is found.

Example: 3 , -4 etc

NUMBERF = This token is returned when [-+]?{digit}+ “.” {digit}* pattern is found. Example : 3.80 ,-10.3 etc.

VARIABLE = This token is returned when [a-zA-z][a-zA-Z0-9]pattern is found.

[-+*/()#<> @ : = ; ,] will return the whole text as a token.

[\t\n] any space tab or newline will be ignored.

Anything found other than above patterns will show the error Invalid input character.

DATA TYPE: There are 2 data types.

1. Integer
2. Float

MAIN FUNCTION: In this project the name of the main function is “function main()”. The statements of main function are written inside a block. The block begins with “start” and it finishes with “end”.

The format of main function is given below:

```
function main()
start
    /*statements*/
end
```

STATEMENTS : inside main function we can write many statements. Each of these statements are associated with a specific types of action.

- **Declaration :** we can declare a variable of any datatype described above and also assign value to it in the same line .
- **Assign :** we can assign value to a variable which has been declared before. We can also store value to a variable after doing arithmetic or logical operations.
- **Output:** we can print the value of any variable.
- **Condition:** in condition sataements we can use if else block or switch block to do any operation and print the value.
- **Loops :** we can use for lopp and while loop to do any repeated work.
- Built in function: In this project there are some built-in function which can do the operation and show related outputs just by passing value to the function.
- **Calculation:** any arithmetic and logical operation can be done in this project.

Syntax of the statements:

Declaration and assignment: multiple variable can be declared and assign in the same line as well as in different lines. ‘;’ ends the line.

Example: integer abc, d=10;

```
abc=12;
```

output: we can print the value of a variable. The variable is passed to the function name show and end with a semicolon.

Example: show(abc);

If_else block: if_else block begins and finished with ‘start’ and ‘end’ sign.

```
Iff < ‘value’ >
   start
       /* operation*/
   end
elif < ‘value’ >
   start
       /* operation*/
   end
else
   start
       /* operation*/
   end
```

Switch-case block: The pattern of switch case block is :

```
switch < ‘value’  > :
start
          case ( ‘value’ ) :
           /* operation*/
           break

          case ( ‘value’ ) :
           /* operation*/
           break

          default :
             /* operation*/
           break
end
```

In switch-case block we can also assign value to a certain variable when the condition is matched. Then we can print the value after switch case block has ended.

For loop: The for loop syntax in this project is similar to the for loop of C language. It keep printing the value of the statement until the initial value is equal to the final value. The pattern of for loop is :

```
for ‘variable’ : “initial value”  to “final value”
start
   /*operation*/
end
```

While loop: in the while loop we have to check the condition or value first . if the value is greater than 0 then the loop will execute and print the value of the statement. After each time executing the statement it will ckeck the condition. Loop will be ended when the condition is no longer satisfied.
The pattern of while loop is :

```
while < “value or condition” >
start
  /* operation*/
 condition
 /*increment or decrement*/
 end
```

Built-in functions: there are many bult-in functions in this project

1. **evenodd(a)** : checks if the value of the variable is even or odd. This function ends with a semicolon.
2. **factorial(a)** : prints the factorial of the given variable or value. It also ends with a semicolon.
3. **sin(a)** : does the sine operation and print the value of the operation. It also ends with a semicolon.
4. **cos(a)** : does the cosine operation and print the value of the operation. It also ends with a semicolon.
5. **tan(a)** : does the tangent operation and print the value of the operation. It also ends with a semicolon.
6. **log(a)** : does the logarithm operation and print the value of the operation It also ends with a semicolon.
7. **squaroot(a)** : prints the square root of the given variable or value. It also ends with a semicolon.

8. **gcd (a,b)** : prints the gcd of the given variables or values. It also ends with a semicolon.
9. **power(a,b)** : does the pow(a,b) operation and print the value of the operation. It also ends with a semicolon.
10. **prime(a)** : checks if the value of the variable is prime or non-prime. This function ends with a semicolon.

Arithmatic operations :

All the arithmetic operation are done with the syntax similar to C language.
Example: a + b , a – b , a * b , a / b;

All these operators are binary operator.

Only increment and decrement operator are unary operators.
Example: inc a , dec a

Logical operations :

• greater than :	a gt b
• less than:	a lt b
• equal:	a eq b
• not equal:	a neq b

All these operators are binary.

Precedence of the operators: The list of the operators according to precedency from higher to lower are given below:

• < >	:no associativity
• All logical operators	:left associativity
• * /	:left associativity
• + -	:left associativity