YACC

1) Parser for IF-THEN Statements

Program:

(if.l)

```
ALPHA [A-Za-z]
DIGIT [0-9]
%%
[ \t\n]
if                return IF;
then                return THEN;
{DIGIT}+            return NUM;
{ALPHA}({ALPHA}|{DIGIT})*         return ID;
"<="                return LE;
">="                return GE;
"=="                return EQ;
"!="                return NE;
"||"                return OR;
"&&"                return AND;
.                return yytext[0];
%%
```

(if.y)

```
%{
#include <stdio.h>
#include <stdlib.h>
%}
%token ID NUM IF THEN LE GE EQ NE OR AND
%right '='
%left AND OR
%left '<' '>' LE GE EQ NE
%left '+''-'
%left '*''/'
%right UMINUS
%left '!'
%%
S   : ST {printf("Input accepted.\n");exit(0);};
ST  :    IF '(' E2 ')' THEN ST1';'
    ;
ST1 : ST
    | E
    ;
E   : ID'='E
    | E'+'E
    | E'-'E
    | E'*'E
    | E'/'E
    | E'<'E
    | E'>'E
    | E LE E
    | E GE E
    | E EQ E
    | E NE E
    | E OR E
    | E AND E
    | ID
    | NUM
    ;
E2  : E'<'E
    | E'>'E
    | E LE E
    | E GE E
    | E EQ E
    | E NE E
    | E OR E
    | E AND E
    | ID
    | NUM
    ;
%%

#include "lex.yy.c"

main()
{
printf("Enter the statement: ");
yyparse();
}
```
Output:
```
nn@linuxmint ~ $ lex if.l
nn@linuxmint ~ $ yacc if.y
nn@linuxmint ~ $ gcc y.tab.c -ll -ly
nn@linuxmint ~ $ ./a.out
Enter the statement: if(i>) then i=1;
syntax error
nn@linuxmint ~ $ ./a.out
Enter the statement: if(i>8) then i=1;
Input accepted.
nn@linuxmint ~ $
```

2) **IMPLEMENTATION OF CALCULATOR USING LEX & YACC**

AIM:

To write a program for implementing a calculator for computing the given expression using semantic rules of the YACC tool and LEX.

ALGORITHM:

Step1: A Yacc source program has three parts as follows:

Declarations %% translation rules %% supporting C routines

Step2: Declarations Section: This section contains entries that:

i. Include standard I/O header file.

ii. Define global variables.

iii. Define the list rule as the place to start processing.

iv. Define the tokens used by the parser. v. Define the operators and their precedence.

Step3: Rules Section: The rules section defines the rules that parse the input stream. Each rule of a grammar production and the associated semantic action.

Step4: Programs Section: The programs section contains the following subroutines. Because these subroutines are included in this file, it is not necessary to use the yacc library when processing this file.

Step5: Main- The required main program that calls the yyparse subroutine to start the program.

Step6: yyerror(s) -This error-handling subroutine only prints a syntax error message.

Step7: yywrap -The wrap-up subroutine that returns a value of 1 when the end of input occurs. The calc.lex file contains include statements for standard input and output, as programmar file information if we use the -d flag with the yacc command. The y.tab.h file contains definitions for the tokens that the parser program uses.

Step8: calc.lex contains the rules to generate these tokens from the input stream.

PROGRAM CODE:

//*Implementation of calculator using LEX and YACC*

LEX PART:

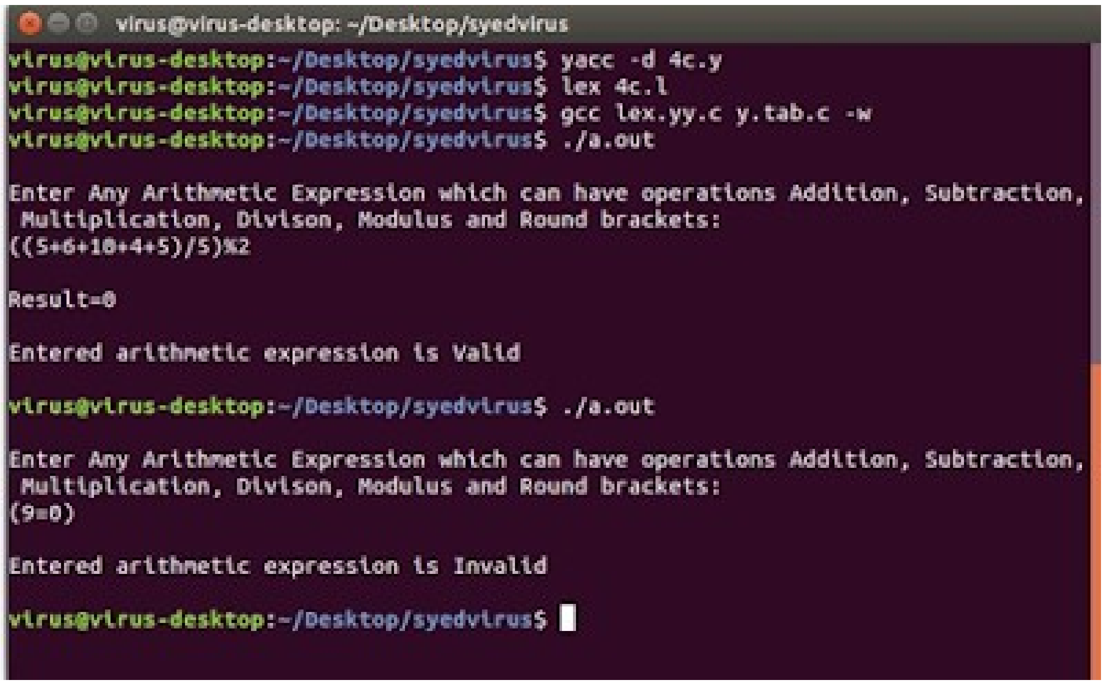
```
%{

#include<stdio.h>
#include "y.tab.h"

extern int yylval;

%}


%%

[0-9]+ {

          yylval=atoi(yytext);

          return NUMBER;

       }

[\t] ;

[\n]        return 0;

. return yytext[0];

%%

int yywrap()

{

return 1;

}
```

YACC PART:

```
%{

   #include<stdio.h>

   int flag=0;


%}

%token NUMBER

%left '+' '-'

%left '*' '/' '%'
```

```
%left '(' ')'

%%

ArithmeticExpression: E{

         printf("\nResult=%d\n",$$);

         return 0;

        };

E:E'+'E {$$=$1+$3;}

 |E'-'E {$$=$1-$3;}

 |E'*'E {$$=$1*$3;}

 |E'/'E {$$=$1/$3;}

 |E'%'E {$$=$1%$3;}

 | NUMBER {$$=$1;}

;

%%

void main()

{

   printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction,
Multiplication, Divison, Modulus and Round brackets:\n");

   yyparse();

  if(flag==0)

   printf("\nEntered arithmetic expression is Valid\n\n");



}

void yyerror()
{
printf("\nEntered arithmetic expression is Invalid\n\n");
flag=1;
}
```

OUTPUT:



```
virus@virus-desktop: ~/Desktop/syedvirus
virus@virus-desktop:~/Desktop/syedvirus$ yacc -d 4c.y
virus@virus-desktop:~/Desktop/syedvirus$ lex 4c.l
virus@virus-desktop:~/Desktop/syedvirus$ gcc lex.yy.c y.tab.c -w
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction,
 Multiplication, Divison, Modulus and Round brackets:
((5+6+10+4+5)/5)%2

Result=0

Entered arithmetic expression is Valid

virus@virus-desktop:~/Desktop/syedvirus$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction,
 Multiplication, Divison, Modulus and Round brackets:
(9=0)

Entered arithmetic expression is Invalid

virus@virus-desktop:~/Desktop/syedvirus$ 
```