Compiler Design Practical and Report File

Logarithmic, Exponential, and Trigonometric Calculator using LEX and YACC

Submitted By:
Mishti Dang - 101918029
Tanya Srivastava - 101918031
Ishita Mullick - 101918035

UNDER THE GUIDANCE OF: DR. RUPALI BHARDWAJ

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY, PATIALA

Index

1. Introduction to LEX and YACC
2. Lab Codes
3. Project Introduction
4. Grammar
5. LR(1) Canonical Items
6. LALR Parsing Tables for common inputs
7. Code (.lex and .yacc files)
8. Output screenshots

Introduction

<u>Lex</u>

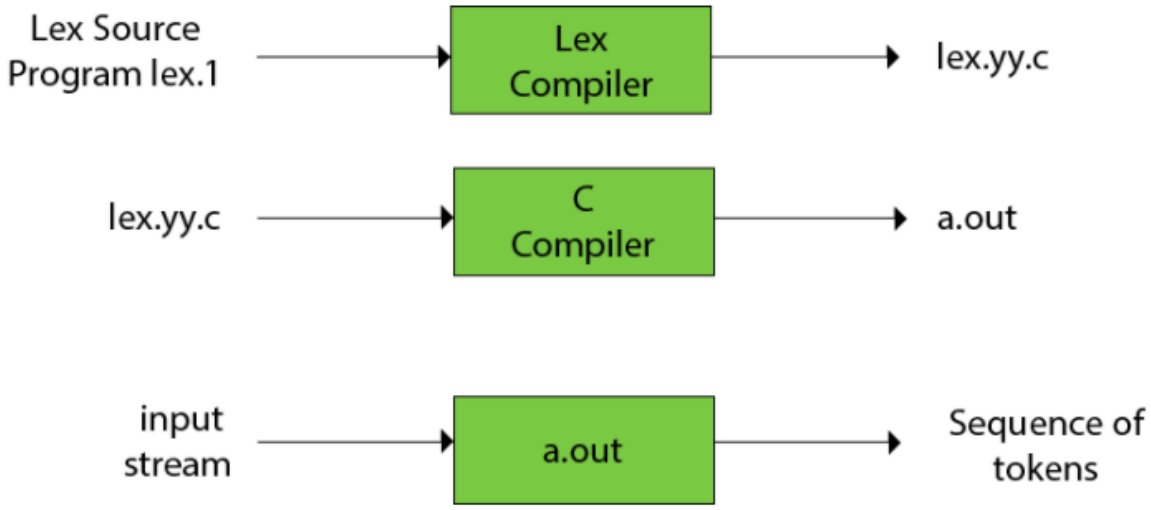
- Lex is a computer program that generates lexical analyzers and was written by Mike Lesk and Eric Schmidt. Lex **reads an input stream** specifying the lexical analyzer and outputs source code implementing the lexer in the C programming language.
- Lex is a program that **generates lexical analyzers**. It is used with the YACC parser generator.
- The lexical analyzer is a program that transforms an input stream into a sequence of tokens.
- It reads the input stream and produces the source code as output through implementing the lexical analyzer in the C program.
- The function of Lex is as follows:
    - Firstly, lexical analyzer creates a program lex.l in the Lex language.
    - Then the Lex compiler runs the lex.l program and produces a C program lex.yy.c.
    - Finally C compiler runs the lex.yy.c program and produces an object program a.out.

  a.out is a lexical analyzer that transforms an input stream into a sequence of tokens.

A Lex program is separated into **three sections** by %% delimiters. The formal of Lex source is as follows:

{ definitions }
%%
{ rules }
%%
{ user subroutines }
Definitions include declarations of constant, variable and regular definitions.

Rules define the statement of form p1 {action1} p2 {action2}....pn {action}.



Where pi describes the regular expression and action1 describes the actions the lexical analyzer should take when pattern pi matches a lexeme.

<u>Yacc</u>

- The name YACC is an acronym for "Yet Another Compiler Compiler." YACC is a specialized compiler that **generates source code** for part of a compiler you're trying to construct. Specifically, YACC is a parser generator, a software tool that helps automate a portion of compiler development, namely building the parser.
- It generates source code for a **LALR** (look-ahead left-to-right) parser that consists primarily of grammar rules and actions to be performed when those grammar rules are recognized.
- As output, YACC typically generates **C source code**, which then becomes part of the source code for the compiler we construct.

Commands used to run LEX files:

```
lex lexfilename.l
cc lex.yy.c -ll
./a.out
```

Commands used to run LEX with YACC files:

```
lex lexfilename.l
yacc -d yaccfilename.y
cc y.tab.c lex.yy.c          //compiling the 2 separately compiled files together
./a.out
```

Lab Codes

List of Codes present in this file -

1. If the user input is Hi, print Hello.
2. Write a code to check whether the input number is even or odd.
3. Write a program to check whether the input is a keyword or identifier or neither.
4. Give a greeting if the user enters Yes (Good Morning)
5. Check whether a word contains a vowel or not.
6. Check if the input string is present in the Kleene closure of ‘a’.
7. Read file and print the number of identifiers.
8. Calculator using functions
9. Calculator

Q1) If the user input is Hi, print Hello.

```
%{ #include<stdio.h>
%}
%%
"hi"      { printf("Hello!\n"); return 1; }
.*        { printf("Random!"); return 1; }
%%
int main()
{
        printf("Enter input: ");
        yylex();
        return 0;
}
int yywrap()
{
        return 1;
}
```

Output -

```
(base) ishita@Ishitas-MacBook-Air lex % ./a.out
Enter input: hi
Hello!
(base) ishita@Ishitas-MacBook-Air lex % ./a.out
Enter input: bye
Random!%
```

Q2) Write a code to check whether the input number is even or odd.

```
%{ #include<stdio.h>
```

```
        int m;
%}
%%
[0-9]+  { m=atoi(yytext);
          if(m%2==0)
                printf("%s is even\n",yytext);
          else
                printf("%s is odd\n",yytext);
          return 0;
        }
.*      { printf("Enter valid number! Try again.\n"); }
%%
int main()
{
        printf("Enter input: ");
        yylex();
}
int yywrap(){
        return 1;
}
```

Output -

```
(base) ishita@Ishitas-MacBook-Air lex % ./a.out
Enter input: 5
5 is odd
(base) ishita@Ishitas-MacBook-Air lex % ./a.out
Enter input: 6
6 is even
```

Q3) Write a program to check whether the input is a keyword or identifier or neither.

```
%{ #include<stdio.h>
%}
letter [a-z|A-Z]
digit [0-9]
%%
"if"|"else"|"for"|"while"|"break"|"continue"|"void"|"int" { printf("%s is a keyword",yytext); return 0; }
{letter}({letter}|{digit})* { printf("%s is a valid identifier",yytext); return 0; }
.*  { printf("%s is neither a keyword nor identifier.",yytext); return 0;}
%%
int main()
{
  printf("Input string to find if it's is a keyword or an identifier: ");
```

```
  yylex();
  return 0;
}
int yywrap()
{
  return 1;
}
```

Output -

```
(base) ishita@Ishitas-MacBook-Air lex % ./a.out
Input string to find if it's is a keyword or an identifier: int
int is a keyword%
(base) ishita@Ishitas-MacBook-Air lex % ./a.out
Input string to find if it's is a keyword or an identifier: ishita123
ishita123 is a valid identifier%
(base) ishita@Ishitas-MacBook-Air lex % ./a.out
Input string to find if it's is a keyword or an identifier: 1ishita
1ishita is neither a keyword nor identifier.%
```

Q4) Give a greeting if the user enters Yes!

```
%{ #include<stdio.h>
char name[20];
%}
%%
("Y"|"y")("E"|"e")("S"|"s") {
                          printf("Enter your name: ");
                          scanf("%s",name);
                          printf("Good morning %s",name);
                         }
("N"|"n")("O"|"o")  { printf("Okay bye!"); return 0;}
.*  { printf("Wrong input, try again!");  }
%%
int main()
{
  printf("Do you want a greeting? ");
  yylex();
}
int yywrap()
{
  return 1;
}
```

Output -

```
Do you want a greeting? yES
Enter your name: ISHITA
Good morning ISHITA

Yes
Enter your name: Ishita
Good morning Ishita

yes
Enter your name: IshiTA
Good morning IshiTA

nO
Okay bye!%
```

Q5) Check whether a word contains a vowel or not.

```
%{ #include<stdio.h>
char word[20];
%}
vowel [a,e,i,o,u,A,E,I,O,U]
letter [a-z|A-Z]
%%
{letter}*{vowel}{letter}*   { printf("%s contains a vowel",yytext); return 0; }
.* {  printf("%s doesn't contain a vowel",yytext); return 0; }
%%
int main()
{
  printf("Enter word to find out if it contains atleast one vowel: ");
  yylex();
}
int yywrap()
{
  return 1;
}
```

Output -

```
Enter word to find out if it contains atleast one vowel: ishita
ishita contains a vowel%
(base) ishita@Ishitas-MacBook-Air lex % ./a.out
Enter word to find out if it contains atleast one vowel: myth
myth doesn't contain a vowel%
```

Q6) Check if the input string is present in the Kleene closure of a.

```
%{  #include<stdio.h>
%}
letter [a]
%%
{letter}*|"^" { printf("%s is present in the kleene closure",yytext); return 0; }
.*  { printf("%s is not present in the Kleene closure",yytext); return 0; }
%%
int main()
{
  printf("Enter string to check if it is present in the Kleene closure of a: ");
  yylex();
  return 0;
}
int yywrap()
{
  return 1;
}
```

Output -

```
Enter string to check if it is present in the Kleene closure of a: ^
^ is present in the kleene closure%
(base) ishita@Ishitas-MacBook-Air lex % ./a.out
Enter string to check if it is present in the Kleene closure of a: aaaaaaa
aaaaaaa is present in the kleene closure%
(base) ishita@Ishitas-MacBook-Air lex % ./a.out
Enter string to check if it is present in the Kleene closure of a: baaa
baaa is not present in the Kleene closure%
```

Q7) Read file and print the number of identifiers.

LEX File

```
%{      #include<stdio.h>
        #include "y.tab.h"
          int yylval;
%}
%%
[}|=|,|%|(|)|#|.|{|+|;|"|+|-|*|/|>|<] { printf("Symbol is %s\n",yytext); return OP; }
[0-9]+  { yylval=atoi(yytext);
        printf("Number is %d\n",yylval);
        return DIGIT; }
int|char|for|do|while    {
                         printf("Keyword is %s\n",yytext);
                         return KW;
```

```
   }
[a-z|A-Z][a-z|A-Z|0-9]*{
                                printf("Identifier is %s\n",yytext);
                                return ID;
   }
%%
int yywrap()
{
        return 1;
}
```

YACC File

```
%{      #include<stdio.h>
        int id=0,dig=0,key=0,op=0;
        int yyerror();
        int yylex();
        extern FILE* yyin;
%}
%token DIGIT ID KW OP
%start input
%%
input : DIGIT { }
        | ID { }
        | KW { }
        | OP { }
        ;
%%
int main()
{
        FILE *fp;
        fp=fopen("freqofnos.c","r"); //change filename
        if(fp==NULL)
        {
                printf("Error");
                return 0;
        }
        yyin=fp;

        for(int i=0;i<50;i++)  // first 50 lines of code
                yyparse();
        return 0;
}
```

```
int yyerror()
{
        printf("Invalid string\n");
        return 0;
}
```

Output -

```
Symbol is #
Identifier is include

Symbol is <
Identifier is stdio

Symbol is .
Identifier is h

Symbol is >

Identifier is struct

 Identifier is dataset
Symbol is {


        Keyword is int
 Identifier is num

Symbol is ;

        Keyword is int

 Identifier is count
Symbol is ;


        Keyword is int
 Identifier is flag

Symbol is ;
```
...

Q8) Calculator using functions

```
%{
        #include<stdio.h>
        void sum();
        void sub();
        void mul();
        void divi();
%}

%%
[+] {sum();}
[-] {sub();}
[*] {mul();}
[/] {divi();}
%%
```

```
void sum()
{
	int num1,num2;
	printf("Enter the numbers\n");
	scanf("%d%d",&num1,&num2);
	printf("%d is the answer",num1+num2);
}
void sub()
{
	int num1,num2;
	printf("Enter the numbers\n");
	scanf("%d%d",&num1,&num2);
	printf("%d is the answer",num1-num2);
}
void mul()
{
	int  num1,num2;
	printf("Enter the numbers\n");
	scanf("%d%d",&num1,&num2);
	printf("%d is the answer",num1*num2);
}

void divi()
{
	int num1,num2;
	printf("Enter the numbers\n");
	scanf("%d%d",&num1,&num2);
	printf("%d is the answer",num1/num2);
}

int main()
{
	printf("enter any operation(+,-,*,/)");
	yylex();
	return 0;
}

int yywrap()
{
	return 1;
}
```

Output -

```
enter any operation(+,-,*,/)/
Enter the numbers
5
9
0 is the answer

*
Enter the numbers
4
3
12 is the answer

-
Enter the numbers
3
1
2 is the answer

+
Enter the numbers
6
5
11 is the answer
```

Q9) Calculator (+/-/*//)

<u>YACC file -</u>

```
%{ #include<stdio.h>
int yylex(void);
void yyerror(char *);
%}
%token INTEGER
%%
program : program expr '\n' {printf("%d\n",$2);}
|;
expr: INTEGER{$$ = $1;} | expr '+' expr {$$ = $1 + $3;} |
expr '-' expr {$$ = $1 - $3;} | expr '/' expr {$$ = $1 / $3;} |
expr '*' expr {$$ = $1 * $3;}
%%
void yyerror(char *s) {
fprintf(stderr,"%s\n",s);
}
int main(void) {
yyparse();
return 0;
}
```

<u>LEX File -</u>

```
%{ #include<stdio.h>
#include "y.tab.h"
#include<stdlib.h>
```

```
void yyerror(char *);
extern int yylval;
%}
%%
[0-9]+ {yylval = atoi(yytext); return INTEGER;}
[-/+*\n] return *yytext;
. yyerror("invalid character");
%%
int yywrap(void) { return 1; }
```

Output -

```
3+4-5*2
-3
1/3
0
```

Project Part 1

Project Introduction

In this project, we try to conceive a grammar to accept a certain syntax of input. This input is in the form of a logarithmic expression which can be evaluated into a single numerical output. The grammar should only accept the input following the given syntax-

log expression expression
expression expression
sin expression
cos expression
tan expression

Here, an expression denotes a combination of 4 operators and numbers. All of these nuances of our project are further explained in detail in the grammar section. We then encode these rules in the .lex and .yacc files (both inclusive in the project file).

Grammar

This the the syntax that we are trying to conceive a grammar of -
log expression expression
expression expression
sin expression
cos expression
tan expression

After careful consideration and hit and trial methods, we have come up with the following grammar -

S' -> S ……..(0)
S -> log E E…….....(1)
S -> E E…….(2)
S -> trig E……..(3)
E -> E + E……..(4)
E -> E - E……...(5)
E -> E * E……...(6)
E -> E / E…….(7)
E -> d……...(8)
trig -> sin……..(9)
trig -> cos…….(10)
trig -> tan……..(11)

Here, S is the starting symbol. The description of all the terminals used are given in Table 1 and Table 2.

Non-Terminal	**Explanation**
S	Starting Non-Terminal
trig	Denotes a trigonometric function
E	Denotes Expression

Table 1 - List of Non-Terminals

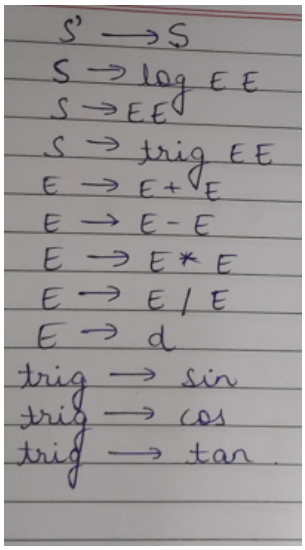
Terminal	**Explanation**
log	Logarithmic function
d	Digits Denotes a combination of digits {0,1,2,3,4,5,6,7,8,9}*
+	Addition operator
-	Subtraction operator
*	Multiplication operator
/	Division operator
sin	Trigonometric function (input in degrees)
cos	Trigonometric function (input in degrees)
tan	Trigonometric function (input in degrees)

Table 2 - List of Terminals

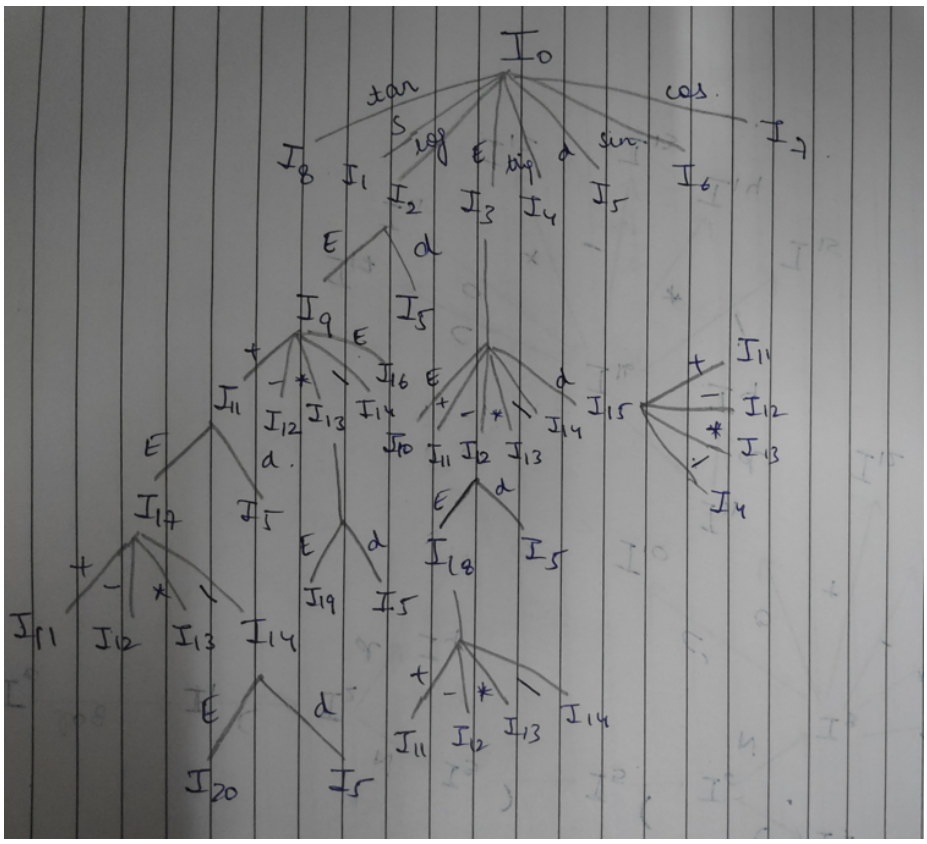
LR(1) Canonical Items

There are a total of 21 canonical items from I0 to I20. Due to the complex nature of the calculation of the items, we have included images of those items in this document.

Augmented Grammar



S' → S
S → log E E
S → E E
S → trig E E
E → E + E
E → E - E
E → E * E
E → E / E
E → d
trig → sin
trig → cos
trig → tan



Joins of canonical items

I0

$S' \rightarrow . S$
$S \rightarrow . \log E E$
$S \rightarrow . E E$
$S \rightarrow . \text{trig } E E$
$E \rightarrow . E + E$
$E \rightarrow . E - E$
$E \rightarrow . E * E$
$E \rightarrow . E / E$
$E \rightarrow . d$
$\text{trig} \rightarrow . \sin$
$\text{trig} \rightarrow . \cos$
$\text{trig} \rightarrow . \tan$

I1

$S' \rightarrow S. , \$$

I2

$S \rightarrow \log . E E , \$$
$E \rightarrow . E + E , d|+|-|*||$
$E \rightarrow . E - E , d|+|-|*||$
$E \rightarrow . E * E , d|+|-|*||$
$E \rightarrow . E / E , d|+|-|*||$
$E \rightarrow . d , d|+|-|*||$

I3

$S \rightarrow E \cdot E, \$$
$E \rightarrow E \cdot + E, d|+|-|*|/$
$E \rightarrow E \cdot - E, d|+|-|*|/$
$E \rightarrow E \cdot * E, d|+|-|*|/$
$E \rightarrow E \cdot / E, d|+|-|*|/$
$E \rightarrow \cdot E + E, \$|+|-|*|/$
$E \rightarrow \cdot E - E, \$|+|-|*|/$
$E \rightarrow \cdot E * E, \$|+|-|*|/$
$E \rightarrow \cdot E / E, \$|+|-|*|/$
$E \rightarrow \cdot d, \$|+|-|*|/$

I4

$S \rightarrow trig \cdot E, \$$
$E \rightarrow \cdot E + E, \$|+|-|*|/$
$E \rightarrow \cdot E - E, \$|+|-|*|/$
$E \rightarrow \cdot E * E, \$|+|-|*|/$
$E \rightarrow \cdot E / E, \$|+|-|*|/$
$E \rightarrow \cdot d, \$|+|-|*|/$

I5

$E \rightarrow d \cdot, d|+|-|*|/|\$$

I6

$trig \rightarrow sin \cdot, d$

I7

$trig \rightarrow cos \cdot, d$

I8

$trig \rightarrow tan \cdot, d$

I9

$S \rightarrow \log E \cdot E$, $

$E \rightarrow E \cdot + E$, d | + | - | * | /

$E \rightarrow E \cdot - E$, d | + | - | * | /

$E \rightarrow E \cdot * E$, d | + | - | * | /

$E \rightarrow E \cdot / E$, d | + | - | * | /

$E \rightarrow \cdot E + E$, $ | + | - | * | /

$E \rightarrow \cdot E - E$, $ | + | - | * | /

$E \rightarrow \cdot E * E$, $ | + | - | * | /

$E \rightarrow \cdot E / E$, $ | + | - | * | /

I10

$S \rightarrow E E \cdot$, $

$E \rightarrow E \cdot + E$, $ | + | - | * | /

$E \rightarrow E \cdot - E$, $ | + | - | * | /

$E \rightarrow E \cdot * E$, $ | + | - | * | /

$E \rightarrow E \cdot / E$, $ | + | - | * | /

I11

$E \rightarrow E + \cdot E$, d | + | - | * | / | $

$E \rightarrow \cdot E + E$, d | + | - | * | / | $

$E \rightarrow \cdot E - E$, d | + | - | * | / | $

$E \rightarrow \cdot E * E$, d | + | - | * | / | $

$E \rightarrow \cdot E / E$, d | + | - | * | / | $

I12

$E \rightarrow E - \cdot E$, d | + | - | * | / | $

$E \rightarrow \cdot E + E$, d | + | - | * | / | $

$E \rightarrow \cdot E - E$, d | + | - | * | / | $

$E \rightarrow \cdot E * E$, d | + | - | * | / | $

$E \rightarrow \cdot E / E$, d | + | - | * | / | $

I13

E → E * . E , d|+|-|*|/|$
E → . E + E , d|+|-|*|/|$
E → . E − E , d|+|-|*|/|$
E → . E * E , d|+|-|*|/|$
E → . E / E , d|+|-|*|/|$

I14

E → E / . E , d|+|-|*|/|$
E → . E + E , d|+|-|*|/|$
E → . E − E , d|+|-|*|/|$
E → . E * E , d|+|-|*|/|$
E → . E / E , d|+|-|*|/|$

I15

S → trig E . , $
E → E . + E , $|+|-|*|/
E → E . − E , $|+|-|*|/
E → E . * E , $|+|-|*|/
E → E . / E , $|+|-|*|/

I16

S → log E E . , $
E → E . + E , $|+|-|*|/
E → E . − E , $|+|-|*|/
E → E . * E , $|+|-|*|/
E → E . / E , $|+|-|*|/

I17

$E \rightarrow E + E.$, $d|+|-|*|/|\$$
$E \rightarrow E. + E$, $d|+|-|*|/|\$$
$E \rightarrow E. - E$, $d|+|-|*|/|\$$
$E \rightarrow E. * E$, $d|+|-|*|/|\$$
$E \rightarrow E. / E$, $d|+|-|*|/|\$$

I18

$E \rightarrow E - E.$, $d|+|-|*|/|\$$
$E \rightarrow E. + E$, $d|+|-|*|/|\$$
$E \rightarrow E. * E$, $d|+|-|*|/|\$$
$E \rightarrow E. - E$, $d|+|-|*|/|\$$
$E \rightarrow E. / E$, $d|+|-|*|/|\$$

I19

$E \rightarrow E * E.$, $d|+|-|*|/|\$$
$E \rightarrow E. + E$, $d|+|-|*|/|\$$
$E \rightarrow E. - E$, $d|+|-|*|/|\$$
$E \rightarrow E. * E$, $d|+|-|*|/|\$$
$E \rightarrow E. / E$, $d|+|-|*|/|\$$

I20

$E \rightarrow E / E.$, $d|+|-|*|/|\$$
$E \rightarrow E. + E$, $d|+|-|*|/|\$$
$E \rightarrow E. - E$, $d|+|-|*|/|\$$
$E \rightarrow E. * E$, $d|+|-|*|/|\$$
$E \rightarrow E. / E$, $d|+|-|*|/|\$$

LALR Parse Table for Common Inputs

Due to the complex nature of the calculation of the items, we have included images of the tables of common inputs, the same input have been included in the codes.

State	log	+	−	*	/	d	sin	cos	tan	$	S'	S	E	trig
0	s2					s5						1	3	4
1										acc				
2													9	
3		s11	s12	s13	s14	s5							10	
4						s5							15	
5		r8	r8	r8	r8	r8				r8				
6						r9								
7						r10								
8						r11								
9		s11	s12	s13	s14	s5							16	
10		s11	s12	s13	s14					r2				
11						s5							17	
12						s5							18	
13						s5							19	
14						s5							20	
15		s11	s12	s13	s14					r3				
16		s11	s12	s13	s14					r1				
17		s11	s12	s13	s14	r4				r4				
18		s11	s12	s13	s14	r5				r5				
19		s11	s12	s13	s14	r6				r6				
20		s11	s12	s13	s14	r7				r7				

Acceptance of input d - d d * d

Step	Stack	Input	Action
1	0	d - d d * d $	s5
2	0 d 5	- d d * d $	r8
3	0 E	- d d * d $	3
4	0 E 3	- d d * d $	s12
5	0 E 3 - 12	d d * d $	s5
6	0 E 3 - 12 d 5	d * d $	r8
7	0 E 3 - 12 E	d * d $	18
8	0 E 3 - 12 E 18	d * d $	r5
9	0 E	d * d $	3
10	0 E 3	d * d $	s5
11	0 E 3 d 5	* d $	r8
12	0 E 3 E	* d $	10
13	0 E 3 E 10	* d $	s13
14	0 E 3 E 10 * 13	d $	s5
15	0 E 3 E 10 * 13 d 5	$	r8
16	0 E 3 E 10 * 13 E	$	19
17	0 E 3 E 10 * 13 E 19	$	r6
18	0 E 3 E	$	10
19	0 E 3 E 10	$	r2
20	0 S	$	1
21	0 S 1	$	acc

Acceptance of input log d / d d * d

Step	Stack	Input	Action
1	0	log d/d d* d $	S2
2	0 log 2	d / d d* d $	S5
3	0 log 2 d 5	/ d d* d $	r8
4	0 log 2 E	/d d* d $	9
5	0 log 2 E 9	/d d* d $	S14
6	0 log 2 E 9/14	d d* d $	S5
7	0 log 2 E 9 /14 d 5	d * d $	r8
8	0 log 2 E 9/14 E	d * d $	20
9	0 log 2 E 9/14 E 20	d * d $	r7
10	0 log 2 E	d * d $	9
11	0 log 2 E 9	d * d $	S5
12	0 log 2 E 9 d 5	* d $	r8
13	0 log 2 E 9 E	* d $	16
14	0 log 2 E 9 E 16	*d $	S13
15	0 log 2 E9E 16 * 13	d $	S5
16	0 log 2 E 9E 16 * 13 d 5	$	r8
17	0 log 2 E 9 E 16 * 13 d 5	$	19
18	0 log 2 E 9 E 16 * 13 E 19	$	r6
19	0 log 2 E 9 E	$	16
20	0 log 2 E 9 E 16	$	r1
21	0 S	$	1
22	0 S 1	$	acc

Acceptance of input sin d * d

Step	Stack	Input	Action
1	0	sin d * d $	s6
2	0 sin 6	d * d $	r9
3	0 trig	d * d $	4
4	0 trig 4	d * d $	s5
5	0 trig 4 d5	* d $	r8
6	0 trig 4 E	* d $	15
7	0 trig 4 E 15	* d $	s13
8	0 trig 4 E 15 * 13	d $	s5
9	0 trig 4 E 15 * 13 d 5	$	r8
10	0 trig 4 E 15 * 13 E	$	19
11	0 trig 4 E 15 * 13 E 19	$	r6
12	0 trig 4 E	$	15
13	0 trig 4 E 15	$	r3
14	0 s	$	1
15	0 s 1	$	acc

<u>Code (.lex and .yacc files)</u>

.lex file

```
%{  #include<stdio.h>
    #include "y.tab.h"
    #include<stdlib.h>
    void yyerror(char *);
    extern int yylval;
%}
%%
sin return SIN;
cos return COS;
tan return TAN;
"^" return POW;
log  return LOG;
"("  return LPAR;
")"   return RPAR;
[0-9]+ {yylval = atoi(yytext); return NUMBER;}
[-/+*\n] return *yytext;
. yyerror("Incorrect symbol!");

%%
int yywrap(void)
{
  return 1;
}
```

.yaac file

```
%{ #include<stdio.h>
  #include<math.h>
  #include<stdlib.h>
int yylex(void);
void yyerror(char *);
%}

%token LOG NUMBER EXPO SIN COS TAN LPAR RPAR POW FLOAT
%left '+' '-'
%left '/' '*'
%%
program : LOG LPAR expr RPAR LPAR expr RPAR {
            if($3==1)
             {  printf("Log cannot be calculated for base = 1\n"); return 0; }
```

```
            printf("%f",log($6)/log($3)); return 0;
          } |
 LPAR expr RPAR POW LPAR expr RPAR { printf("%f",pow($2,$6));return 0;  } |
 SIN LPAR expr RPAR { printf("%f",sin(($3*3.14)/180)); return 0; } |
 COS LPAR expr RPAR { printf("%f",cos(($3*3.14)/180)); return 0; } |
 TAN LPAR expr RPAR { printf("%f",tan(($3*3.14)/180)); return 0; }
;

expr:
NUMBER {$$ = $1; } |
expr '+' expr {$$ = $1 + $3;} |
expr '-' expr {$$ = $1 - $3;} |
expr '/' expr {$$ = $1 / $3;} |
expr '*' expr {$$ = $1 * $3;}
;

%%
void yyerror(char *s) {
  fprintf(stderr,"%s\n",s);
}
int main() {
  yyparse();
  return 0;
}
```

Output screenshots

1. (7-3)^(2*2)

```
(base) ishita@Ishitas-MacBook-Air-2 desktop % ./a.out
(7-3)(2*2)
256.000000%
```

2. log(4/2)(8*2-4)

```
(base) ishita@Ishitas-MacBook-Air-2 desktop % ./a.out
log(4/2)(8*2-4)
3.584963%
```

3. sin(15*2)

```
(base) ishita@Ishitas-MacBook-Air-2 desktop % ./a.out
sin(15*2)
0.499770%
```

4. cos(90/2-15)

```
(base) ishita@Ishitas-MacBook-Air-2 desktop % ./a.out
cos(90/2-15)
0.866158%
```

5. tan(5*5+10/2)

```
(base) ishita@Ishitas-MacBook-Air-2 desktop % ./a.out
tan(5*5+10/2)
0.576996%
```

Project Part 2

Project Extension

To increase the applicability of the project, we added 2 functions - normal expression calculation and exponential function. Additionally, we integrated the acceptance (as input) and calculation of floating point numbers with the project.

Now, the grammar accepts the input following only the given syntax-

log expression expression
expression ^ expression
sin expression
cos expression
tan expression
e ^ expression
expression

As before, an expression denotes a combination of 4 operators and numbers. The extended functionality of the project includes the acceptance of floating point numbers as input. These nuances are further explained in detail in the grammar section.

Grammar

This the the syntax that we are trying to conceive a grammar of -

log expression expression
expression ^ expression
sin expression
cos expression
tan expression
e ^ expression
expression

After careful consideration and hit and trial methods, we have come up with the following grammar -

S' -> S ……..(0)
S -> log E E……....(1)
S -> E E……(2)
S -> E……..(3)
S -> e ^ E……...(4)
S -> trig E……..(5)

E -> E + E……..(6)
E -> E - E……...(7)
E -> E * E……...(8)
E -> E / E…….(9)
E -> f ……...(10)
trig -> sin……..(11)
trig -> cos…….(12)
trig -> tan……..(13)

Here, S is the starting symbol. The description of all the terminals used are given in Table 1 and Table 2.

Non-Terminal	**Explanation**
S	Starting Non-Terminal
trig	Denotes a trigonometric function
E	Denotes Expression

Table 1 - List of Non-Terminals

Terminal	**Explanation**
log	Logarithmic Function
f	Digits - Denotes a combination of the digits - {0,1,2,3,4,5,6,7,8,9}*
e	Exponential function
+	Addition operator
-	Subtraction operator
*	Multiplication operator
/	Division operator
^	Power operator
sin	Trigonometric function (input in degrees)
cos	Trigonometric function (input in degrees)
tan	Trigonometric function (input in degrees)

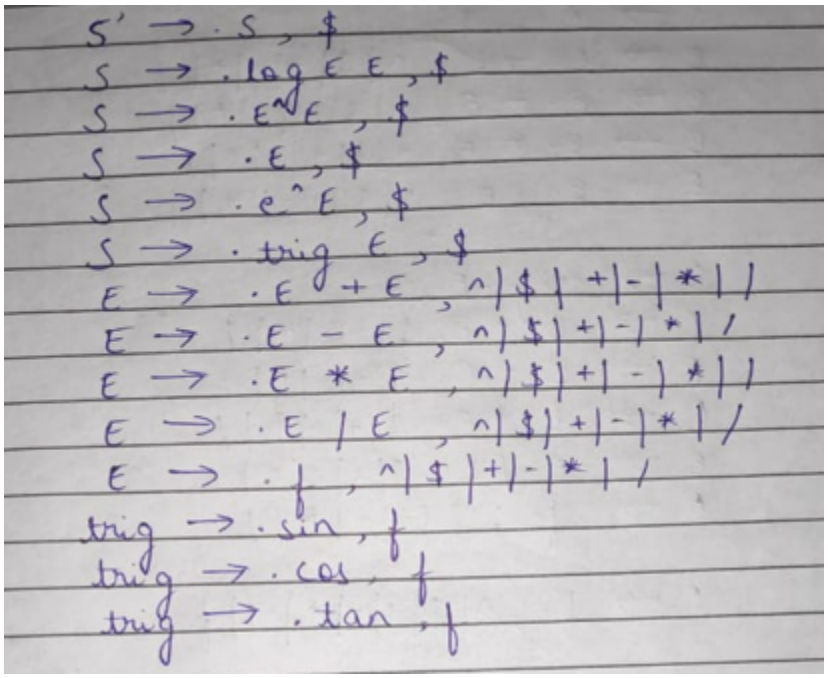
Table 2 - List of Terminals

LR(1) Canonical Items

There are a total of 25 canonical items from I0 to I24. Due to the complex nature of the calculation of the items, we have included images of those items in this document.

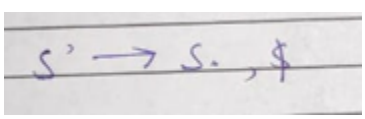
Augmented Grammar

I0



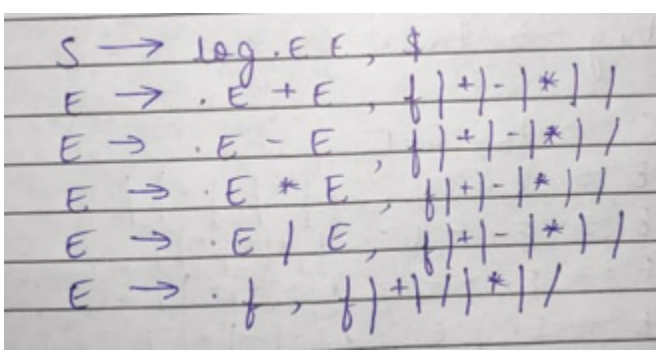
S' → . S , $
S → . log E E , $
S → . E^E , $
S → . E , $
S → . e^E , $
S → . trig E , $
E → . E + E , ^|$|+|-|*|/
E → . E − E , ^|$|+|-|*|/
E → . E * E , ^|$|+|-|*|/
E → . E / E , ^|$|+|-|*|/
E → . f , ^|$|+|-|*|/
trig → . sin , f
trig → . cos , f
trig → . tan , f

I1



S' → S. , $

I2



S → log . E E , $
E → . E + E , f|+|-|*|/
E → . E − E , f|+|-|*|/
E → . E * E , f|+|-|*|/
E → . E / E , f|+|-|*|/
E → . f , f|+|/|*|/

I3

$S \rightarrow E . \wedge E , \$$

$S \rightarrow E . , \$$

$E \rightarrow E . + E , \wedge | \$ | + | - | * | /$

$E \rightarrow E . - E , \wedge | \$ | + | - | * | /$

$E \rightarrow E . * E , \wedge | \$ | + | - | * | /$

$E \rightarrow E . / E , \wedge | \$ | + | - | * | /$

I4

$S \rightarrow e . \wedge E , \$$

I5

$S \rightarrow trig . E , \$$

$E \rightarrow . E + E , \$ | + | - | * | /$

$E \rightarrow . E - E , \$ | + | - | * | /$

$E \rightarrow . E * E , \$ | + | - | * | /$

$E \rightarrow . E / E , \$ | + | - | * | /$

$E \rightarrow . f , \$ | + | - | * | /$

I6

$E \rightarrow f . , \wedge | \$ | + | - | * | / | f$

I7

$trig \rightarrow sin . , f$

I8

trig → cos. , f

I9

trig → tan. , f

I10

S → log E. E , $
E → E. + E , f | + | - | * | /
E → E. − E , f | + | - | * | /
E → E. * E , f | + | - | * | /
E → E. / E , f | + | - | * | /
E → .E + E , $ | + | - | * | /
E → .E − E , $ | + | - | * | /
E → .E * E , $ | + | - | * | /
E → .E / E , $ | + | - | * | /
E → .f , $ | + | - | * | /

I11

S → E^.E , $
E → .E + E , $ | + | - | * | /
E → .E − E , $ | + | - | * | /
E → .E * E , $ | + | - | * | /
E → .E / E , $ | + | - | * | /
E → .f , $ | + | - | * | /

I12

E → E + .E , ^ | $ | + | - | / | * | f
E → .E + E , ^ | $ | + | - | * | / | f
E → .E − E , ^ | $ | + | - | * | / | f
E → .E * E , ^ | $ | + | - | * | / | f
E → .E / E , ^ | $ | + | - | * | / | f
E → .f , ^ | $ | + | - | * | / | f

I13

E → E − · E , ^|$|+|-|*|/|f
E → · E + E , ^|$|+|-|*|/|f
E → · E − E , ^|$|+|-|*|/|f
E → · E * E , ^|$|+|-|*|/|f
E → · E / E , ^|$|+|-|*|/|f
E → · f , ^|$|+|/|-|*|f

I14

E → E * · E , ^|$|+|-|*|/|f
E → · E + E , ^|$|+|-|*|/|f
E → · E − E , ^|$|+|-|*|/|f
E → · E * E , ^|$|+|-|*|/|f
E → · E / E , ^|$|+|-|*|/|f
E → · f , ^|$|+|-|*|/|f

I15

E → E / · E , ^|$|+|-|*|/|f
E → · E + E , ^|$|+|-|*|/|f
E → · E − E , ^|$|+|-|*|/|f
E → · E * E , ^|$|+|-|*|/|f
E → · E / E , ^|$|+|-|*|/|f
E → · f , ^|$|+|-|*|/|f

I16

S → e ^ · E , $
E → · E + E , $|+|-|*|/
E → · E − E , $|+|-|*|/
E → · E * E , $|+|-|*|/
E → · E / E , $|+|-|*|/
E → · f , $|+|-|*|/

I17

S → trig E. , $
E → E. + E , $|+|-|*||
E → E. − E , $|+|-|*||
E → E. * E , $|+|-|*||
E → E. | E , $|+|-|*||

I18

S → log E E. , $
E → E. + E , $|+|-|*||
E → E. − E , $|+|-|*||
E → E. * E , $|+|-|*||
E → E. | E , $|+|-|*||

I19

S → E ^ E. , $
E → E. + E , $|+|-|*||
E → E. − E , $|+|-|*||
E → E. * E , $|+|-|*||
E → E. | E , $|+|-|*||

I20

E → E + E. , ^|$|+|-|*|||f
E → E. + E , ^|$|+|-|*|||f
E → E. − E , ^|$|+|-|*|||f
E → E. * E , ^|$|+|-|*|||f
E → E. | E , ^|$|+|-|*|||f

I21

E → E - E . , ^ | $ | + | - | * | / |)

E → E . + E , ^ | $ | + | - | * | / |)

E → E . - E , ^ | $ | + | - | * | / |)

E → E . * E , ^ | $ | + | - | * | / |)

E → E . / E , ^ | $ | + | - | * | / |)

I22

E → E * E . , ^ | $ | + | - | * | / |)

E → E . + E , ^ | $ | + | - | * | / |)

E → E . - E , ^ | $ | + | - | * | / |)

E → E . * E , ^ | $ | + | - | * | / |)

E → E . / E , ^ | $ | + | - | * | / |)

I23

E → E / E . , ^ | $ | + | - | * | / |)

E → E . + E , ^ | $ | + | - | * | / |)

E → E . - E , ^ | $ | + | - | * | / |)

E → E . * E , ^ | $ | + | - | * | / |)

E → E . / E , ^ | $ | + | - | * | / |)

I24

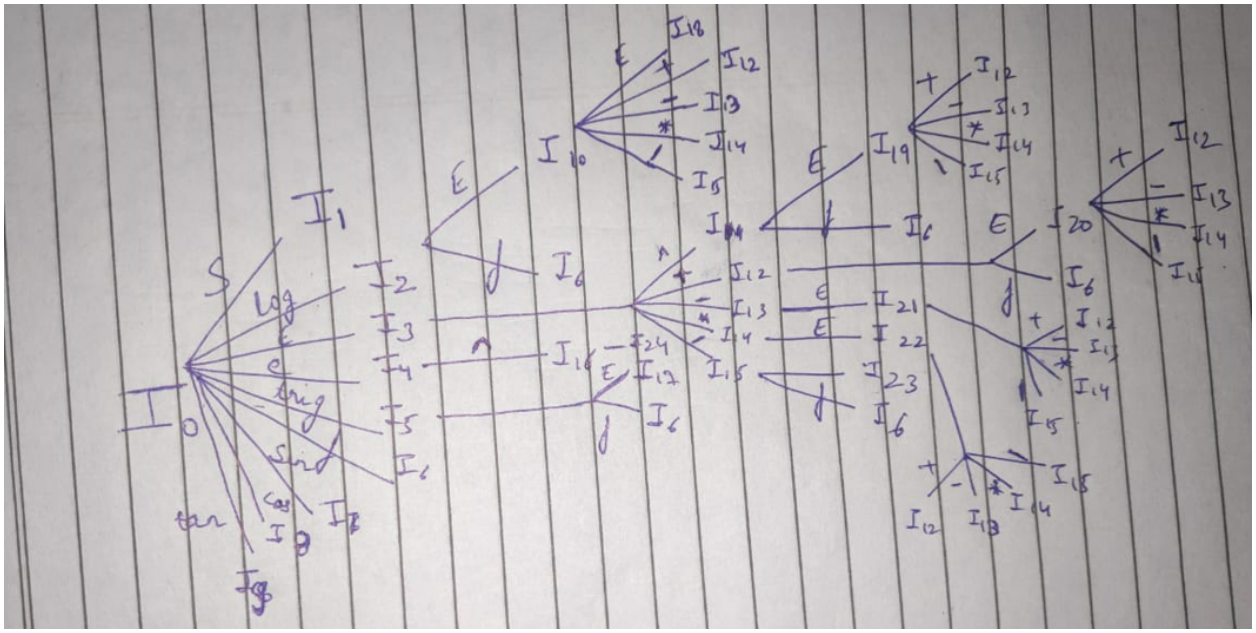
S → e ^ E . , $

E → E . + E , $ | + | - | * | /

E → E . - E , $ | + | - | * | /

E → E . * E , $ | + | - | * | /

E → E . / E , $ | + | - | * | /



Combined structure of Canonical Items

LALR Parse Table for Common Inputs

Due to the complex nature of the calculation of the items, we have included images of the tables of common inputs, the same input have been included in the codes.

	ACTION												GOTO			
State	log	^	e	+	-	*	/	f	sin	cos	tan	$	S'	S	E	trig
0	s2		s4					s6	s7	s8	s9			1	3	5
1												acc				
2								s6							10	
3		s11		s12	s13	s14	s15					r3				
4		s16														
5								s6							17	
6		r10		r10	r10	r10	r10	r10				r10				
7								r11								
8								r12								
9								r13								
10				s12	s13	s14	s15	s6							18	
11								s6							19	
12								s6							20	
13								s6							21	
14								s6							22	
15								s6							23	
16								s6							24	
17				s12	s13	s14	s15					r5				
18				s12	s13	s14	s15					r1				
19				s12	s13	s14	s15					r2				
20		r6		s12	s13	s14	s15	r6				r6				
21		r7		s12	s13	s14	s15	r7				r7				
22		r8		s12	s13	s14	s15	r8				r8				
23		r9		s12	s13	s14	s15	r9				r9				
24				s12	s13	s14	s15					r4				

Acceptance of input e ^ f

Step	Stack	Input	Action
1	0	e^f$	s4
2	0e4	^f$	s16
3	0e4^16	f$	s6
4	0e4^16f6	$	r10
5	0e4^16E	$	24
6	0e4^16E24	$	r4
7	0S	$	1
8	0S1	$	acc

Acceptance of input f * f

Step	Stack	Input	Action
1	0	f*f$	S6
2	0f6	*f$	r10
3	0E	*f$	3
4	0E3	*f$	s14
5	0E3*14	f$	s6
6	0E3*14f6	$	r10
7	0E3*14E	$	22
8	0E3*14E22	$	r8
9	0E	$	3
10	0E3	$	r3
11	0S	$	1
12	0S1	$	acc

<u>Code (.lex and .yacc files)</u>

.lex file

```
%{  #include<stdio.h>
    #include "y.tab.h"
    #include<stdlib.h>
    void yyerror(char *);
%}
%%

sin return SIN;
cos return COS;
tan return TAN;
"^" return POW;
log  return LOG;
"("  return LPAR;
")"   return RPAR;
"e" return E;
"sqrt" return SQRT;
[0-9]+\.*[0-9]* { yylval.du = atof(yytext); return NUMBER; }
[-/+*\n] return *yytext;
. yyerror("Incorrect symbol!");

%%
int yywrap(void)
{
  return 0;
}
```

.yacc file

```
%{ #include<stdio.h>
  #include<math.h>
  #include<stdlib.h>
  int yylex(void);
  void yyerror(char *);
%}

//YYLVAL will take all values as double
%union {
    double du;
}
```

```
%token <du> SIN COS TAN POW LOG LPAR RPAR E SQRT NUMBER
%type <du> expr

//precedence
%left '+' '-'
%left '*' '/'
%%

//grammar
// log(expr1)(expr2) -> log(base)(a)
program : LOG LPAR expr RPAR LPAR expr RPAR {
            if($3==1)
            {
              printf("Log cannot be calculated for base = 1\n");

            }
            printf("\n    (     %f    )    (    %f    )\n",$3,$6);
              double val=log($6)/log($3);

              printf("\nlog (      expr       )    (       expr       )\n\n");
              printf("\nlog (     %f    )    (    %f    ) = %f \n\n",$3,$6,log($6)/log($3));

              printf("%f",val);
              return 0;
            } |

//(expr1)^(expr2)
 LPAR expr RPAR POW LPAR expr RPAR {
     printf("\n    (     %f    ) ^ (    %f    )\n",$2,$6);

     printf("\n    (      expr       ) ^ (       expr       )\n\n");
     double val=pow($2,$6);
     printf("\n    (     %f    )  ^  (    %f    ) = %f \n\n",$2,$6,val);
     return 0; } |

//sin(expr)
 SIN LPAR expr RPAR { printf("\nsin (    %f    )",$3);
                    printf("\nsin (       expr       )");
                    printf("\nsin(%f) = %f",$3,sin(($3*3.14)/180));
                    return 0; } |

 //cos(expr)
 COS LPAR expr RPAR { printf("\ncos (    %f    )",$3);
                    printf("\ncos (       expr       )");
```

```
                printf("\ncos(%f) = %f",$3,cos(($3*3.14)/180));
                return 0; } |

 //tan(expr)
 TAN LPAR expr RPAR { printf("\ntan (    %f    )",$3);
                printf("\ntan (       expr       )");
                printf("\ntan(%f) = %f",$3,tan(($3*3.14)/180));
                return 0; } |
 //e^(expr)
 E POW LPAR expr RPAR {
                  printf("\n e^(    %f    )",$4);
                  printf("\n e^(       expr       )");
                  /* printf("\ne^(%f) = %f",$3,pow(exp,$4)); */
                  return 0;
                } |
//sqrt(expr)
 SQRT LPAR expr RPAR {
                  double val;
                  printf("\n sqrt(    %f    )",$3);
                  printf("\n sqrt(       expr     )");
                  val=pow($3,0.5);
                  printf("\nsqrt(%f) = %f",$3,val);
                  return 0;
                } |

//expr
  expr  { printf("%f",$1); return 0; } //calculates normal expressions
;

expr:
NUMBER {$$ = $1; } |
E {$$ = M_E; } |
expr '+' expr {printf("     %f+%f ",$1,$3); $$ = $1 + $3;}; |
expr '-' expr {printf("     %f-%f ",$1,$3); $$ = $1 - $3;} |
expr '/' expr {printf("     %f/%f ",$1,$3); $$ = $1 / $3;} |
expr '*' expr {printf("     %f*%f ",$1,$3); $$ = $1 * $3;}
;

%%
void yyerror(char *s) {
  fprintf(stderr,"%s\n",s);
}

int main() {
```

```
  yyparse();
  return 0;
}
```

Output screenshots

1. (7-3)^(2*2)

```
(base) ishita@Ishitas-MacBook-Air-2 desktop % ./a.out
(7-3)(2*2)
256.000000%
```

2. log(4/2)(8*2-4)

```
(base) ishita@Ishitas-MacBook-Air-2 desktop % ./a.out
log(4/2)(8*2-4)
3.584963%
```

3. sin(15*2)

```
(base) ishita@Ishitas-MacBook-Air-2 desktop % ./a.out
sin(15*2)
0.499770%
```

4. cos(90/2-15)

```
(base) ishita@Ishitas-MacBook-Air-2 desktop % ./a.out
cos(90/2-15)
0.866158%
```

5. tan(5*5+10/2)

```
(base) ishita@Ishitas-MacBook-Air-2 desktop % ./a.out
tan(5*5+10/2)
0.576996%
```

6. 7+3-2*8+1

```
(base) ishita@Ishitas-MacBook-Air-2 desktop % ./a.out
7+3-2*8+1
-5.000000
```

7. e^(2*3-6+5/3)

```
(base) ishita@Ishitas-MacBook-Air-2 desktop % ./a.out
e^(2*3-6+5/3)
5.294490
```

8. Parse tree for (e)^(2*3)

```
(base) ishita@Ishitas-MacBook-Air-2 desktop % ./a.out
(e)^(2*3)
     2.000000*3.000000
    (     2.718282    ) ^ (    6.000000    )

    (      expr       ) ^ (       expr       )


    (     2.718282    )  ^  (    6.000000    ) = 403.428793
```

9. Parse tree for log(2+3)(3*3)

```
(base) ishita@Ishitas-MacBook-Air-2 desktop % ./a.out
log(2+3)(3*3)
     2.000000+3.000000      3.000000*3.000000
    (     5.000000    )    (    9.000000    )

log (      expr       )    (       expr       )


log (     5.000000    )    (    9.000000    ) = 1.365212
```

--- END ---