Project Report
on
Compiler for Layman Friendly Calculator

Developed by
Jay Shah- IT001 - 18ITUON028
Ronak Agnani- IT004 - 18ITUON049
Arrshi Kandroo- IT007 - 18ITUOS073
Drashti Bhingradiya- IT017 - 18ITUOS047

Department of Information Technology
Faculty of Technology,
Dharmsinh Desai University
College Road, Nadiad-387001
2020-2021

DHARMSINH DESAI UNIVERSITY
NADIAD-387001, GUJARAT

CERTIFICATE

This is to certify that the project entitled **" LAYMAN FRIENDLY CALCULATOR "** is a bonafide report of the work carried out by

1) Mr. Jay Shah , Student ID No: 18ITUON028
2) Mr. Ronak Agnani , Student ID No: 18ITUONO49
3) Miss. Arrshi Kandroo , Student ID No: 18ITUOS073
4) Miss. Drashti Bhingradiya , Student ID No: 18ITUOS047

of Department of Information Technology, semester VI, under the guidance and supervision for the award of the degree of Bachelor of Technology at Dharmsinh Desai University, Nadiad (Gujarat). They were involved in Project in subject of "**Language Translator**" during the academic year 2020-2021.

Prof. N.P. Desai
(Lab Incharge)
Department of Information Technology,
Faculty of Technology,
Dharmsinh Desai University, Nadiad
Date:

Prof. (Dr.)V K Dabhi,
Head , Department of Information Technology,
Faculty of Technology,
Dharmsinh Desai University, Nadiad
Date:

INDEX

Chapter 1:
INTRODUCTION

1.1 PROJECT DETAILS

Our project is focused on making a language descriptor and compiler for layman friendly Calculator where a user can enter the english sentences for all calculations.

For example,
Valid program in language :

ADD 23 WITH 444.45 . ALSO ADD 4543.78 TO 123 . NOW DO ADDITION OF BOTH ANSWERS . WHAT IS RESULT?

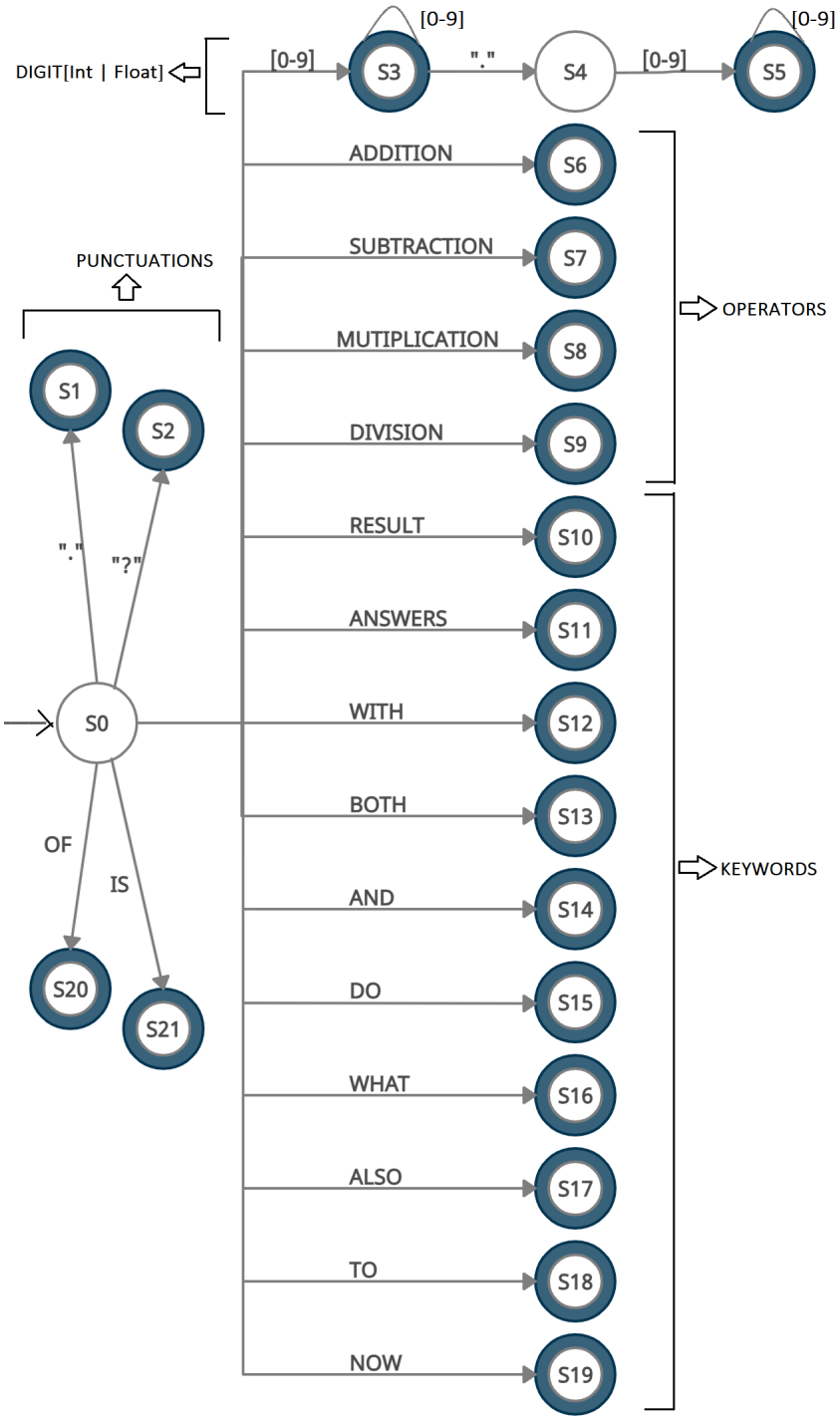
The above language is to be processed by the compiler designed by us and calculate appropriate result from it.

1.2 PROJECT PLANNING

Students involved	**Roles and Responsibilities**
IT004-Ronak Agnani	Regular Expression and DFA for lexer
IT007-Arrshi Kandroo	Algorithm from RE for lexer
IT001-Jay Shah	Implementation of lexer
IT017-Drashti Bhingradiya	Yacc based implementation of syntax analyzer

Chapter 2:
LEXICAL PHASE DESIGN

2.1 DETERMINISTIC FINITE AUTOMATA DESIGN FOR LEXER



2.2 ALGORITHM FOR LEXER

```
state:= s0 input(ch);
while not eof do
        case state of
                s0: case ch of
                        ‘?’: state:= s1;
                        '.': state:= s2;
                        'digit': state:= s3; input(ch);
                        'ADDITION’: state:= s6;
                        'SUBTRACTION’: state:= s7;
                        'MULTIPLICATION’: state:= s8;
                        ‘DIVISION’: state:= s8;
                        'RESULT’: state:= s10;
                        ‘ANSWERS’: state:= s11;
                        ‘WITH’: state:= s12;
                        ‘BOTH’: state:= s13;
                        ‘AND’: state:= s14;
                        ‘DO’: state:= s15;
                        ‘WHAT’: state:= s16;
                        ‘ALSO’: state:= s17;
                        ‘TO’: state:= s18;
                        ‘NOW’: state:= s19;
                        ‘IS’: state:= s20;
                        ‘OF’: state:= s21;

                        else exit while;
                end case;

                s3: case ch of
                        '.': state:= s4; input(ch);
                        'digit': state:= s3; input(ch);
                        else exit while;
                end case;

                s4: case ch of
                        'digit': state:= s5; input(ch);
                        else exit while;
                end case;
```

```
            s5: case ch of
                    'digit': state:= s5; input(ch);
                    else exit while;
                end case;

        end case;
exit while;

if (state:=s1 || state:=s2 || state:= s3 || state:=s5 || state:= s6 || state:= s7 || state:= s8 ||
state:= s9 || state:= s10 || state:= s11 || state:= s12 || state:=s13 || state:= s14 || state:=
15 || state:= 16 || state:= s17 || state:= s18 || state:= s19 || state:= s20 || state:= s21 )
        then accept_state;
else error;
end if;

end Algorithm;
```

2.2.1 IMPLEMENTATION OF ALGORITHM IN JAVA LANGUAGE

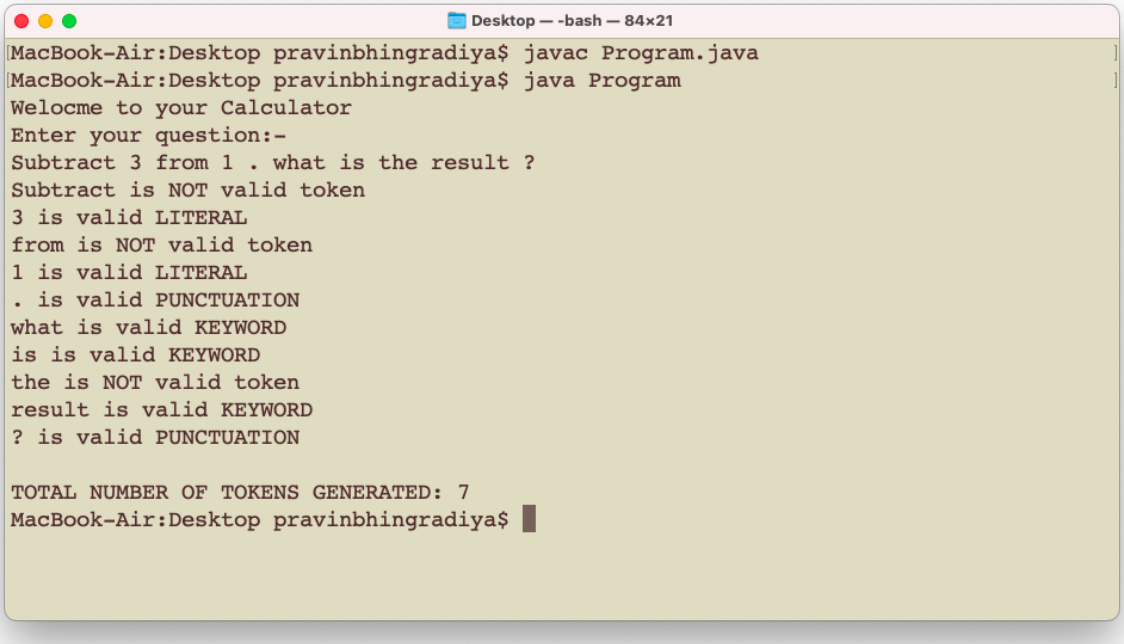
JAVA CODE:

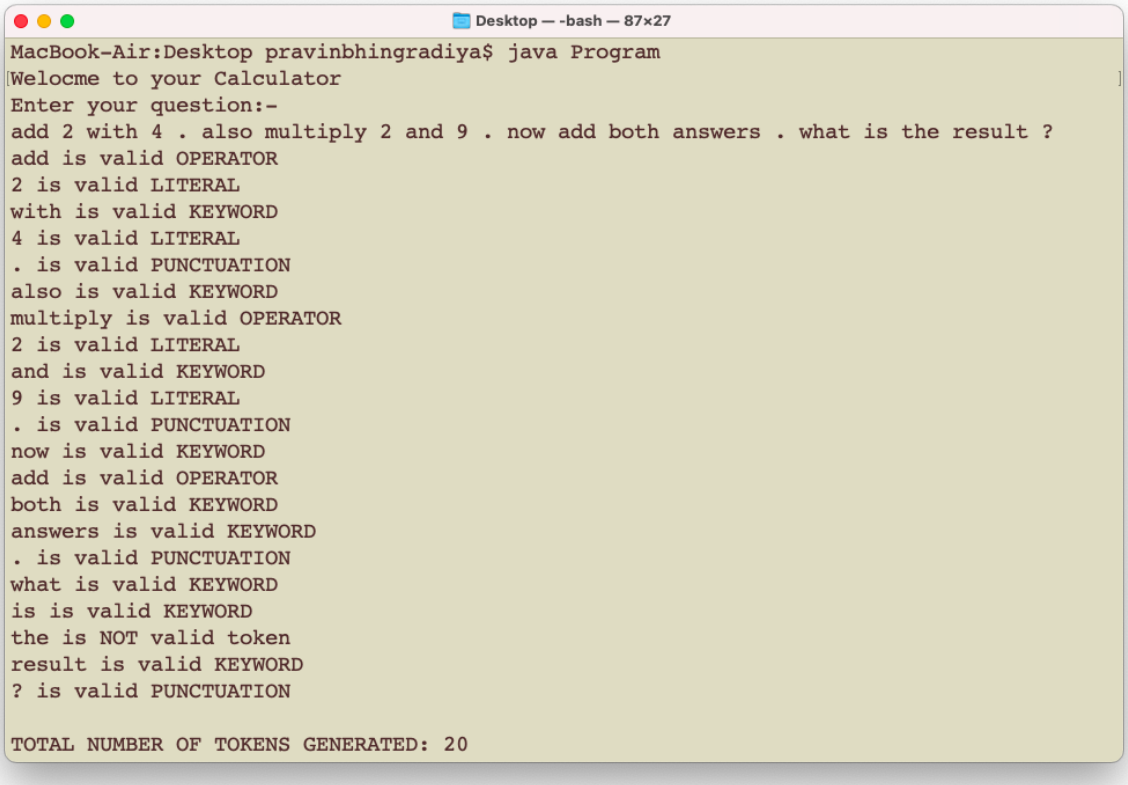
```
/*
Keywords: to, with, and, now, of, both, answers, do, also, what, is, result
operator: add, sub, multiply, divide, addition, subtraction, multiplication, division
intc (integer literal):  (digit)+
fpl (fixed point literal):  (digit)*“.”(digit)+
pun (punctuation):  ‘.’ , ‘?’
del (delimiter): [\n\t]
*/

import java.util.Scanner;
public class Program
{
	public static void main(String[] args)
	{
		System.out.println("Welocme to your Calculator");
		System.out.println("Enter your question:-");
		Scanner sc = new Scanner(System.in);
		String user_in = sc.nextLine();
		String token[] = user_in.split("\\s+");

		String validKeyword = " is valid KEYWORD";
		String validOperator = " is valid OPERATOR";
		String validLiteral = " is valid LITERAL";
		String validPunctuation = " is valid PUNCTUATION";
		String validDelimiter = " is valid DELIMITER";
		int count=0;

		for(String s: token)
		{
			if(s.equalsIgnoreCase("to") || s.equalsIgnoreCase("with") ||
				s.equalsIgnoreCase("now") || s.equalsIgnoreCase("both") ||
				s.equalsIgnoreCase("of") ||s.equalsIgnoreCase("and") ||
				s.equalsIgnoreCase("answers") || s.equalsIgnoreCase("do")
||
				s.equalsIgnoreCase("also") || s.equalsIgnoreCase("what") ||
				s.equalsIgnoreCase("is") || s.equalsIgnoreCase("result") )
			{
```

```
                        count++;
                        System.out.println(s + validKeyword);
                }
                else if(s.equalsIgnoreCase("add") || s.equalsIgnoreCase("addition")
||
                        s.equalsIgnoreCase("sub") ||
s.equalsIgnoreCase("subtraction") ||
                        s.equalsIgnoreCase("multiply") ||
s.equalsIgnoreCase("multiplication") ||
                        s.equalsIgnoreCase("divide") ||
s.equalsIgnoreCase("division") )
                {
                        count++;
                        System.out.println(s + validOperator);
                }
                else if(s.equalsIgnoreCase(".") || s.equalsIgnoreCase("?") )
                {
                        count++;
                        System.out.println(s + validPunctuation);
                }
                else if(s.equals(" ") || s.equalsIgnoreCase("\n") ||
                        s.equalsIgnoreCase("\t") )
                {
                        count++;
                        System.out.println(s + validDelimiter);
                }
                else if( validateInteger(s) )
                {
                        count++;
                        System.out.println(s + validLiteral);
                }
                else if( validateFloat(s))
                {
                        count++;
                        System.out.println(s + validLiteral);

                }
                else
                {
                        System.out.println(s + " is NOT valid token");
```

```
                }
            }

            System.out.println("\nTOTAL NUMBER OF TOKENS GENERATED: "
+count);

        }


            public static boolean validateInteger(String input)
            {
                try
                {
                    Integer.parseInt(input);
                    return true;
                }
                catch( NumberFormatException e )
                {
                    return false;
                }
            }

            public static boolean validateFloat( String input )
            {
                try
                {
                    Float.parseFloat( input );
                    return true;
                }
                catch( NumberFormatException e )
                {
                    return false;
                }
            }

}
```

OUTPUT:


Desktop — -bash — 84×21
MacBook-Air:Desktop pravinbhingradiya$ javac Program.java
MacBook-Air:Desktop pravinbhingradiya$ java Program
Welocme to your Calculator
Enter your question:-
Subtract 3 from 1 . what is the result ?
Subtract is NOT valid token
3 is valid LITERAL
from is NOT valid token
1 is valid LITERAL
. is valid PUNCTUATION
what is valid KEYWORD
is is valid KEYWORD
the is NOT valid token
result is valid KEYWORD
? is valid PUNCTUATION
TOTAL NUMBER OF TOKENS GENERATED: 7
MacBook-Air:Desktop pravinbhingradiya$


Desktop — -bash — 87×27
MacBook-Air:Desktop pravinbhingradiya$ java Program
Welocme to your Calculator
Enter your question:-
add 2 with 4 . also multiply 2 and 9 . now add both answers . what is the result ?
add is valid OPERATOR
2 is valid LITERAL
with is valid KEYWORD
4 is valid LITERAL
. is valid PUNCTUATION
also is valid KEYWORD
multiply is valid OPERATOR
2 is valid LITERAL
and is valid KEYWORD
9 is valid LITERAL
. is valid PUNCTUATION
now is valid KEYWORD
add is valid OPERATOR
both is valid KEYWORD
answers is valid KEYWORD
. is valid PUNCTUATION
what is valid KEYWORD
is is valid KEYWORD
the is NOT valid token
result is valid KEYWORD
? is valid PUNCTUATION
TOTAL NUMBER OF TOKENS GENERATED: 20

2.3 IMPLEMENTATION OF LEXER

```
%{
#include<stdio.h>
int yyerror(char *errormsg);
%}

digit [0-9]
keywords
WITH|BOTH|AND|DO|WHAT|RESULT|ALSO|TO|NOW|OF|BOTH|ANSWERS|IS
whitespace \t|\b|\n
punctuations "."|"?"
opadd "ADD"|"ADDITION"
opsub "SUBTRACT"|"SUBTRACTION"
opmul "MULTIPLY"|"MULTIPLICATION"|"TIMES"
opdiv "DIVIDE"|"DIVISION"
%%
{keywords}          {printf("%s is valid KEYWORD\n",yytext);}

{digit}+            {printf("%s is valid Literal\n",yytext);}

{digit}*"."{digit}+ {printf("%s is valid Literal\n",yytext);}

{punctuations}      {printf("%s is valid Punctuation\n",yytext);}
[ \t\n]+

{opadd}             {printf("%s is valid + OPERATOR\n",yytext);}

{opsub}             {printf("%s is valid - OPERATOR\n",yytext);}

{opmul}             {printf("%s is valid * OPERATOR\n",yytext);}

{opdiv}             {printf("%s is valid / OPERATOR\n",yytext);}

.                   {printf("%s is NOT a valid Token\n",yytext);}
%%
int yywrap(void)
{
        return 1;
}
```

```
int main(void)
{
yylex();
return 0;
}
int yyerror(char *errormsg)
{
printf("\n======ERROR GENERATED:======\n");
fprintf(stderr, "line %d: %s\n:", yylineno, errormsg);
exit(1);
}
```

2.4 EXECUTION ENVIRONMENT SETUP

If you are using Ubuntu (or any Linux based OS) then your system will have inbuilt software for lex and bison.
For Windows OS you can easily run your programs in any hassles by following these steps:
Installing Softwares:

1. Download Flex 2.5.4a
2. Download Bison 2.4.1
3. Download DevC++
4. Install Flex at "C:\GnuWin32"
5. Install Bison at "C:\GnuWin32"
6. Install DevC++ at "C:\Dev-Cpp"
7. Open Environment Variables.
8. Add "C:\GnuWin32\bin; C:\Dev-Cpp\bin;" to the path.

Compilation & Execution of your Program:

1. Open Command prompt and switch to your working directory where you have stored your lex file (".l")
2. Let your lex file be "hello.l". Now, follow the preceding steps to compile and run your program.
    1. For Compiling Lex file only: (LINUX)
        1. lex hello.l
        2. make lex.yy.c
    2. For Executing the Program
        1. ./lex.yy

2.5 OUTPUT SCREENSHOTS OF LEXER

Test Case 1-

```
(base) MacBook-Air:lt_lab pravinbhingradiya$ ./lex.yy
299 PLUS 35 MINUS 89 DIVIDE 17 . WHAT IS RESULT ?
299 is valid Literal
P is NOT a valid Token
L is NOT a valid Token
U is NOT a valid Token
S is NOT a valid Token
35 is valid Literal
M is NOT a valid Token
I is NOT a valid Token
N is NOT a valid Token
U is NOT a valid Token
S is NOT a valid Token
89 is valid Literal
DIVIDE is valid / OPERATOR
17 is valid Literal
. is valid Punctuation
WHAT is valid KEYWORD
IS is valid KEYWORD
RESULT is valid KEYWORD
? is valid Punctuation
```

Test Case 2-

```
(base) MacBook-Air:lt_lab pravinbhingradiya$ lex project.l
(base) MacBook-Air:lt_lab pravinbhingradiya$ make lex.yy
cc     lex.yy.c   -o lex.yy
(base) MacBook-Air:lt_lab pravinbhingradiya$ ./lex.yy
MULTIPLY 2.3 AND 33.4 . ALSO DIVIDE 233.4 AND 47 . NOW DO ADDITION OF BOTH ANSWERS . WHAT IS RESULT ?
MULTIPLY is valid * OPERATOR
2.3 is valid Literal
AND is valid KEYWORD
33.4 is valid Literal
. is valid Punctuation
ALSO is valid KEYWORD
DIVIDE is valid / OPERATOR
233.4 is valid Literal
AND is valid KEYWORD
47 is valid Literal
. is valid Punctuation
NOW is valid KEYWORD
DO is valid KEYWORD
ADDITION is valid + OPERATOR
OF is valid KEYWORD
BOTH is valid KEYWORD
ANSWERS is valid KEYWORD
. is valid Punctuation
WHAT is valid KEYWORD
IS is valid KEYWORD
RESULT is valid KEYWORD
? is valid Punctuation
^C
(base) MacBook-Air:lt_lab pravinbhingradiya$
```

Chapter 3:
SYNTAX ANALYZER DESIGN

3.1 GRAMMAR RULES

Regular definition-

keywords
WITH|BOTH|AND|DO|WHAT|RESULT|ALSO|TO|NOW|OF|BOTH|ANSWERS|IS
whitespace \t|\b|\n
punctuations "."|"?"
opadd "ADD"|"ADDITION"
opsub "SUBTRACT"|"SUBTRACTION"
opmul "MULTIPLY"|"MULTIPLICATION"|"TIMES"
opdiv "DIVIDE"|"DIVISION"

Grammar-

S -> A B C D

A -> + INTEGER KEYWORD FLOATS PUNCTUATION
| - INTEGER KEYWORD FLOATS PUNCTUATION
| * INTEGER KEYWORD FLOATS PUNCTUATION
| / INTEGER KEYWORD FLOATS PUNCTUATION

B -> KEYWORD + FLOATS KEYWORD INTEGER PUNCTUATION
| KEYWORD - FLOATS KEYWORD INTEGER PUNCTUATION
| KEYWORD * FLOATS KEYWORD INTEGER PUNCTUATION
| KEYWORD / FLOATS KEYWORD INTEGER PUNCTUATION

C -> KEYWORD KEYWORD + KEYWORD KEYWORD KEYWORD PUNCTUATION
| KEYWORD KEYWORD - KEYWORD KEYWORD KEYWORD PUNCTUATION
| KEYWORD KEYWORD * KEYWORD KEYWORD KEYWORD PUNCTUATION
| KEYWORD KEYWORD / KEYWORD KEYWORD KEYWORD PUNCTUATION

D -> KEYWORD KEYWORD KEYWORD PUNCTUATION

Sample Input: ADD 23 WITH 444.45 . ALSO ADD 4543.78 TO 123 . NOW DO ADDITION OF BOTH ANSWERS . WHAT IS RESULT?

Expected Output - Result is 5134.23

3.2 YACC BASED IMPLEMENTATION OF SYNTAX ANALYZER

Total Files:

1. lex file (project.l)
2. Yacc file (project.y)
3. y.tab.h (header file) (Auto-generated)

CODE:

1. project.l

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

keywords
WITH|BOTH|AND|DO|WHAT|RESULT|ALSO|TO|NOW|OF|BOTH|ANSWERS|IS
whitespace \t|\b|\n
punctuations "."|"?"
opadd "ADD"|"ADDITION"
opsub "SUBTRACT"|"SUBTRACTION"
opmul "MULTIPLY"|"MULTIPLICATION"|"TIMES"
opdiv "DIVIDE"|"DIVISION"

%%
{keywords}          {printf("%s is valid KEYWORD\n",yytext); return KEYWORD;}

(([0-9]+)|([0-9]*\.[0-9]+))
{ printf("%s is valid Literal\n",yytext); yylval.number=(float)atof(yytext); return NUM;}

{punctuations}      {printf("%s is valid Punctuation\n",yytext); return PUNCTUATION;}
[ \t\n]+

{opadd}             {printf("%s is valid + OPERATOR\n",yytext); return '+';}

{opsub}             {printf("%s is valid - OPERATOR\n",yytext); return '-';}

{opmul}             {printf("%s is valid * OPERATOR\n",yytext); return '*';}

{opdiv}             {printf("%s is valid / OPERATOR\n",yytext); return '/';}
```

```
.                       {printf("%s is NOT a valid Token\n",yytext);}
%%

int yywrap(void)
{
        return 1;
}

int yyerror(char *errormsg)
{
printf("\n======ERROR GENERATED:======\n");
fprintf(stderr, "line %d: %s\n:", yylineno, errormsg);
exit(1);
}
```

2. project.y

```
%{
#include <stdio.h>
#include <stdlib.h>
int yylex(void);
int yyerror(const char *s);
int flag=0;
float tmp[3];
float ans=0.0;
%}
%union
{
float number;
}

%token <number> NUM FLOAT
%type <number> A B C

%token KEYWORD PUNCTUATION OPERATOR

%%
S : A B C D { printf("===Entered Query is CORRECTLY WRITTEN!===\n");
                        if(flag==0){printf ("RESULT = %.3f\n",ans);} return 0;}
```

```
        ;

A : '+' NUM KEYWORD NUM PUNCTUATION          { $$ = $2 + $4;
                tmp[0]=$$;
                printf("ENTERED FIRST EXPRESSION IS: %f + %f = %f
\n",$2,$4,tmp[0]);
        }
        | '-' NUM KEYWORD NUM PUNCTUATION     { $$ = $2 - $4;
                tmp[0]=$$;
                printf("ENTERED FIRST EXPRESSION IS: %f - %f = %f\n",$2,$4,tmp[0]);
        }
        | '*' NUM KEYWORD NUM PUNCTUATION     { $$ = $2 * $4;
                tmp[0]=$$;
                printf("ENTERED FIRST EXPRESSION IS: %f * %f = %f\n",$2,$4,tmp[0]);
        }
        | '/' NUM KEYWORD NUM PUNCTUATION     {
        if($4==0){printf("===ERROR: DIVIDE BY ZERO \n CANNOT CALCULATE
RESULT!==\n"); flag=1; }
                else{
                $$=$2/$4;
                tmp[0]=$$;
                printf("ENTERED FIRST EXPRESSION IS: %f /%f = %f\n",$2,$4,tmp[0]); }
        }
        ;

B : KEYWORD '+' NUM KEYWORD NUM PUNCTUATION            { $$ = $3 + $5;
                tmp[1]=$$;
                printf("ENTERED SECOND EXPRESSION IS: %f + %f =
%f\n",$3,$5,tmp[1]);
        }
        | KEYWORD '-' NUM KEYWORD NUM PUNCTUATION  { $$ = $3 - $5;
                tmp[1]=$$;
                printf("ENTERED SECOND EXPRESSION IS: %f - %f =
%f\n",$3,$5,tmp[1]);
        }
        | KEYWORD '*' NUM KEYWORD NUM PUNCTUATION   { $$ = $3 * $5;
                tmp[1]=$$;
                printf("ENTERED SECOND EXPRESSION IS: %f * %f =
%f\n",$3,$5,tmp[1]);
        }
```

```
        | KEYWORD '/' NUM KEYWORD NUM PUNCTUATION { if($5==0){printf("===ERROR: DIVIDE BY ZERO \n CANNOT CALCULATE RESULT!==\n"); flag=1; }
        else{
        $$=$3/$5;
        tmp[1]=$$;
        printf("ENTERED SECOND EXPRESSION IS: %f / %f = %f\n",$3,$5,tmp[1]); }
    }
    ;

C : KEYWORD KEYWORD '+' KEYWORD KEYWORD KEYWORD PUNCTUATION {
        ans = tmp[0] + tmp[1];
        printf("THIRD OPERATION PROVIDED IS + \n");
    }
    | KEYWORD KEYWORD '-' KEYWORD KEYWORD KEYWORD PUNCTUATION
    {
        ans = tmp[0] - tmp[1];
        printf("THIRD OPERATION PROVIDED IS - \n");
    }
    | KEYWORD KEYWORD '*' KEYWORD KEYWORD KEYWORD PUNCTUATION
    {
        ans = tmp[0] * tmp[1];
        printf("THIRD OPERATION PROVIDED IS * \n");
    }
    | KEYWORD KEYWORD '/' KEYWORD KEYWORD KEYWORD PUNCTUATION
    {
        if(tmp[1]==0){printf("===ERROR: DIVIDE BY ZERO \n CANNOT CALCULATE RESULT!==\n"); flag=1; }
        else{
        ans = tmp[0]/tmp[1];
        printf("THIRD OPERATION PROVIDED IS / /n"); }
    }
    ;

D : KEYWORD KEYWORD KEYWORD PUNCTUATION
    ;
%%

int main()
```

```
{
printf("Enter Your Query : \n");
yyparse();
printf("\n SUCCESSFULLY EXECTUED PROGRAM! \N PROGRAM TERMINATING…
\n");
return 0;
}
```

3.3 EXECUTION ENVIRONMENT SETUP

Assuming that we already have all the required softwares to run the yacc and lex programs. If not, the environment setup given in 2.4 is to be followed first.
Steps to run and execute programs:
Compilation & Execution of your Program:

1. Open Command prompt and switch to your working directory where you have stored your lex file (".l") and yacc file (".y")

2. Let your lex and yacc files be "project.l" and "project.y". Now, follow the preceding steps to compile and run your program.

    1. For Compiling Lex & Yacc file both: (LINUX)
        1. yacc -d project.y
        2. lex project.l
        3. cc lex.yy.c y.tab.c -o project.out

    2. For Executing the Program
        1. ./project.out

3.4 OUTPUT SCREENSHOTS OF YACC BASED IMPLEMENTATION

Sample Output:

```
(base) MacBook-Air:project pravinbhingradiya$ yacc project.y
(base) MacBook-Air:project pravinbhingradiya$ lex project.l
(base) MacBook-Air:project pravinbhingradiya$ cc lex.yy.c y.tab.c -o project.out
(base) MacBook-Air:project pravinbhingradiya$ ./project.out
Enter Your Query :
ADD 23 WITH 444.45 . ALSO ADD 4543.78 TO 123 . NOW DO ADDITION OF BOTH ANSWERS . WHAT IS RESULT ?
ADD is valid + OPERATOR
23 is valid Literal
WITH is valid KEYWORD
444.45 is valid Literal
. is valid Punctuation
ENTERED FIRST EXPRESSION IS: 23.000000 + 444.450012 = 467.450012
ALSO is valid KEYWORD
ADD is valid + OPERATOR
4543.78 is valid Literal
TO is valid KEYWORD
123 is valid Literal
. is valid Punctuation
ENTERED SECOND EXPRESSION IS: 4543.779785 + 123.000000 = 4666.779785
NOW is valid KEYWORD
DO is valid KEYWORD
ADDITION is valid + OPERATOR
OF is valid KEYWORD
BOTH is valid KEYWORD
ANSWERS is valid KEYWORD
. is valid Punctuation
THIRD OPERATION PROVIDED IS +
WHAT is valid KEYWORD
IS is valid KEYWORD
RESULT is valid KEYWORD
? is valid Punctuation
===Entered Query is CORRECTLY WRITTEN!===
RESULT = 5134.230

SUCCESSFULLY EXECTUED PROGRAM!
 PROGRAM TERMINATING...
(base) MacBook-Air:project pravinbhingradiya$
```

output from yacc file

Test case 1-
SUBTRACT 32.3 AND 3 . ALSO MULTIPLY 12 TO 4 . NOW DO ADDITION OF BOTH ANSWERS . WHAT IS RESULT ?

```
(base) MacBook-Air:project pravinbhingradiya$ ./project.out
Enter Your Query :
SUBTRACT 32.3 AND 3 . ALSO MULTIPLY 12 TO 4 . NOW DO ADDITION OF BOTH ANSWERS . WHAT IS RESULT ?
SUBTRACT is valid - OPERATOR
32.3 is valid Literal
AND is valid KEYWORD
3 is valid Literal
. is valid Punctuation
ENTERED FIRST EXPRESSION IS: 32.299999 - 3.000000 = 29.299999
ALSO is valid KEYWORD
MULTIPLY is valid * OPERATOR
12 is valid Literal
TO is valid KEYWORD
4 is valid Literal
. is valid Punctuation
ENTERED SECOND EXPRESSION IS: 12.000000 * 4.000000 = 48.000000
NOW is valid KEYWORD
DO is valid KEYWORD
ADDITION is valid + OPERATOR
OF is valid KEYWORD
BOTH is valid KEYWORD
ANSWERS is valid KEYWORD
. is valid Punctuation
THIRD OPERATION PROVIDED IS +
WHAT is valid KEYWORD
IS is valid KEYWORD
RESULT is valid KEYWORD
? is valid Punctuation
===Entered Query is CORRECTLY WRITTEN!===
RESULT = 77.300

SUCCESSFULLY EXECTUED PROGRAM!
 PROGRAM TERMINATING...
(base) MacBook-Air:project pravinbhingradiya$
```

Test case 2-

MULTIPLY 2.3 AND 33.4 . ALSO DIVIDE 233.4 AND 47 . NOW DO ADDITION OF BOTH ANSWERS . WHAT IS RESULT ?

```
(base) MacBook-Air:project pravinbhingradiya$ ./project.out
Enter Your Query :
MULTIPLY 2.3 AND 33.4 . ALSO DIVIDE 233.4 AND 47 . NOW DO ADDITION OF BOTH ANSWERS . WHAT IS RESULT ?
MULTIPLY is valid * OPERATOR
2.3 is valid Literal
AND is valid KEYWORD
33.4 is valid Literal
. is valid Punctuation
ENTERED FIRST EXPRESSION IS: 2.300000 * 33.400002 = 76.820000
ALSO is valid KEYWORD
DIVIDE is valid / OPERATOR
233.4 is valid Literal
AND is valid KEYWORD
47 is valid Literal
. is valid Punctuation
ENTERED SECOND EXPRESSION IS: 233.399994 / 47.000000 = 4.965957
NOW is valid KEYWORD
DO is valid KEYWORD
ADDITION is valid + OPERATOR
OF is valid KEYWORD
BOTH is valid KEYWORD
ANSWERS is valid KEYWORD
. is valid Punctuation
THIRD OPERATION PROVIDED IS +
WHAT is valid KEYWORD
IS is valid KEYWORD
RESULT is valid KEYWORD
? is valid Punctuation
===Entered Query is CORRECTLY WRITTEN!===
RESULT = 81.786

SUCCESSFULLY EXECTUED PROGRAM!
 PROGRAM TERMINATING...
(base) MacBook-Air:project pravinbhingradiya$
```

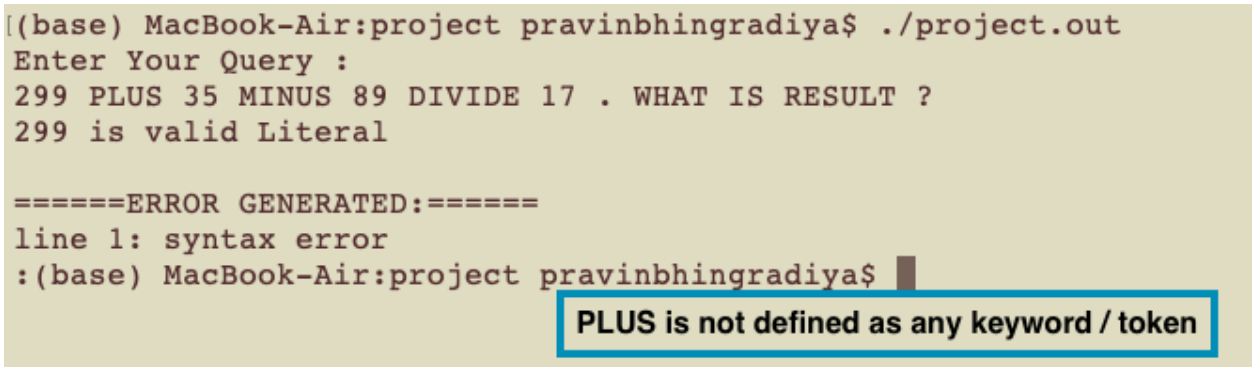
ERROR HANDLING:

1. Divide by zero:

```
(base) MacBook-Air:project pravinbhingradiya$ yacc project.y
(base) MacBook-Air:project pravinbhingradiya$ lex project.l
(base) MacBook-Air:project pravinbhingradiya$ cc lex.yy.c y.tab.c -o project.out
(base) MacBook-Air:project pravinbhingradiya$ ./project.out
Enter Your Query :
DIVIDE 2 AND 0 . ALSO MULTIPLY 233.4 TO 47 . NOW DO ADDITION OF BOTH ANSWERS . WHAT IS RESULT ?
DIVIDE is valid / OPERATOR
2 is valid Literal
AND is valid KEYWORD
0 is valid Literal
. is valid Punctuation
===ERROR: DIVIDE BY ZERO
 CANNOT CALCULATE RESULT!==  2/0
ALSO is valid KEYWORD
MULTIPLY is valid * OPERATOR
233.4 is valid Literal
TO is valid KEYWORD
47 is valid Literal
. is valid Punctuation
ENTERED SECOND EXPRESSION IS: 233.399994 * 47.000000 = 10969.799805
NOW is valid KEYWORD
DO is valid KEYWORD
ADDITION is valid + OPERATOR
OF is valid KEYWORD
BOTH is valid KEYWORD
ANSWERS is valid KEYWORD
. is valid Punctuation
THIRD OPERATION PROVIDED IS +
WHAT is valid KEYWORD
IS is valid KEYWORD
RESULT is valid KEYWORD
? is valid Punctuation
===Entered Query is CORRECTLY WRITTEN!===

SUCCESSFULLY EXECTUED PROGRAM!
 PROGRAM TERMINATING...
(base) MacBook-Air:project pravinbhingradiya$
```

```
(base) MacBook-Air:project pravinbhingradiya$ ./project.out
Enter Your Query :
SUBTRACT 2 AND 3.0 . ALSO MULTIPLY 1 TO 0 . NOW DO DIVISION OF BOTH ANSWERS . WHAT IS RESULT ?
SUBTRACT is valid - OPERATOR
2 is valid Literal
AND is valid KEYWORD
3.0 is valid Literal
. is valid Punctuation
ENTERED FIRST EXPRESSION IS: 2.000000 - 3.000000 = -1.000000
ALSO is valid KEYWORD
MULTIPLY is valid * OPERATOR
1 is valid Literal
TO is valid KEYWORD
0 is valid Literal
. is valid Punctuation
ENTERED SECOND EXPRESSION IS: 1.000000 * 0.000000 = 0.000000
NOW is valid KEYWORD
DO is valid KEYWORD
DIVISION is valid / OPERATOR
OF is valid KEYWORD
BOTH is valid KEYWORD
ANSWERS is valid KEYWORD
. is valid Punctuation
===ERROR: DIVIDE BY ZERO
 CANNOT CALCULATE RESULT!==  -1/0
WHAT is valid KEYWORD
IS is valid KEYWORD
RESULT is valid KEYWORD
? is valid Punctuation
===Entered Query is CORRECTLY WRITTEN!===

SUCCESSFULLY EXECTUED PROGRAM!
 PROGRAM TERMINATING...
(base) MacBook-Air:project pravinbhingradiya$
```

2. Syntax error:



Chapter 4:
CONCLUSION AND FUTURE WORK

Conclusion

We have learnt different phases of a compiler and successfully implemented lexical and syntax phases using lex and yacc. A layman can now write any basic mathematical expression in english and can now get desired output.

Future Work

Implementing other all phases of phases of compiler starting with semantic analysis, intermediate code generation and react till final code generation using symbol table management and having adequate error handling mechanisms.

Furthermore, We can expand our calculator to perform scientific calculations and other complex number equations.

Since this is a layman friendly calculator, we can also enhance this calculator in different languages such gujarati and hindi.

Adding more innovative technology, we can first perform a voice recognition machine learning algorithm which will take the input from the user in the form of speech and convert it to text which is then supplied to our compiler for further processing.