Project Report on Compiler for

Number Operation

Developed by

Smitkumar Rathod – IT133 – 19ITUBS129

Harsh Raval – IT134 – 20ITUOD072

Rushi Raval – IT135 – 20ITUOD008

Guided By:

Prof. Nikita P. Desai

Dept. of Information Technology

Department of Information Technology
Faculty of Technology, Dharmsinh Desai University
College Road, Nadiad-387001 202-2022
DHARMSINH DESAI UNIVERSITY
NADIAD-387001, GUJARAT

CERTIFICATE

This is to certify that the project entitled “**Number Operations”** is a bonafide report of the work carried out by

1) Smitkumar Rathod Student ID No: 19ITUBS129

2) Harsh Raval Student ID No: 20ITUOD072

3) Rushi Raval Student ID No: 20ITUOD008

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 a Project in the subject of “**Language Translator**” during the academic year 2021-2022.

Prof. N.P. Desai
(Lab In charge)
Department of Information Technology,
Faculty of Technology,
Dharmsinh Desai University, Nadiad

Date: 13/0

Prof. (Dr.)V K Dabhi,
Head , Department of Information Technology,

Faculty of Technology,
Dharmsinh Desai University, Nadiad
Date:

Index

1.0 Introduction

2.0 Lexical phase design

3.0 Syntax analyzer design

1.0 INTRODUCTION

1.0.1 Project Details

Language Name: Number Operations.

Language description:

Performs mathematical operations like nthFibonacci, Factorial, Prime, Even, Odd in english langauge.

Example of a valid program in this language is

- give factorial of 5?
- give palindrome of 123?
- is 5 prime?
- is 8 even?
- is 9 odd?
- give nth fibonacci of 5?

1.0.2 Project Planning

List of Students with their Roles/Responsibilities:

IT133 – Smitkumar Rathod– DFA and Yacc Implementation
IT134 – Harsh Raval – Algorithm and Syntax Analyzer
IT135 – Rushi Raval – Lex and CPP Scanner Implementation

2.0 LEXICAL PHASE DESIGN

2.0.1 Regular Expression:

KeyWords :

RE	Token
give	give
is	is

Operations :

RE	Token	Attribute
factorial	op	factorial
palindrome	op	palindrome
prime	op	prime
even	op	even
odd	op	odd
fibonacci		op
fibonacci		

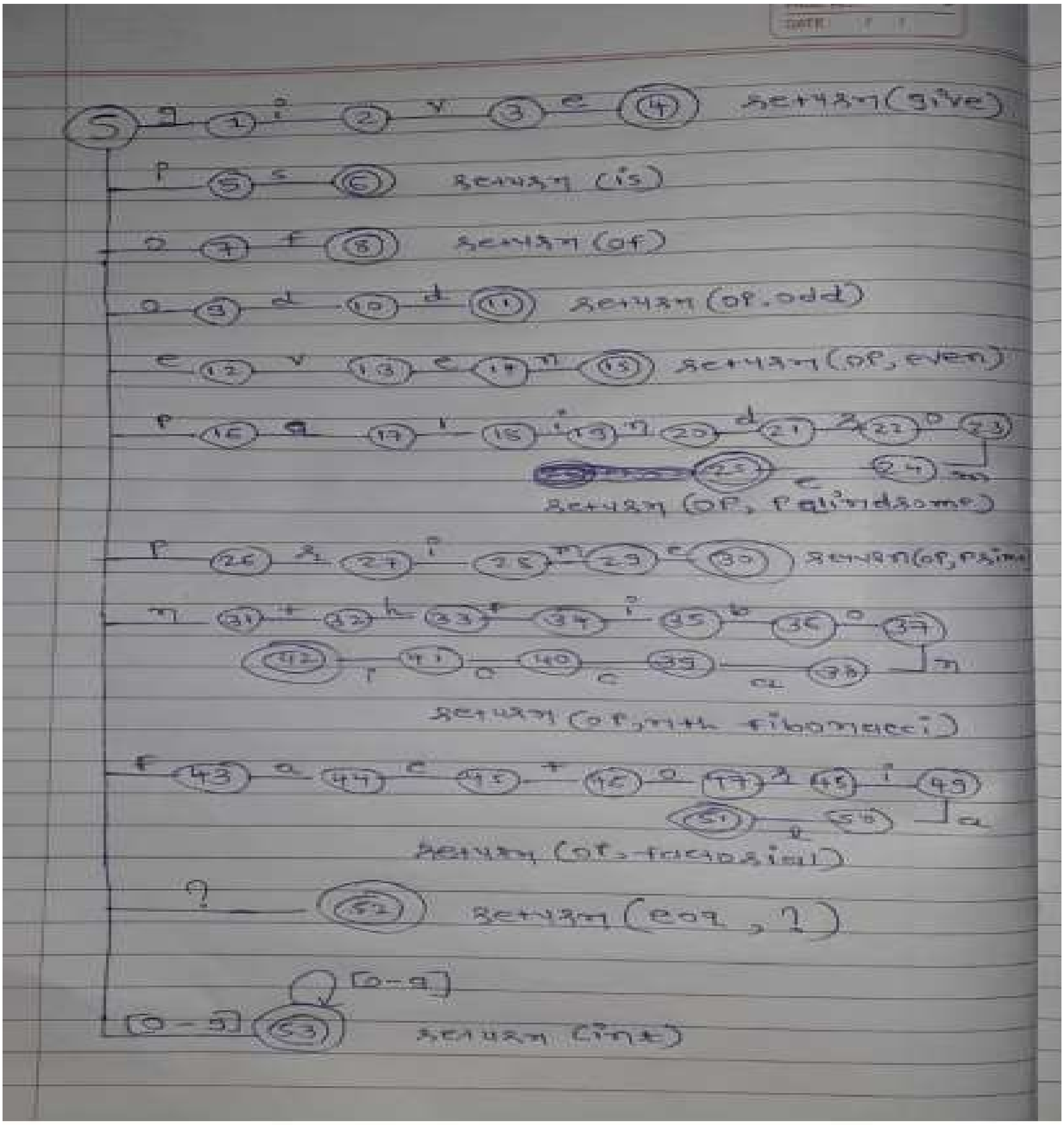
Values type : int

RE	Token
[0-9]+	int

Delimiters : { ? \t}

RE	Token
?	eoq

2.0.2 Deterministic Finite Automata design for lexer



Aim: Design DFA and Algorithm for assigned Language

DFA Design for lexer

Algorithm for Lexer

```
char getNext();
{
return cha[i++];
}
while(ch!='\0')
{
switch(state)
{
case 0:
{
switch(ch)
{
case 'g':
{
state=1;
ch=getNext();
break;
}c
ase 'i':
{
state=5;
ch=getNext();
break;
}c
ase 'o':
{
state=7;
ch=getNext();
break;
}c
ase 'o':
{
state=9;
ch=getNext();
break;
}c
ase 'e':
{
```

```
state=12;
ch=getNext();
break;
}c
ase 'p':
{
state=16;
ch=getNext();
break;
}c
ase 'p':
{
state=26;
ch=getNext();
break;
}c
ase 'n':
{
state=31;
ch=getNext();
break;
}c
ase 'f':
{
state=43;
ch=getNext();
break;
}c
ase '?':
{
state=16;
ch=getNext();
break;
}c
ase '0-9':
{
state=53;
ch=getNext();
break;
} c
ase ' ':
{
state=0;
ch=getNext();
break;
}d
```

```
efault:
{s
tate=53;
}
}b
reak;
case 1:
{
if(ch=='i')
{
state=2;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 2:
{
if(ch=='v')
{
state=3;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 3:
{
if(ch=='e')
{
state=4;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 4:
```

```
{
printf("give: Triger\n");
state=0;
break;
}c
ase 5:
{
if(ch=='s')
{
state=6;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 6:
{
printf("is: Triger\n");
state=0;
break;
}c
ase 7:
{
if(ch=='f')
{
state=8;
}e
lse
{
state=53;
}
ch=getNext();
break;
}c
ase 8:
{
printf("of: Triger\n");
state=0;
break;
}c
ase 9:
{
if(ch=='d')
```

```
{
state=10;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 10:
{
if(ch=='d')
{
state=11;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 11:
{
printf("odd: Triger\n");
state=0;
break;
}c
ase 12:
{
if(ch=='v')
{
state=13;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 13:
{
if(ch=='e')
{
```

```
state=14;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 14:
{
if(ch=='n')
{
state=15;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 15:
{
printf("even: Triger\n");
state=0;
break;
}
case 16:
{
if(ch=='a')
{
state=17;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 17:
{
if(ch=='l')
{
state=18;
```

```
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 18:
{
if(ch=='i')
{
state=19;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 19:
{
if(ch=='n')
{
state=20;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 20:
{
if(ch=='d')
{
state=21;
}e
lse
{
state=53;
}c
h=getNext();
break;
```

```
}c
ase 21:
{
if(ch=='r')
{
state=22;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 22:
{
if(ch=='0')
{
state=23;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 23:
{
if(ch=='m')
{
state=24;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 24:
{
if(ch=='e')
{
state=25;
}e
```

```
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 25:
{
printf("palindrome: Triger\n");
state=0;
break;
}c
ase 26:
{
if(ch=='r')
{
state=27;
}
else
{
state=53;
}c
h=getNext();
break;
}c
ase 27:
{
if(ch=='i')
{
state=28;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 28:
{
if(ch=='m')
{
state=29;
}e
lse
```

```
{
state=53;
}c
h=getNext();
break;
}c
ase 29:
{
if(ch=='e')
{
state=30;
}e
lse
{
state=53;
}
ch=getNext();
break;
}c
ase 30:
{
printf("prime: Triger\n");
state=0;
break;
} c
ase 31:
{
if(ch=='t')
{
state=32;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 32:
{
if(ch=='h')
{
state=33;
}e
lse
{
```

```
state=53;
}c
h=getNext();
break;
}c
ase 33:
{
if(ch=='f')
{
state=34;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 34:
{
if(ch=='i')
{
state=35;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 35:
{
if(ch=='b')
{
state=36;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 36:
{
```

```
if(ch=='o')
{
state=37;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 37:
{
if(ch=='n')
{
state=38;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 38:
{
if(ch=='a')
{
state=39;
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 39:
{
if(ch=='c')
{
state=40
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 40
{
if(ch=='c')
{
state=41
}e
lse
{
state=53;
}c
h=getNext();
break;
} c
ase 41
{
if(ch=='i')
{
state=42
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 42
{
printf("nth fidonacci: Triger\n");
state=0;
break;
}c
ase 43
{
if(ch=='a')
{
state=44
}e
lse
{
state=53;
}c
```

```
h=getNext();
break;
}c
ase 44
{
if(ch=='c')
{
state=45
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 45
{
if(ch=='t')
{
state=46
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 46
{
if(ch=='o')
{
state=47
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 47
{
if(ch=='r')
{
```

```
state=48
}e
lse
{
state=53;
}c
h=getNext();
break;
} c
ase 48
{
if(ch=='i')
{
state=49
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 49
{
if(ch=='a')
{
state=50
}e
lse
{
state=53;
}c
h=getNext();
break;
} c
ase 50
{
if(ch=='l')
{
state=51
}e
lse
{
state=53;
}c
h=getNext();
```

```
break;
}c
ase 52
{
printf("factorial: Triger\n");
state=0;
break;
}c
ase 51
{
if(ch=='?')
{
state=52
}e
lse
{
state=53;
}c
h=getNext();
break;
}c
ase 53
{
printf("?: Triger\n");
state=0;
break;
}c
ase 54
{
if(ch=='0-9')
{
state=53
}e
lse
{
state=53;
}
ch=getNext();
break;
}c
ase 55
{
printf("0-9: Triger\n");
state=0;
break;
}}c
```

```
ase state of
"4" |"6"|"8" :
print('keyword');
"11"|"'15"|"25"|"30"|"42"|"51" :
print('operator');
"52" :
print('eoq');
"53" :
print('int');
}
default
{
printf("Invalid\n");
ch:=nextchar();
end case;
}
}
```

2.0.5 Execution environment setup

Step by Step Guide to Install FLEX and Run FLEX ProgramusingCommand Prompt(cmd)

Step 1

/*For downloading CODEBLOCKS */

- Open your Browser and type in "codeblocks"
- Goto to Code Blocks and go to downloads section - Click on "Download the binary release"
- Download codeblocks-20.03mingw-setup.exe
- Install the software keep clicking on next

/*For downloading FLEX GnuWin32 */

- Open your Browser and type in "download flex gnuwin32" - Goto to "Download GnuWin from SourceForge.net" - Downloading will start automatically
- Install the software keep clicking on next

/*SAVE IT INSIDE C FOLDER*/

Step 2 /*PATH SETUP FOR CODEBLOCKS*/

- After successful installation

Goto program files->CodeBlocks-->MinGW-->Bin - Copy the address of bin :it should somewhat look like this

C:\Program Files (x86)\CodeBlocks\MinGW\bin

- Open Control Panel-->Goto System-->Advance System Settings-->Environment Variables
- Environment Variables--> Click on Path which is inside Systemvariables Click on edit
- Click on New and paste the copied path to it:- C:\Program Files (x86)\CodeBlocks\MinGW\bin -
Press Ok!

Step 3 /*PATH SETUP FOR GnuWin32*/

- After successful installation Goto C folder
- Goto GnuWin32-->Bin
- Copy the address of bin it should somewhat look like this C:\GnuWin32\bin
- Open Control Panel-->Goto System-->Advance System Settings-->Environment Variables
- Environment Variables--> Click on Path which is inside Systemvariables

Click on edit

- Click on New and paste the copied path to it:- C:\GnuWin32\bin - Press Ok!

/*WARNING!!! PLEASE MAKE SURE THAT PATH OF CODEBLOCKSIS BEFORE GNUWIN32---THE ORDER MATTERS*/

Step 4

- Create a folder on Desktop flex_programs or whichever name you like - Open notepad type in a flex program - Save it inside the folder like filename.l

-Note :- also include “”” void yywrap(){} “””””” in the .l file

/*Make sure while saving save it as all files rather than as a text document*/ Step 5 /*To RUN FLEX PROGRAM*/

- Goto to Command Prompt(cmd)
- Goto the directory where you have saved the program - Type in command :- **flex filename.l**
- Type in command :- **gcc lex.yy.c**
- Execute/Run for windows command promt :- **a.exe**

Step 6

- Finished

2.0.4 Implementation of lexer

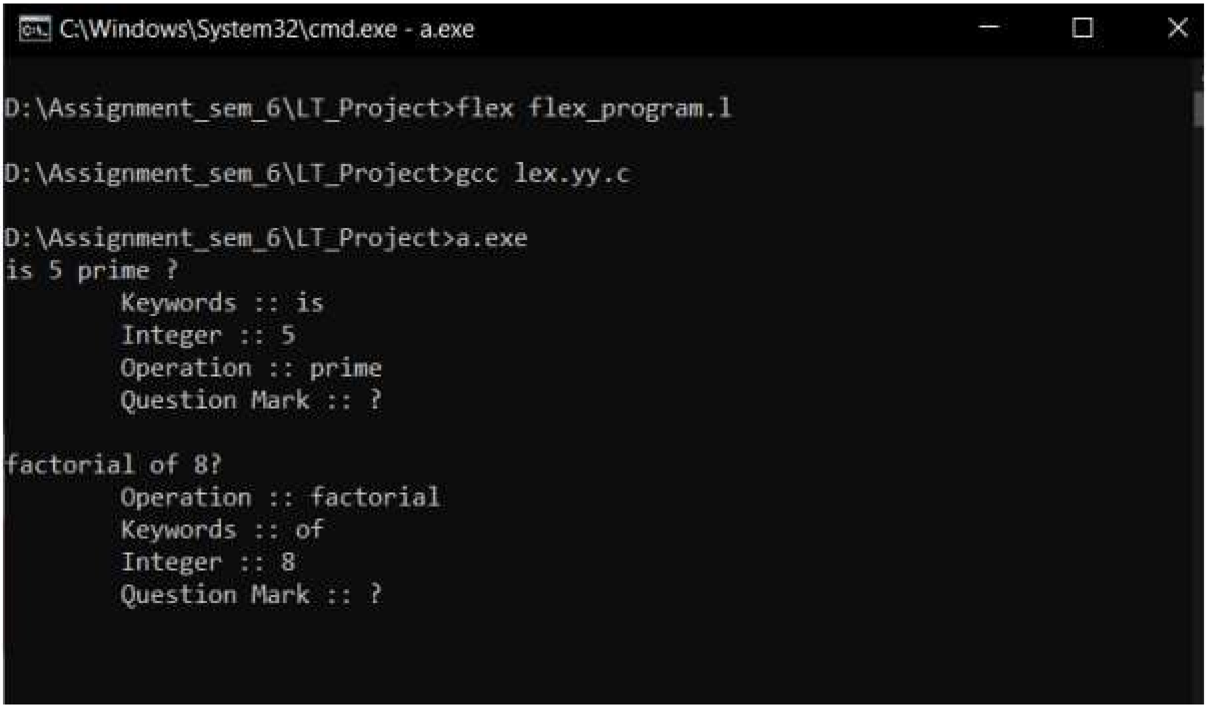
Flex Program:

```
%{
#include<stdio.h>
%}
keywords "give"|"of"|"is"
operation "factorial"|"palindrome"|"prime"|"even"|"odd"|"nth
fibonacci"
digit [0-9]
Int {digit}
Float {digit}+(.{digit})
quest "?"
%x LEXING_ERROR
%%
{keywords} {printf("\tKeywords :: %s \n", yytext);}
{operation} {printf("\tOperation :: %s \n", yytext);}
{Int} {printf("\tInteger :: %s \n", yytext);}
{Float} {printf("\tFloat :: %s \n", yytext);}
{quest} {printf("\tQuestion Mark :: %s \n", yytext);}
" " {}
. { printf("\tINVALID TOKEN \n"); }
%%

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

2.0.6 Output screenshots of lexer.

Output:



```
D:\Assignment_sem_6\LT_Project>flex flex_program.l

D:\Assignment_sem_6\LT_Project>gcc lex.yy.c

D:\Assignment_sem_6\LT_Project>a.exe
is 5 prime ?
        Keywords :: is
        Integer :: 5
        Operation :: prime
        Question Mark :: ?

factorial of 8?
        Operation :: factorial
        Keywords :: of
        Integer :: 8
        Question Mark :: ?
```

3.0 SYNTAX ANALYZER DESIGN

3.0.1 Grammar rules and First and Follow

Grammer

S-> K1 S1 EOQ | K2

S2 EOQ

S1-> OP1 K3 NUM

S2-> NUM OP2

K1-> give K2-> is K3-

> of NUM-> int

OP1-> factorial |

palindrome |

nthFibonacci OP2->

prime | even | odd

EOQ-> ?

Terminals and Non-Terminals

Terminals :-{give, is, of, int, factorial, palindrome, nthFibonacci, prime, even, odd, ? }

Non-Terminal :- { S, S1, S2, K1, K2, K3, OP1, OP2, NUM, EOQ }

punctuation : { ? } integer : { int }

first:

first(NUM) : {int}

first(OP1) : {factorial,palindrome,nthFibonacci}

first(OP2) : {prime,even,odd}

first(EOQ) : {?}

first(S) : {give,is}

first(S1) : {factorial,palindrome,nthFibonacci}

first(S2) : {int}

follow:

follow(S) : {$}

follow(K1) : {factorial,palindrome,nthFibonacci}

follow(K2) : {int} follow(K3) : {int}

follow(NUM) : {prime,even,odd,?}

follow(OP1) : {of} follow(OP2) : {?}

follow(EOQ) : {$} follow(S1) : {?}

follow(S2) : {?}

3.0.2 Yacc based implementation of syntax analyzer · project.l (Lex file)

```
%{
#include<stdio.h>
#include "lab10.tab.h"
%}
keyword "give"|"is"|"of"|
keyword1 "give"
keyword2 "is"
keyword3 "of"
op "prime"|"odd"|"palindrome"|"factorial"|"nthFibonacci"|"even"
op1 "odd"|"even"|"prime"
op2 "palindrome"|"factorial"|"nthFibonacci"
digit [0-9]
integer {digit}+
eoq "?"
ws " "
%%
{keyword1} {
printf("%10s : keyword1\n",yytext);
return KEYWORD1;
} {keyword2}
{
printf("%10s : keyword2\n",yytext);
return KEYWORD2;
} {keyword3}
{
printf("%10s : keyword3\n",yytext);
return KEYWORD3;
}
{op1} {
printf("%10s : oprator1\n",yytext);
return OPERATOR1;
} {op2}
{
```

```
printf("%10s : oprator2\n",yytext);
return OPERATOR2;
} {integer}
{
printf("%10s : integer\n",yytext);
return NUMBER;
}
{eoq} {
printf("%10s : end of question\n",yytext);
return EOQ;
} {ws}
{
return WHITESPACE;
}
. {
printf("%10s : invalid\n",yytext);
}
%%
int yywrap(){
return 1;
}
```

project.y (yacc code)

```
%{

#include<stdio.h>

#include<stdlib.h>

#define

YYERROR_VERBOSE 1

void yyerror(char *err);

%}

%token KEYWORD1

KEYWORD2

KEYWORD3

OPERATOR1

OPERATOR2 NUMBER

EOQ WHITESPACE

%%

s: a {printf("\nthis

sentence is valid.\n");

return 0;};
```

```
a: KEYWORD1
WHITESPACE s1
WHITESPACE EOQ {}
| KEYWORD2
WHITESPACE s1
WHITESPACE EOQ {};
s1: OPERATOR2
WHITESPACE
KEYWORD3
WHITESPACE
NUMBER {} |
NUMBER
WHITESPACE
OPERATOR1 {};
%%
void yyerror(char *err) {
printf("Error: ");
fprintf(stderr, "%s\n", err);
exit(1);
}

void main(){

printf("Enter String: ");

yyparse();

printf("\n valid

Expression...\n");

}
```

3.0.3 Execution environment setup Download flex and bison from the given links.

http://gnuwin32.sourceforge.net/packages/flex.htm
http://gnuwin32.sourceforge.net/packages/bison.htm

when installing on windows you store this in c:/gnuwin32 folder andnot in c:/program files(X86)/gnuwin32

Download IDE

https://sourceforge.net/projects/orwelldevcpp/ set environment variable for flex and bison.

To run the program:

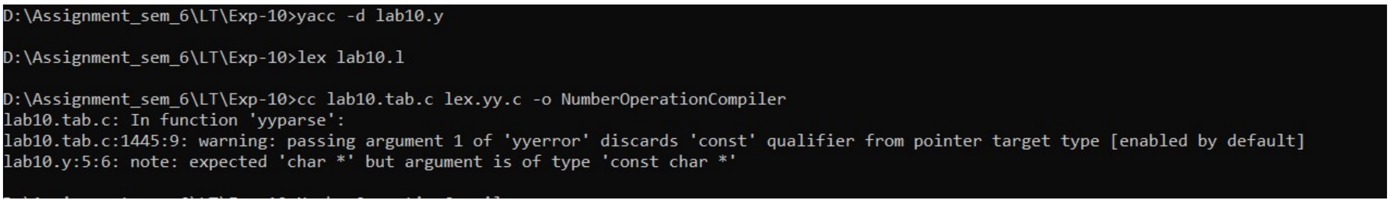
Open a prompt, cd to the directory where your ".l" and ".y" are, andcompile them with: yacc -d yacc.y

```
lex lex.l

gcc yacc.tab.c lex.yy.c -o Compiler.exe

Compiler.exe
```

3.0.4 Output screenshots of yacc based implementation

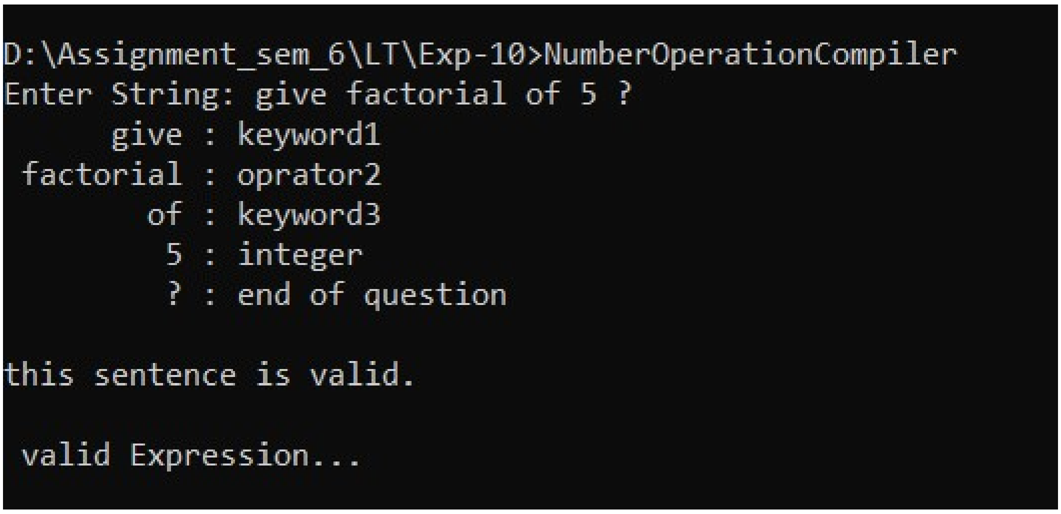


```
D:\Assignment_sem_6\LT\Exp-10>yacc -d lab10.y

D:\Assignment_sem_6\LT\Exp-10>lex lab10.l

D:\Assignment_sem_6\LT\Exp-10>cc lab10.tab.c lex.yy.c -o NumberOperationCompiler
lab10.tab.c: In function 'yyparse':
lab10.tab.c:1445:9: warning: passing argument 1 of 'yyerror' discards 'const' qualifier from pointer target type [enabled by default]
lab10.y:5:6: note: expected 'char *' but argument is of type 'const char *'
```

Factorial :

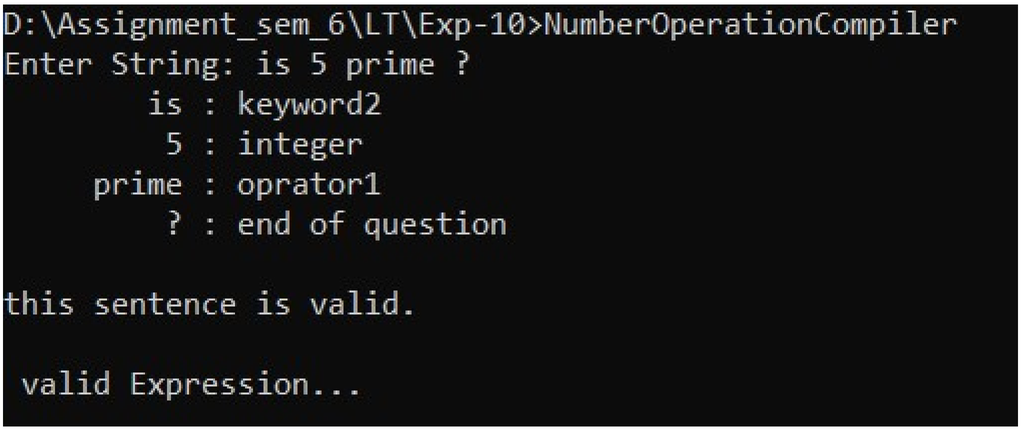


```
D:\Assignment_sem_6\LT\Exp-10>NumberOperationCompiler
Enter String: give factorial of 5 ?
      give : keyword1
 factorial : oprator2
        of : keyword3
         5 : integer
         ? : end of question

this sentence is valid.

 valid Expression...
```

Prime :

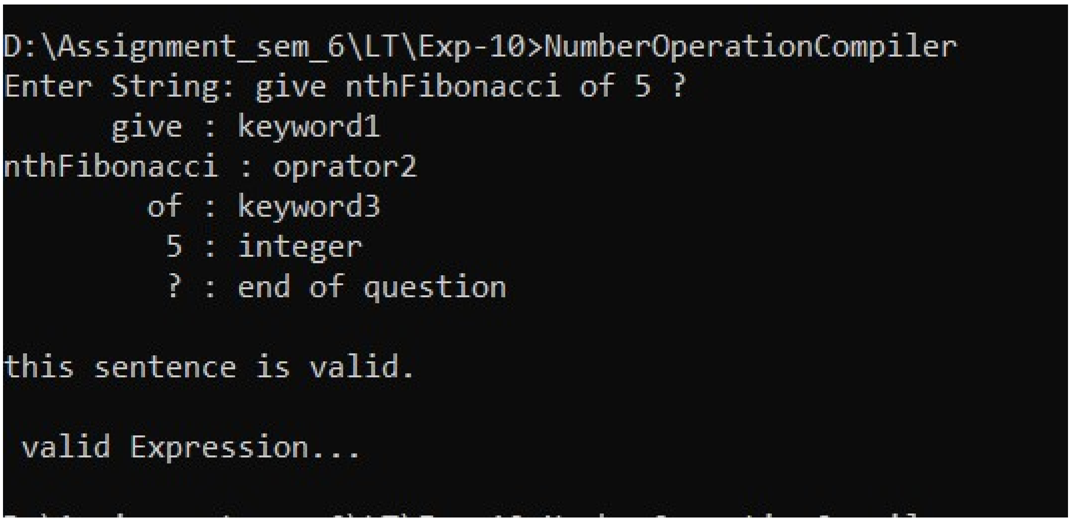


```
D:\Assignment_sem_6\LT\Exp-10>NumberOperationCompiler
Enter String: is 5 prime ?
        is : keyword2
         5 : integer
     prime : oprator1
         ? : end of question

this sentence is valid.

 valid Expression...
```

Fibonacci of n :

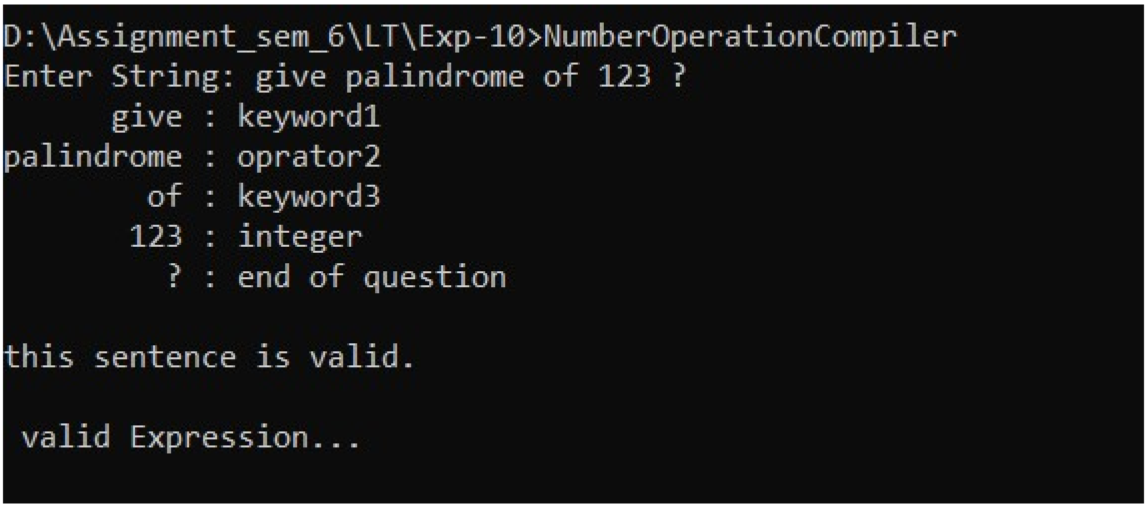


```
D:\Assignment_sem_6\LT\Exp-10>NumberOperationCompiler
Enter String: give nthFibonacci of 5 ?
      give : keyword1
nthFibonacci : oprator2
        of : keyword3
         5 : integer
         ? : end of question

this sentence is valid.

 valid Expression...
```

Palindrome :

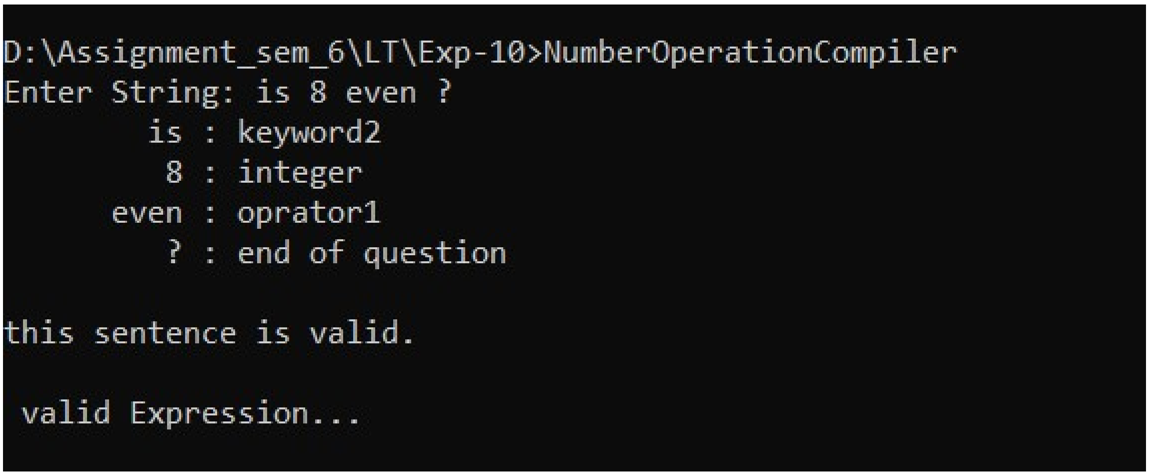


```
D:\Assignment_sem_6\LT\Exp-10>NumberOperationCompiler
Enter String: give palindrome of 123 ?
      give : keyword1
palindrome : oprator2
        of : keyword3
       123 : integer
         ? : end of question

this sentence is valid.

 valid Expression...
```

Even :

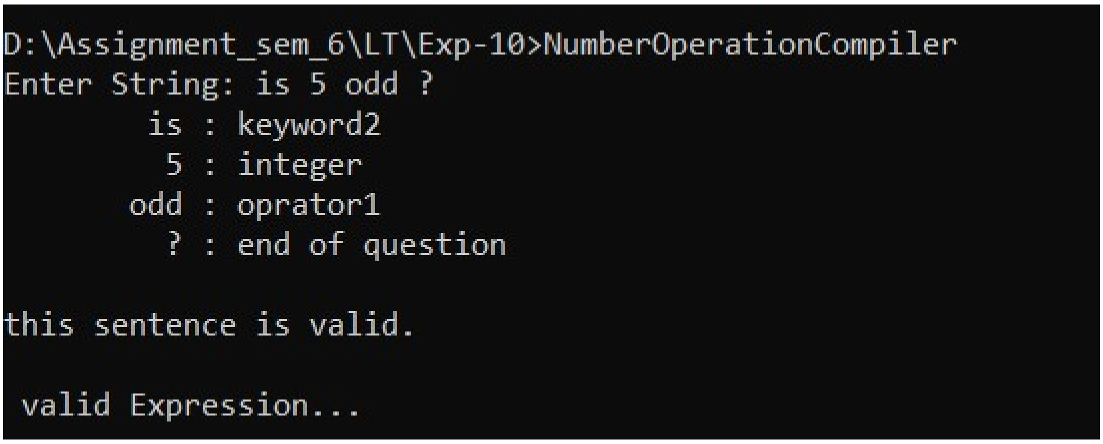


```
D:\Assignment_sem_6\LT\Exp-10>NumberOperationCompiler
Enter String: is 8 even ?
        is : keyword2
         8 : integer
      even : oprator1
         ? : end of question

this sentence is valid.

 valid Expression...
```

Odd :

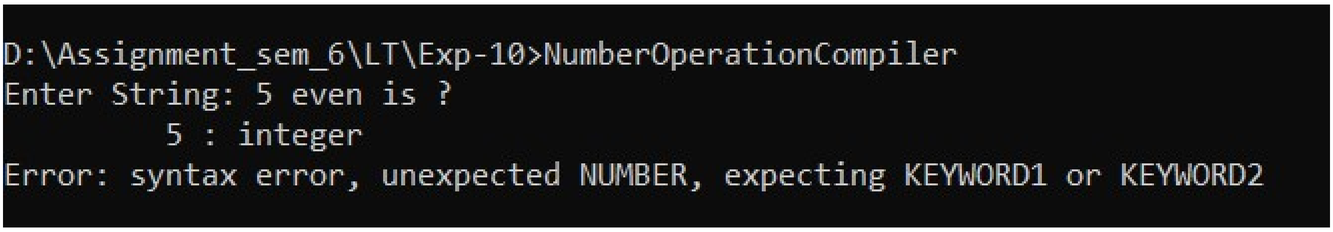


```
D:\Assignment_sem_6\LT\Exp-10>NumberOperationCompiler
Enter String: is 5 odd ?
        is : keyword2
         5 : integer
       odd : oprator1
         ? : end of question

this sentence is valid.

 valid Expression...
```

Invalid



```
D:\Assignment_sem_6\LT\Exp-10>NumberOperationCompiler
Enter String: 5 even is ?
         5 : integer
Error: syntax error, unexpected NUMBER, expecting KEYWORD1 or KEYWORD2
```

4.0 CONCLUSION

This project has been implemented from what we have learned in our college curriculum and many rich resources from the web. After doing this project we conclude that we have got more knowledge about how different compilers are working in the practical world and also how various types of errors are handled.