A

Project Report

on

**Compiler for**

**Mathematical operations**

**using**

**English like sentences**

Developed by

**Karia Stuti-IT055-18ITUON119**

**Kathiriya Darshakkumar-IT057-18ITUOS097**

**Nihal Limbani-IT064-18ITUOF051**



**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 “**Mathematical operations using**

### English like sentences” is a bonafied report of the work carried out by

1) Miss Karia Stuti, Student ID No: 18ITUON119

2) Mr. Kathiriya Darshakkumar, Student ID No: 18ITUOS097

3) Mr. Limbani Nihal, Student ID No: 18ITUOF051

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 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

1. Introduction

1.1 Project Details………………………………………..……………….....1

1.2 Project Planning ……………………..………………………………….2

2.0. Lexical phase design

2.1 Deterministic Finite Automaton design for lexer………………………3

2.2 Algorithm of lexer……………………………………………………..4

2.3 Implementation of lexer………………………………………………..9

2.4 Execution environment setup………………………………………….12

2.5 Output screenshots of lexer…………………………………………….14

3.0 Syntax analyzer design

3.1 Grammar rules…………………………………………………………....18

3.2 Yacc based imlementation of syntax analyzer…………………………...18

3.3 Execution environment setup …………………………………………....20

3.4 Output screenshots of yacc based implementation……………………….21

4.0 Conclusion ……………….…………………………………...…….24

# INTRODUCTION

## Project Details

**Grammar Name:** Mathematical operations using English like sentences

### Grammar Rules:

Write an appropriate language description for a layman language which can do mathematical operations using English like sentences.

Example of valid program in this language is –

Add 100,200,300,400.

Subtract 250 from result.

Multiply 400 to it.

Divide the answer by 2.

Show me the answer.

### Regular Expression:

1. **Regular definition for layman Language:**

|  |  |
| --- | --- |
| Regular Defination | Examples: |
| Keywords | From, from, Show, show, by, to, it, result, the, me, Answer, answer |
| Operation | Add, Sub(Subtract), Mul(Multiply), Div(Divide) |
| Digit(Number) | [0-9] |
| Que. Mark | “?”(EOF) |
| White Space | (Tab | Newline)**+** |
| Letter | [A-Za-z] |

1. **Regular Expression Related to Regular Language:**

|  |  |
| --- | --- |
| **Int** | {Digit}+- Atleast one or more Digit |
| **Float** | {Digit}+(\.{Digit}+)?(E[+\-]?{Digit}+)? – Means Digit followed by digit or exponent of 10(digit) |
| **Space** | {white space}+ |

## Project Planning

**List of Students with their Roles/Responsibilities:**

1. Karia Stuti (IT055)

* DFA design
* Regular Expressions

1. Kathiriya Darshakkumar (IT057)

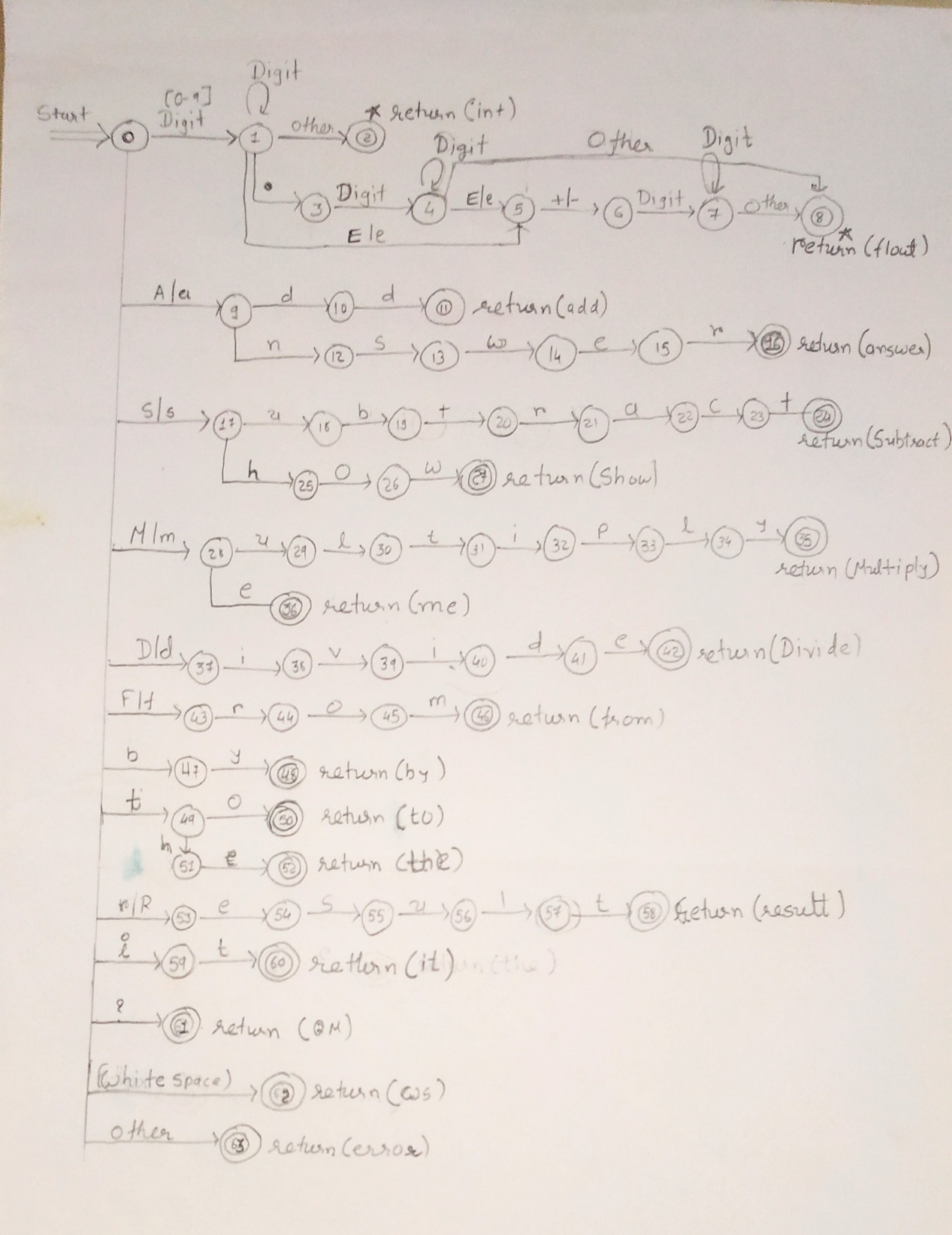
* Grammar Rules
* Complete parser with Flex and Bison (Yacc) code
* Testing with different test cases
* Documentation

1. Limbani Nihal (IT064)

* Algorithm Design
* Regular Expressions
* C++ implementation

# **LEXICAL PHASE DESIGN**

## Deterministic Finite Automaton design for lexer



## Algorithm of lexer

while not eof do

state := 0;

input(ch);

case state of

0:case ch of

digit: state:= 1;

'a'|'A': state:=9;

's'|'S': state:=17;

'm'|'M': state:=28;

'd'|'D': state:=37;

'f'|'F': state:=43;

'b': state:=47;

't': state:=49;

'r'|'R': state:=53;

'i': state:=59;

'?': state:=61;

' '|'\t: state:=62;

else: state:=63;

end case

1:case ch of

digit: state:=1;

'.': state:=3;

'e'|'E': state:=5;

else: state:2;

end case;

2:case ch of

unput(ch);//return int

exit while;

end case;

3:case ch of

digit: state:=4;

else: exit while;

end case;

4:case ch of

digit: state:=4;

'e'|'E': state:=5;

else: state:=8;

end case;

5:case ch of

'+'|'-': state:=6;

else: exit while;

end case;

6:case ch of

digit: state:=7;

else: exit while;

end case;

7:case ch of

digit: state:=7;

else: state:=8;

end case;

8:case ch of

unput(ch);//return float

exit while;

end case;

9:case ch of

'd': state:=10;

'n': state:=12;

else: exit while;

end case;

10:case ch of

'd': state:=11;

else: exit while;

end case;

11:case ch of

unput(ch);//return add

exit while;

end case;

12:case ch of

's': state:=13;

else: exit while;

end case;

13:case ch of

'w': state:=14;

else: exit while;

end case;

14:case ch of

'e': state:=15;

else: exit while;

end case;

15:case ch of

'r': state:=16;

else: exit while;

end case;

16:case ch of

unput(ch);//return answer

exit while;

end case;

17:case ch of

'u': state:=18;

'h': state:=25;

else: exit while;

end case;

18:case ch of

'b': state:=19;

else: exit while;

end case;

19:case ch of

't': state:=20;

else: exit while;

end case;

20:case ch of

'r': state:=21;

else: exit while;

end case;

21:case ch of

'a': state:=22;

else: exit while;

end case;

22:case ch of

'c': state:=23;

else: exit while;

end case;

23:case ch of

't': state:=24;

else: exit while;

end case;

24:case ch of

unput(ch);//return subtract

exit while;

end case;

25:case ch of

'o': state:=26;

else: exit while;

end case;

26:case ch of

'w': state:=27;

else: exit while;

end case;

27:case ch of

unput(ch);//return show

exit while;

end case;

28:case ch of

'u': state:=29;

'e': state:=36;

else: exit while;

end case;

29:case ch of

'l': state:=30;

else: exit while;

end case;

30:case ch of

't': state:=31;

else: exit while;

end case;

31:case ch of

'i': state:=32;

else: exit while;

end case;

32:case ch of

'p': state:=33;

else: exit while;

end case;

33:case ch of

'l': state:=34;

else: exit while;

end case;

34:case ch of

'y': state:=35;

else: exit while;

end case;

35:case ch of

unput(ch);//return multiply

exit while;

end case;

36:case ch of

unput(ch);//return me

exit while;

end case;

37:case ch of

'i': state:=38;

else: exit while;

end case;

38:case ch of

'v': state:=39;

else: exit while;

end case;

39:case ch of

'i': state:=40;

else: exit while;

end case;

40:case ch of

'd': state:=41;

else: exit while;

end case;

41:case ch of

'e': state:=42;

else: exit while;

end case;

42:case ch of

unput(ch);//return divide

exit while;

end case;

43:case ch of

'r': state:=44;

else: exit while;

end case;

44:case ch of

'o': state:=45;

else: exit while;

end case;

45:case ch of

'm': state:=46;

else: exit while;

end case;

46:case ch of

unput(ch);//return from

exit while;

end case;

47:case ch of

'y': state:=48;

else: exit while;

end case;

48:case ch of

unput(ch);//return by

exit while;

end case;

49:case ch of

'o': state:=50;

'h': state:=51;

else: exit while;

end case;

50:case ch of

unput(ch);//return to

exit while;

end case;

51:case ch of

'e': state:=52;

else: exit while;

end case;

52:case ch of

unput(ch);//return the

exit while;

end case;

53:case ch of

'e': state:=54;

else: exit while;

end case;

54:case ch of

's': state:=55;

else: exit while;

end case;

55:case ch of

'u': state:=56;

else: exit while;

end case;

56:case ch of

'l': state:=57;

else: exit while;

end case;

57:case ch of

't': state:=58;

else: exit while;

end case;

58:case ch of

unput(ch);//return result

exit while;

end case;

59:case ch of

't': state:=60;

else: exit while;

end case;

60:case ch of

unput(ch);//return it

exit while;

end case;

61:case ch of

unput(ch);//return QM

exit while;

end case;

62:case ch of

unput(ch);//return WS

exit while;

end case;

63:case ch of

unput(ch);//return OTHER(error)

exit while;

end case;

exit while;

## Implementation of lexer

**C++ code:**

#include <bits/stdc++.h>

using namespace std;

int operation(char buf[])

{

char op[10][10]={"Add","add","Sub","sub","Mul","mul","Div","div"};

for(int i=0;i<8;i++)

{

if(strcmp(op[i],buf)==0)

return 1;

}

return 0;

}

int keyword(char buf[])

{

char key[20][20]={"From", "from", "Show", "show", "by", "to", "it", "number", "and", "result","Result", "the", "me", "Answer", "answer" };

for(int i=0;i<15;i++)

{

if(strcmp(key[i],buf)==0)

return 1;

}

return 0;

}

int number(char buf[])

{

int n=strlen(buf);

for(int i=0;i<n;i++)

{

if(buf[i]>='0' && buf[i]<='9')

{

return 1;

}

else

{

return 0;

}

}

return 1;

}

int main()

{

FILE \*f;

f=freopen("D:/00 Study/SEM 6/0LAB/LT/LAB 3/inputExpfile2.txt", "r", stdin);

freopen("D:/00 Study/SEM 6/0LAB/LT/LAB 3/output2.txt", "w", stdout);

char ch,buffer[15];

int d=0;

while((ch = fgetc(f)) != EOF){

if(ch=='\n')

{

printf("Started New Line. \n");

continue;

}

if(ch=='?')

{

printf("It shown end of line: %c\n",ch);

continue;

}

if(isalnum(ch))

{

buffer[d++]=ch;

}

else if((ch==' ' || ch=='\n') && (d!=0)){

buffer[d]='\0';

d=0;

if(ch!=',')

{

if(number(buffer)==1){

printf("Number identify: %s\n", buffer);

}

}

else{

printf("\n");

continue;

}

if(operation(buffer)==1)

{

printf("operator identify: %s\n",buffer);

}

else if(keyword(buffer)==1)

{

printf("Keyword identify: %s\n",buffer);

}

}

else if(ch==','||ch==' ')

{

continue;

}

else

{

printf("Not identify token %c\n",ch);

}

}

}

**Flex-code**:

%{

#include<stdio.h>

int totaltk=0;

%}

Keywords "From"|"from"|"show"|"Show"|"by"|"to"|"it"|"the"|"me"|"number"|"and"|"Result"|"result"|"answer"|"Answer"

Operator "Add"|"add"|"Sub"|"sub"|"Mul"|"mul"|"Div"|"div"

Digit [0-9]

QM "?"

WS [\t\n]

Int {Digit}+

Float {Digit}+(\.{Digit}+)?(E[+\-]?{Digit}+)?

Space {WS}+

%%

{Keywords} {printf("Keyword : %s\n",yytext);totaltk++;}

{Operator} {printf("operator is: %s\n",yytext);totaltk++;}

{Int} {printf("Integer : %s\n",yytext);totaltk++;}

{Float} {printf("Float No : %s\n",yytext);totaltk++;}

{QM} {printf("\n");totaltk++;}

{Space} {}

. {}

%%

int yywrap()

{

return 1;

}

int main()

{

yylex();

printf("Total Number of Tokens In our Example: %d\n",totaltk);

return 0;

}

## Execution environment setup

### Step by Step Guide to Install FLEX and Run FLEX Program using Command Prompt(cmd)

**Step 1** /\*For downloading CODEBLOCKS \*/

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

/\*For downloading FLEX GnuWin32 \*/

* Open your Browser and type in "download flex gnuwin32" - Go 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 → Advanced System Settings → Environment Variables
* Environment Variables → Click on path which is inside System variables → 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 → Advanced System Settings → Environment Variables
* Environment Variables → Click on Path which is inside System variables → 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 CODEBLOCKS IS 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\*/

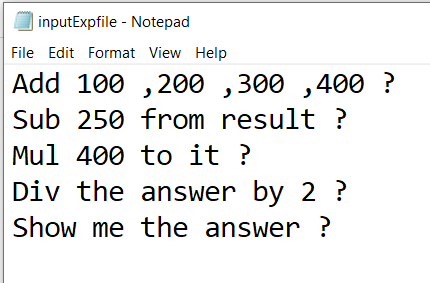
* Goto to Command Prompt(cmd)
* Goto the directory where you have saved the program.
* Type in command :- **flex filename.l**
* Type in command :- **gcc lex.yy.c**
* Execute/Run for windows command prompt :- **a.exe**

**Finished.**

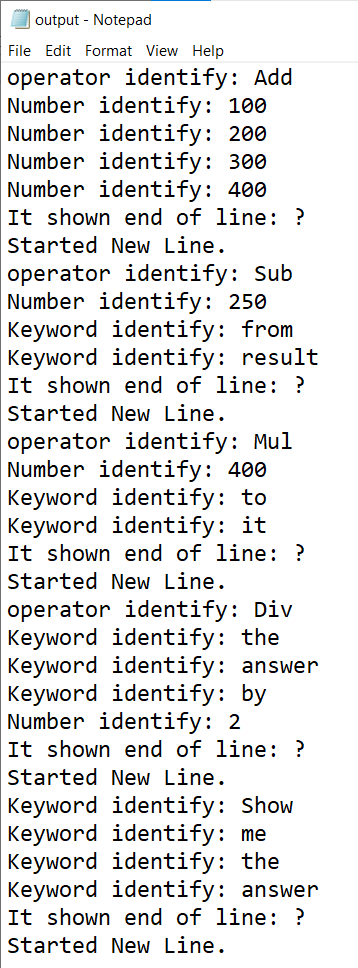
* 1. **Output screenshots of lexer**

**C++ code Output:**

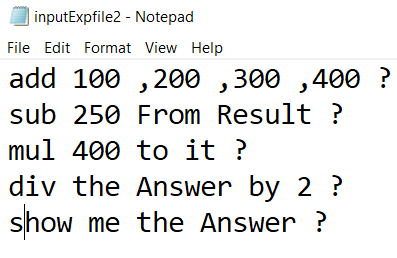
**Input File:**

****

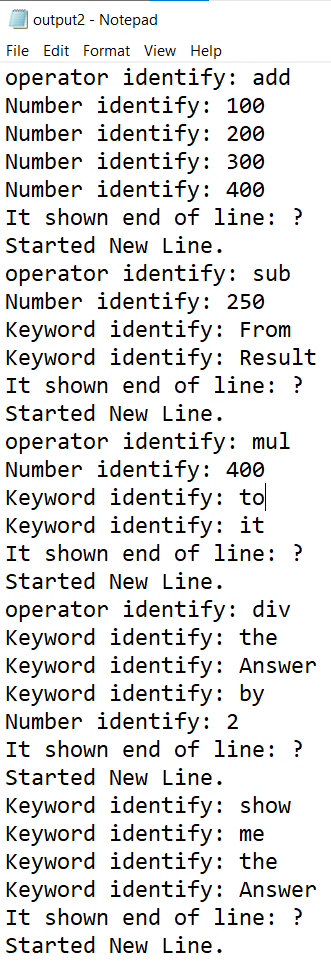
**Output File:**

****

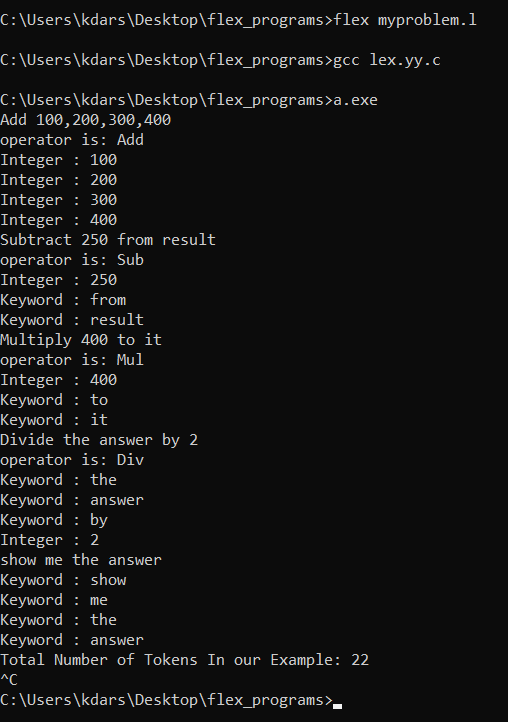
**Input File2: Show case sensitive**

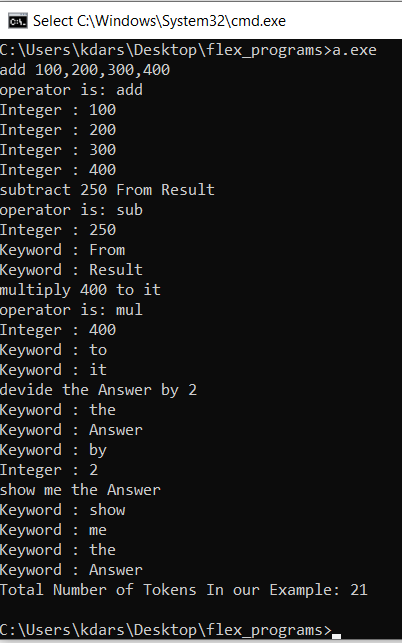
****

**Output File2:**

****

**Flex Code Output:**

****

****

In second ss there is devide instead of divide. =>Show that one less token identify.

All see that case sensitive validation in this ss.

# SYNTAX ANALYZER DESIGN

## Grammar rules

E->S

S-> OPERATION |SHOW

OPERATION -> NUMBER SEMI | NUMBER | KEYWORD

NUMBER-> SEMI NUMBER | KEYWORD | QUE

SHOW-> SHOW KEYWORD

KEYWORD-> KEYWORD | QUE

* 1. **Yacc based implementation of syntax analyzer**
* **lab9.l**

%{

#include<stdio.h>

#include "y.tab.h"

%}

%%

"Add"|"add"|"Sub"|"sub"|"Mul"|"mul"|"Div"|"div" {printf("<%s, OPERATION>\n", yytext); return OPERATION;}

"From"|"from"|"by"|"to"|"it"|"number"|"and"|"result"|"Result"|"the"|"me"|"Answer"|"answer" {printf("<%s, KEYWORD>\n", yytext); return KEYWORD;}

"Show"|"show" {printf("<%s, KEYWORD>\n", yytext); return SHOW;}

"," {return SEMICOLON;}

[0-9]+ {printf("<%s, NUMBER>\n",yytext); return NUMBER;}

"?" {printf("<%s, QUESTION MARK>\n",yytext); return QM;}

\n {return NL;}

. {}

%%

int yywrap(void)

{

return 1;

}

* **lab9.y**

%{

#include<stdio.h>

#include "y.tab.h"

int yyerror(char \*s);

int yyparse(void);

%}

%token OPERATION KEYWORD SHOW SEMICOLON QM NUMBER NL

%%

E : S NL { return 0;}

S : T {printf("\nYour Given String is Valid\n\n");}

T : OPERATION NUMBER SEMICOLON NUMBER SEMICOLON NUMBER SEMICOLON NUMBER QM |

OPERATION NUMBER KEYWORD KEYWORD QM |

OPERATION KEYWORD KEYWORD KEYWORD NUMBER QM|

SHOW KEYWORD KEYWORD KEYWORD QM

%%

int main()

{

while(1)

{

printf("Enter your language input:");

yyparse();

}

}

int yyerror(char \*s)

{

fprintf(stderr,"%s\n",s);

exit(0);

}

## Execution environment setup

* <http://gnuwin32.sourceforge.net/packages/flex.html>
* <http://gnuwin32.sourceforge.net/packages/bison.html>
* When installing on windows you store this in c:/gnuwin32 folder and not in c:/program files(X86)/gnuwin32
* <https://sourceforge.net/projects/orwelldevcpp/>
* Set environment variable and then run program
* Open a prompt, cd to the directory where your ".l" and ".y" are, and compile them with:

flex lab9.l

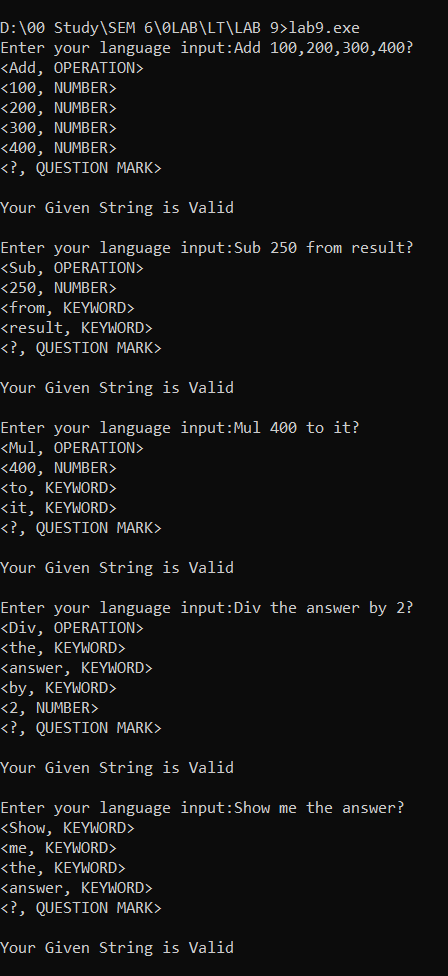
bison -dy lab9.y

gcc lex.yy.c y.tab.c lab9.exe

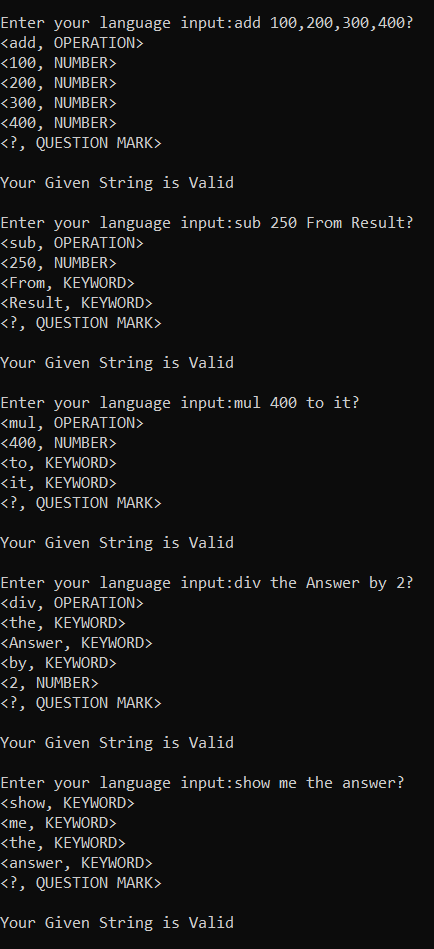
lab9.exe

* 1. **Output screenshots of implementation**

**Case: Valid Input**

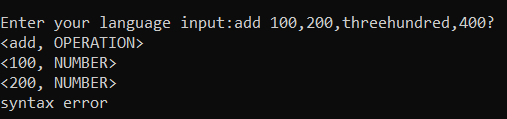


**Case:** Language seen Some Case sensitive Input.

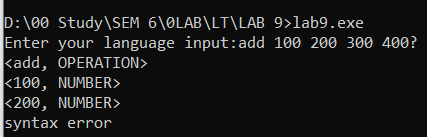


**Invalid Input Error:**

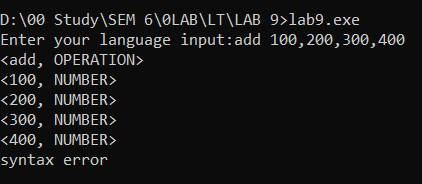
If user enter number in word form then:



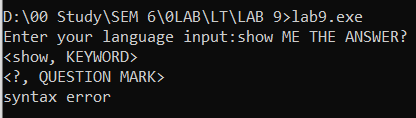
If user forget to enter semicolon in between two number then:



If user forget to enter “?” at the end then:



If user enter invalid case Input then:



# **4.CONCLUSION**

This project has been implemented from what we have learnt in our college curriculum and many resources from the web.

After doing this project we were able to better understand different concepts of Language Translators. We conclude that we have got more knowledge about how different compilers are working and handling the errors.

We would like to thank Prof. Nikita P. Desai for teaching this interesting subject, for the guidance in the project.