Project Report on

Compiler for Layman friendly different countries' currency conversation in rupee program which can do this operation using Gujarati language

Developed by

Parmar Manish -IT092 -19ITUBS070

Parmar Nihal – IT093 – 19ITUBS134

Parmar Riya – IT094 - 19ITUBS128

Guided By:

Prof. Nikita P. Desai

Dept. of Information Technology

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 different countries' currency conversation in rupee program which can do this operation using Gujarati language**"** is a bonafied report of the work carried out by

1) Parmar Manish D., Student ID No: 19ITUBS070
2) Parmar Nihal , Student ID No: 19ITUBS134
3) Parmar Riya , Student ID No:19ITUBS128

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 2021-2022.

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

1.0 INTRODUCTION

1.0.1 Project Details

Language Name: "Layman friendly different countries' currency conversation in rupee program which can do this operation using Gujarati language**"**

Language description:

This Complier will convert some big countries' currency into rupees. This will be a layman friendly Complier. We will only allow some specific currency. If anyone who knows the Gujarati language can use this Complier. Some valid allowed sentences are:

1. 23 dollar na ketla rupiya?
2. 5 pound na ketla rupiya?
3. 200 rupiya na ketla pounds?
4. 3 EURO na ketla rupiya?

1.0.2 Project Planning

List of Students with their Roles/Responsibilities:
Nihal : DFA design

Manish: Lexer algorithm

Riya: Syntax analyzer design

2.0LEXICAL PHASE DESIGN

2.0.1 Regular Expression:

Keywords :

RE	Token
rupiya	rupiya
dollar	dollar
euro	euro
canadaindollar	canadaindollar
chineseyaun	chineseyaun
pound	pound
thaibaht	thaibaht
swissfranc	swissfranc
iraqidinar	iraqidinar
brazilianreal	brazilianreal

Operators:

RE	Token
naa	op
ketla	op

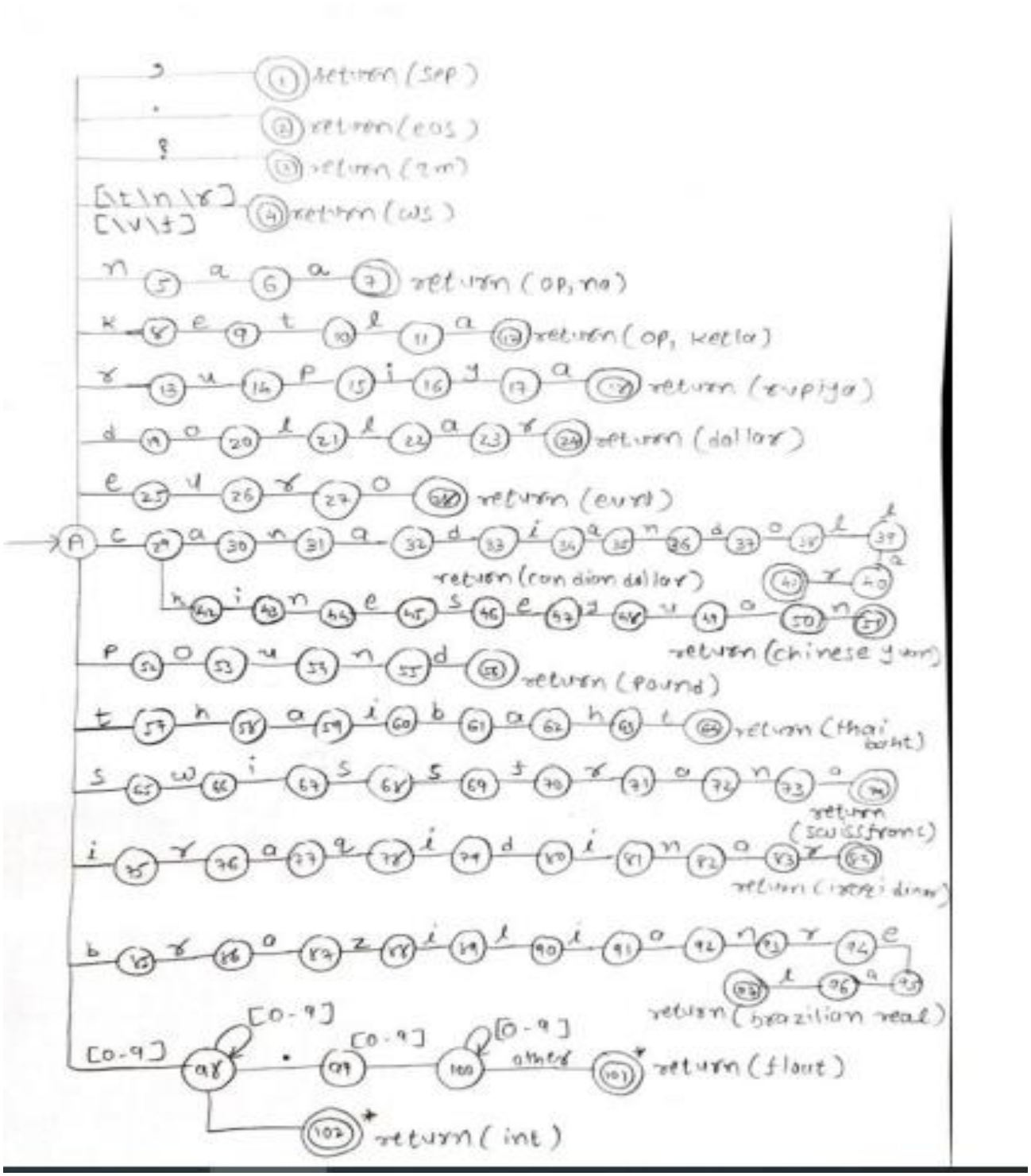
Values type : int and float

RE	Token
[0-9]+	int
[0-9]+(.[0-9]+)	float

Delimiters : {. , ? \t}

RE	Token
.	eos
,	sep
?	qm
[\t\n\r\v\f]	ws

2.0.2 Deterministic Finite Automata design for lexer



2.0.3 Algorithm of lexer

```
lexer {
int c =
0;
bool f = false;
int len = string.length();
while not eof do
{
state="A";
while not eof do (c < len)
{
if (f)
{
f= false;
}
char ch = nextchar();
switch (state) {
case state of "A":
case state of
',':
state = "1";
ch = nextchar();
f = true;
break;
'.':
state = "2";
ch = nextchar();
f = true;
```

```
break;
'?':
state = "3";
ch = nextchar();
f = true;
break;
[\t\n\r\v\f]:
state = "4";
ch = nextchar();
f= true;
break;
'n':
state = "5";
ch = nextchar();
break;
'k':
state = "8";
ch = nextchar();
break;
'r':
state = "13";
ch = nextchar();
break;
'd':
state = "19"
```

```
ch = nextchar();
break;
'e':
state = "25";
ch = nextchar();
break;
'c':
state = "29";
ch = nextchar();
break;
'p':
state = "52";
ch = nextchar();
break;
't':
state = "57";
ch = nextchar();
break;
's':
state = "65";
ch = nextchar();
break;
'i':
state = "75";
ch = nextchar();
```

```
break;
'b':
state = "85";
ch = nextchar();
break;
[0-9]:
state = "98";
ch = nextchar();
break;
}
default:
f= true;
end case
case state of "5":
case state of 'a':
state = "6";
ch = nextchar();
case state of "6":
case state of 'a':
state = "7";
ch = nextchar();
f= true;
case state of "8":
case state of 'e':
state = "9";
ch = nextchar();
```

```
case state of "9":
case state of 't':
state = "10";
ch = nextchar();
case state of "10":
case state of 'l':
state = "11";
ch = nextchar();
case state of "11":
case state of 'a':
state = "12";
ch = nextchar();
f= true;
case state of "13":
case state of 'u':
state = "14";
ch = nextchar();
case state of "14":
case state of 'p':
state = "15";
ch = nextchar();
case state of "15":
case state of 'i':
state = "16";
ch = nextchar();
case state of "16":
case state of 'y':
state = "17";
```

```
ch = nextchar();
case state of "17":
case state of 'a':
state = "18";
ch = nextchar();
f = true;
case state of "19":
case state of 'o':
state = "20";
ch = nextchar();
case state of "20":
case state of 'l':
state = "21";
ch = nextchar();
case state of "21":
case state of 'l':
state = "22";
ch = nextchar();
case state of "22":
case state of 'a':
state = "23";
ch = nextchar();
case state of "23":
case state of 'r':
state = "24";
ch = nextchar();
f= true;
```

```
case state of "25":
case state of 'u':
state = "26";
ch = nextchar();
case state of "26":
case state of 'r':
state = "27";
ch = nextchar();
case state of "27":
case state of 'o':
state = "28";
ch = nextchar();
f= true;
case state of "29":
case state of 'a':
state = "30";
ch = nextchar();
break;
case state of "h":
state = "42";
ch = nextchar();
break;
default:
f= true;
case state of "30":
case state of 'n':
state = "31";
ch = nextchar();
```

```
case state of "31":
case state of 'a':
state = "32";
ch = nextchar();
case state of "32":
case state of 'd':
state = "33";
ch = nextchar();
case state of "33":
case state of 'i':
state = "34";
ch = nextchar();
case state of "34":
case state of 'a':
state = "35";
ch = nextchar();
case state of "35":
case state of 'n':
state = "36";
ch = nextchar();
case state of "36":
case state of 'd':
state = "37";
ch = nextchar();
case state of "37":
case state of '0':
state = "38";
```

```
ch = nextchar();
case state of "38":
case state of 'l':
state = "39";
ch = nextchar();
case state of "39":
case state of 'l':
state = "40";
ch = nextchar();
case state of "40":
case state of 'a':
state = "41";
ch = nextchar();
f= true;
case state of "42":
case state of 'i':
state = "43";
ch = nextchar();
case state of "43":
case state of 'n':
state = "44";
ch = nextchar();
case state of "44":
case state of 'e':
state = "45";
ch = nextchar();
case state of "45":
```

```
case state of 's':
state = "46";
ch = nextchar();
case state of "46":
case state of 'e':
state = "47";
ch = nextchar();
case state of "47":
case state of 'y':
state = "48";
ch = nextchar();
case state of "48":
case state of 'u':
state = "49";
ch = nextchar();
case state of "49":
case state of 'a':
state = "50";
ch = nextchar();
case state of "50":
case state of 'n':
state = "51";
ch = nextchar();
f= true;
case state of "52":
case state of 'o':
state = "53";
ch = nextchar();
```

```
case state of "53":
case state of 'u':
state = "54";
ch = nextchar();
case state of "54":
case state of 'n':
state = "55";
ch = nextchar();
case state of "55":
case state of 'd':
state = "56";
ch = nextchar();
f= true;
case state of "57":
case state of 'h':
state = "58";
ch = nextchar();
case state of "58":
case state of 'a':
state = "59";
ch = nextchar();
case state of "59":
case state of 'i':
state = "60";
ch = nextchar();
case state of "60":
case state of 'b':
state = "61";
```

```
ch = nextchar();
case state of "61":
case state of 'a':
state = "62";
ch = nextchar();
case state of "62":
case state of 'h':
state = "63";
ch = nextchar();
case state of "63":
case state of 't':
state = "64";
ch = nextchar();
f= true;
case state of "65":
case state of 'w':
state = "66";
ch = nextchar();
case state of "66":
case state of 'i':
state = "67";
ch = nextchar();
case state of "67":
case state of 's':
state = "68";
ch = nextchar();
case state of "68":
case state of 's':
```

```
state = "69";
ch = nextchar();
case state of "69":
case state of 'f':
state = "70";
ch = nextchar();
case state of "70":
case state of 'r':
state = "71";
ch = nextchar();
case state of "71":
case state of 'a':
state = "72";
ch = nextchar();
case state of "72":
case state of 'n':
state = "73";
ch = nextchar();
case state of "73":
case state of 'c':
state = "74";
ch = nextchar();
f= true;
case state of "75":
case state of 'r':
state = "76";
ch = nextchar();
case state of "76":
```

```
case state of 'a':
state = "77";
ch = nextchar();
case state of "77":
case state of 'q':
state = "78";
ch = nextchar();
case state of "78":
case state of 'i':
state = "79";
ch = nextchar();
case state of "79":
case state of 'd':
state = "80";
ch = nextchar();
case state of "80":
case state of 'i':
state = "81";
ch = nextchar();
case state of "81":
case state of 'n':
state = "82";
ch = nextchar();
case state of "82":
case state of 'a':
state = "83";
ch = nextchar();
case state of "83":
```

```
case state of 'r':
state = "84";
ch = nextchar();
f= true;
case state of "85":
case state of 'r':
state = "86";
ch = nextchar();
case state of "86":
case state of 'a':
state = "87";
ch = nextchar();
case state of "87":
case state of 'z':
state = "88";
ch = nextchar();
case state of "88":
case state of 'i':
state = "89";
ch = nextchar();
case state of "89":
case state of 'l':
state = "90";
ch = nextchar();
case state of "90":
case state of 'i':
state = "91";
ch = nextchar();
```

```
case state of "91":
case state of 'a':
state = "92";
ch = nextchar();
case state of "92":
case state of 'n':
state = "93";
ch = nextchar();
case state of "93":
case state of 'r':
state = "94";
ch = nextchar();
case state of "94":
case state of 'e':
state = "95";
ch = nextchar();
case state of "95":
case state of 'a':
state = "96";
ch = nextchar();
case state of "96":
case state of 'l':
state = "97";
ch = nextchar();
f= true;
case state of "98":
case state of [0-9]:
ch = nextchar();
```

```
break;

case state of '.':
state = "99";
ch = nextchar();
break;
default:
state = "102";
f= true;
case state of
“99”;
case state of [0-9]:
state = "100";
ch = nextchar();
default:
f = true;
case state of "100":
case state of [0-9]:
ch = nextchar();
default:
state = "101";
f = true;
}
case state of "18"|"24"|"28"|"41"|"51"|"56"|"64"|"74"|"84"|"97": print("
        keyword");
"102": print(" int");
"101": print(" float");
"7"|"10": print("operator");
"1": print(" sep");
```

```
"2": print(" eos");
"3": print(" que tag");
"4": print(" ws ");
default:
print("invalid input");
ch := nextchar();
end case;
}
}
```

2.0.4 Implementation of lexer

Flex Program:

```
%{
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void extern yyerror(char*);
void InvalidToken();
int noline =0;
%}
Keyword "dollar"|"pound"|"euro"|"canadiandollar"|"rupiya"| “canadiandollar”|
“swissfran”|”thaibaht”|”chienseyaun”|”brazilianreal”
Op "na"|"ketla"
Digit [0-9]
Int {Digit}+
Float {Digit}+(.{Digit})
qm "?"
ws [ \t\r\v\f ]|" "
newline [\n]
eos "."
sep ","
%%
{Keyword} {printf("Valid Keyword - %s\n",yytext);}
```

```
{Op} {printf("Valid Operator - %s\n",yytext);}
{Int} {printf("Valid Integer - %s\n",yytext);}
{qm} {printf("Que tag - %s\n",yytext);}
{eos} {printf("eos - %s\n",yytext);}
{sep} {printf("sep - %s\n",yytext);}
[/]{1}[/]{1}[a-zA-Z0-9|" "]* printf(" single Line Comment ");
{ws} ;
{newline} ++noline;
. {InvalidToken();}
%%
int yywrap()
{
return 1;
}
int main()
{
yylex();
printf("number of Lines = %d \n",noline);
return 0;
}
void yyerror(char *s)
{
 fprintf(stderr,"\nError On Line %d : \n %s \n",noline,s);
 exit(0);
```

```
}
void InvalidToken()
{
 printf("Error On Line %d : \n Invalid Token %s \n",noline,yytext);
 exit(0);
}
```

2.0.5 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 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 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-->Advance 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**

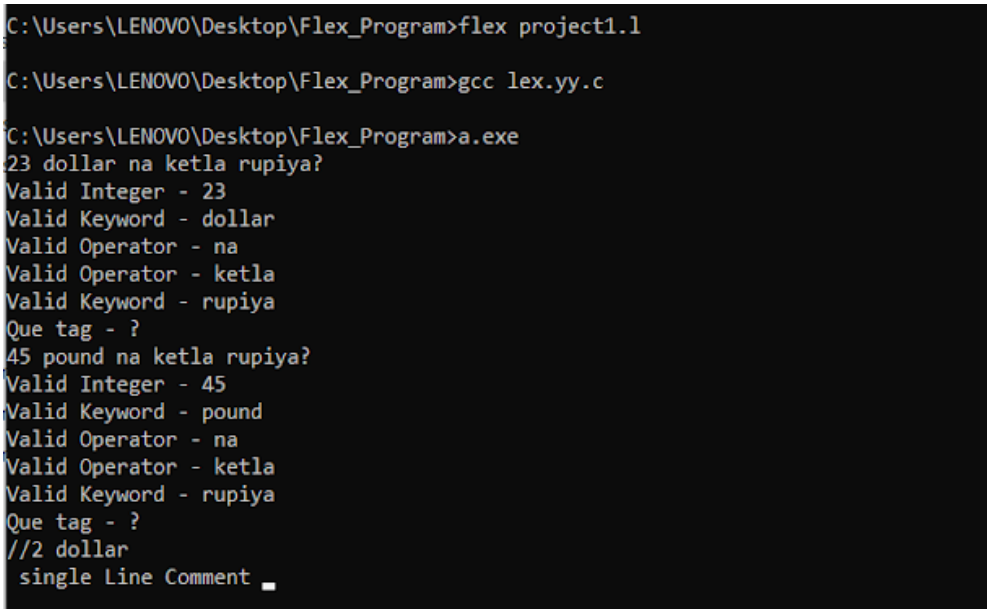
- Execute/Run for windows command promt :- **a.exe**

Step 6
- Finished

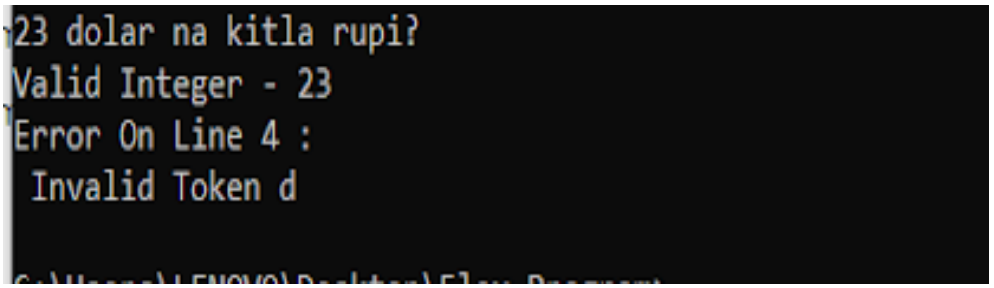
2.0.6 Output screenshots of lexer.

Input/Output:

1) Valid tokens:



2) Invalid tokens:



3.0 SYNTAX ANALYZER DESIGN

3.0.1 Grammar rules

S -> D

D -> A B F

F -> C K

K -> C E

A -> NUMBER WHITESPACE

B -> KEYWORD WHITESPACE

C -> OPERATION WHITESPACE

E -> KEYWORD QUESTION

KEYWORD -> rupiya | dollar | pound | swissfranc | euro | canadiandollar | thiabaht | chienceyaun | brazalianreal | iraqidinar

OPERATION -> na | ketla

WHITESPACE -> [\t] | [\n]

NUMBER -> int | float

QUESTION -> ?

3.0.2 Yacc based imlementation of syntax analyzer

- **ltproject.l (Lex file)**

```
%{
 #include<stdio.h>
 #include<stdlib.h>
 #include<string.h>
 #include "y.tab.h";
%}
Keyword
"rupiya"|"dollar"|"pound"|"swissfranc"|"euro"|"canadiandollar"|"thiabaht"|"chie
nceyaun"|"brazalianreal"|"iraqidinar"
Op      "na"|"ketla"
Digit   [0-9]
Int     {Digit}+
qm      "?"
ws      [ \t\n]
%%
{Keyword} {printf("Keyword - %s\n",yytext);return KEYWORD;}
{Op}      {printf("Operator - %s\n",yytext);return OPERATION;}
{Int}     {printf("Integer - %s\n",yytext);return NUMBER;}
{qm}      {return QUESTION;}
{ws}      {return WHITESPACE;}
.         {printf("Invalid Token : %s\n",yytext); return *yytext;}
%%

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

- **ltproject.y (yacc code)**

```
%{
#include<stdio.h>
#include<stdlib.h>
#define YYERROR_VERBOSE 1
void yyerror(char* err);
%}

%token KEYWORD OPERATION NUMBER WHITESPACE
QUESTION

%%
S : D { printf("\nThese Sentences are Valid. \n\n"); exit(0); }
;
D : A B F
;
F : C K
;
K : C E
;
A : NUMBER WHITESPACE
;
B : KEYWORD WHITESPACE
;
C : OPERATION WHITESPACE
;
E : KEYWORD QUESTION
;
%%

void yyerror(char *err) {
printf("Error: ");
fprintf(stderr, "%s\n", err);
```

```
exit(1);
}

int main() {
printf("Enter Sentences :\n");
yyparse();
}
```

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 and not in c:/program files(X86)/gnuwin32

Download IDE

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

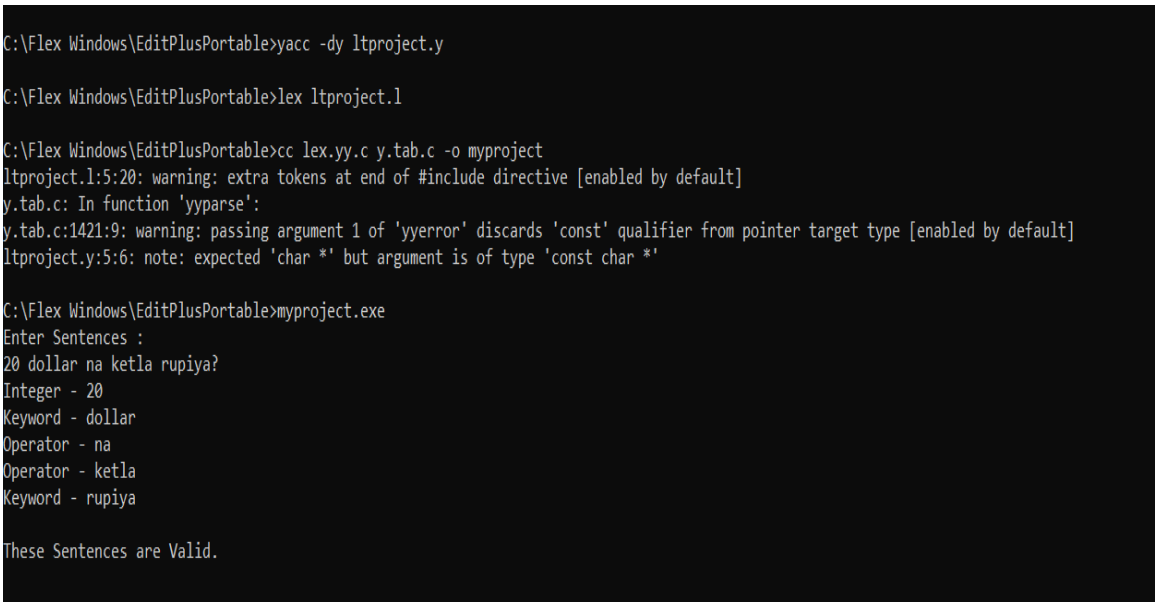
To run the program:

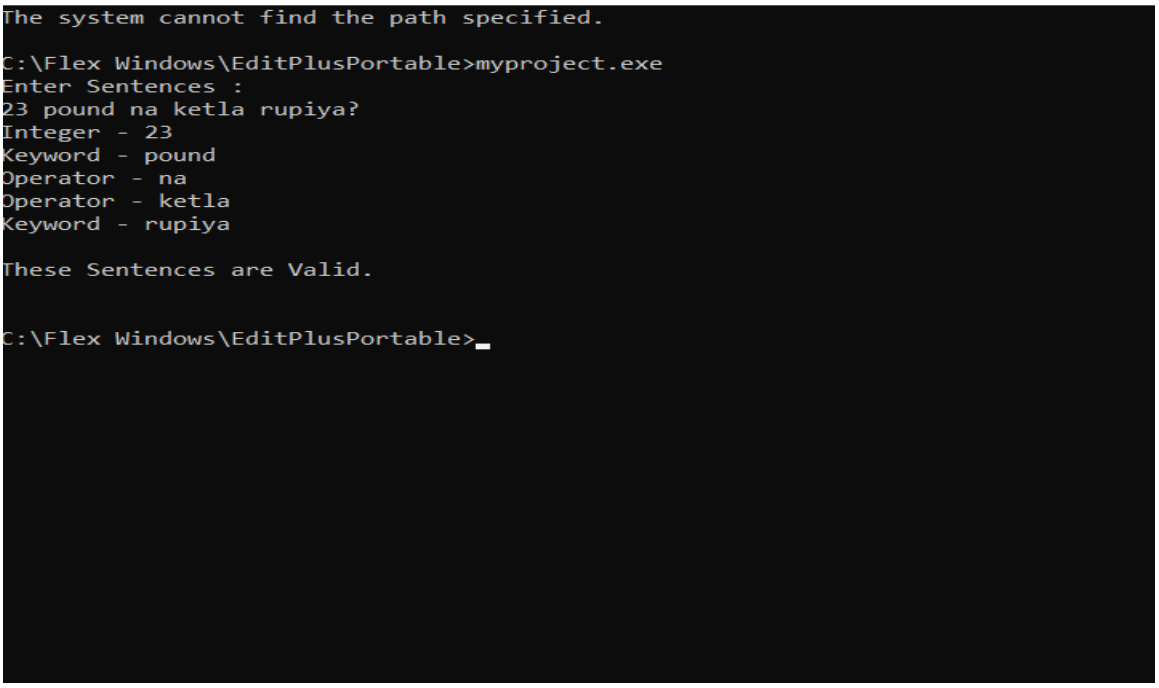
Open a prompt, cd to the directory where your ".l" and ".y" are, and compile them with:

```
flex  yacc.l
bison -dy yacc.y
gcc lex.yy.c y.tab.c -o yacc.exe
```

3.0.4 Output screenshots of yacc based implementation

- **Valid output: (sentences are valid)**





```
C:\Flex Windows\EditPlusPortable>myprojectt.exe
Enter Sentences :
67 canadiandollar na ketla rupiya?
Integer - 67
Keyword - canadiandollar
Operator - na
Operator - ketla
Keyword - rupiya

These Sentences are Valid.
```

```
C:\Flex Windows\EditPlusPortable>myprojectt.exe
Enter Sentences :
45 euro na ketla rupiya?
Integer - 45
Keyword - euro
Operator - na
Operator - ketla
Keyword - rupiya

These Sentences are Valid.
```

Invalid Syntax Output:

1. Program is not complete yet (expecting input after dot)

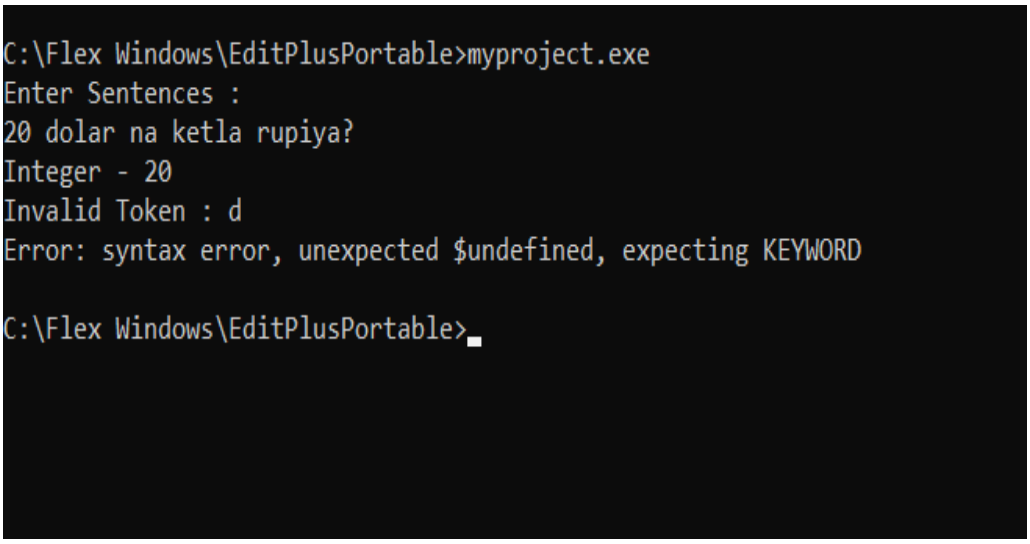
```
C:\Flex Windows\EditPlusPortable>myproject.exe
Enter Sentences :
23 dollar
Integer - 23
Keyword - dollar
Error: syntax error, unexpected $undefined, expecting OPERATOR
```

2. Question mark should be used to mark end of the sentence

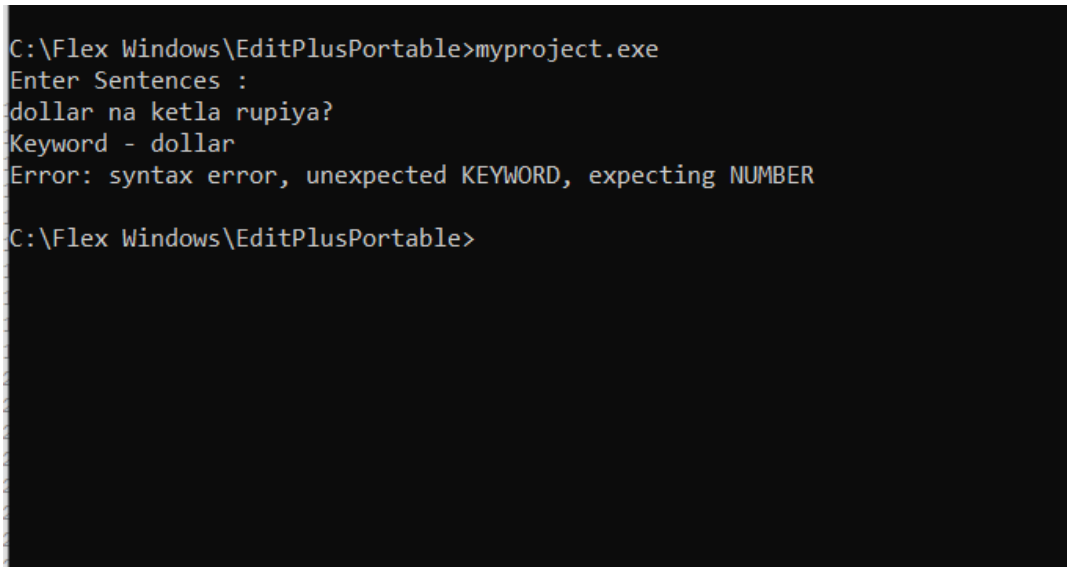
```
C:\Flex Windows\EditPlusPortable>myproject.exe
Enter Sentences :
20 dollar na ketla rupiya.
Integer - 20
Keyword - dollar
Operator - na
Operator - ketla
Keyword - rupiya
Invalid Token : .
Error: syntax error, unexpected $undefined, expecting QUESTION

C:\Flex Windows\EditPlusPortable>
```

3. Invalid token



4. Starting of the sentence must be number



5. Invalid Operator

```
C:\Flex Windows\EditPlusPortable>myprojectt.exe
Enter Sentences :
67 euro naa ketla rupiya?
Integer - 67
Keyword - euro
Operator - na
Invalid Token : a
Error: syntax error, unexpected $undefined, expecting WHITESPACE

C:\Flex Windows\EditPlusPortable>_
```

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 practical world and also how various types of errors are handled. Also we learned about which kind of phases the compliers have. We can generate compliers for our own layman free language.