Project Report
on
Compiler for <<String Operations Using Gujarati Language>>

Developed by

Manisha Yadav – IT144 –18ITUOS059

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 "**Compiler for String Operations using Gujarati language"** is a bonafied report of the work carried out by

1) Miss. Manisha Yadav, Student ID No: 18ITUOS004

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

1.0 INTRODUCTION

1.0.1 Project Details

Language Name: String Operations using Gujarati language

Language description:

Write an appropriate language description for a layman language which can do string operations using Gujarati sentences ,written in roman script .

Example of valid program in this language is -
2000 ,500 ,400 ane 23 noo sarvadoo kar. Ema thi 300 baad kar. Havee jawab ne 3 thi guni nakh. Pachi 4 thi bhagi nakh. Cheloo jawab shu chey?

1.0.2 Project Planning

List of Students with their Roles/Responsibilities:

IT144 MANISHA YADAV : Regular Expression , DFA Design, Algorithm Design and implmentation, Scanner phase Implementation, Grammar rules, YACC implementation, Final Report.

2.0 LEXICAL PHASE DESIGN

2.0.1 Regular Expression:

Keywords :

RE	Token
jawab	jawab
havee	havee
ne	ne
noo	noo
nakh	nakh
thi	thi
ane	ane
kar	kar
Ema	ema
Pachi	pachi
Cheloo	cheloo
chey	chey
shu	shu

Operations :

RE	Token	Attribute
Sarvadoo	op	pop
Guni	op	mop
Baad	op	sop
Bhagi	op	dop

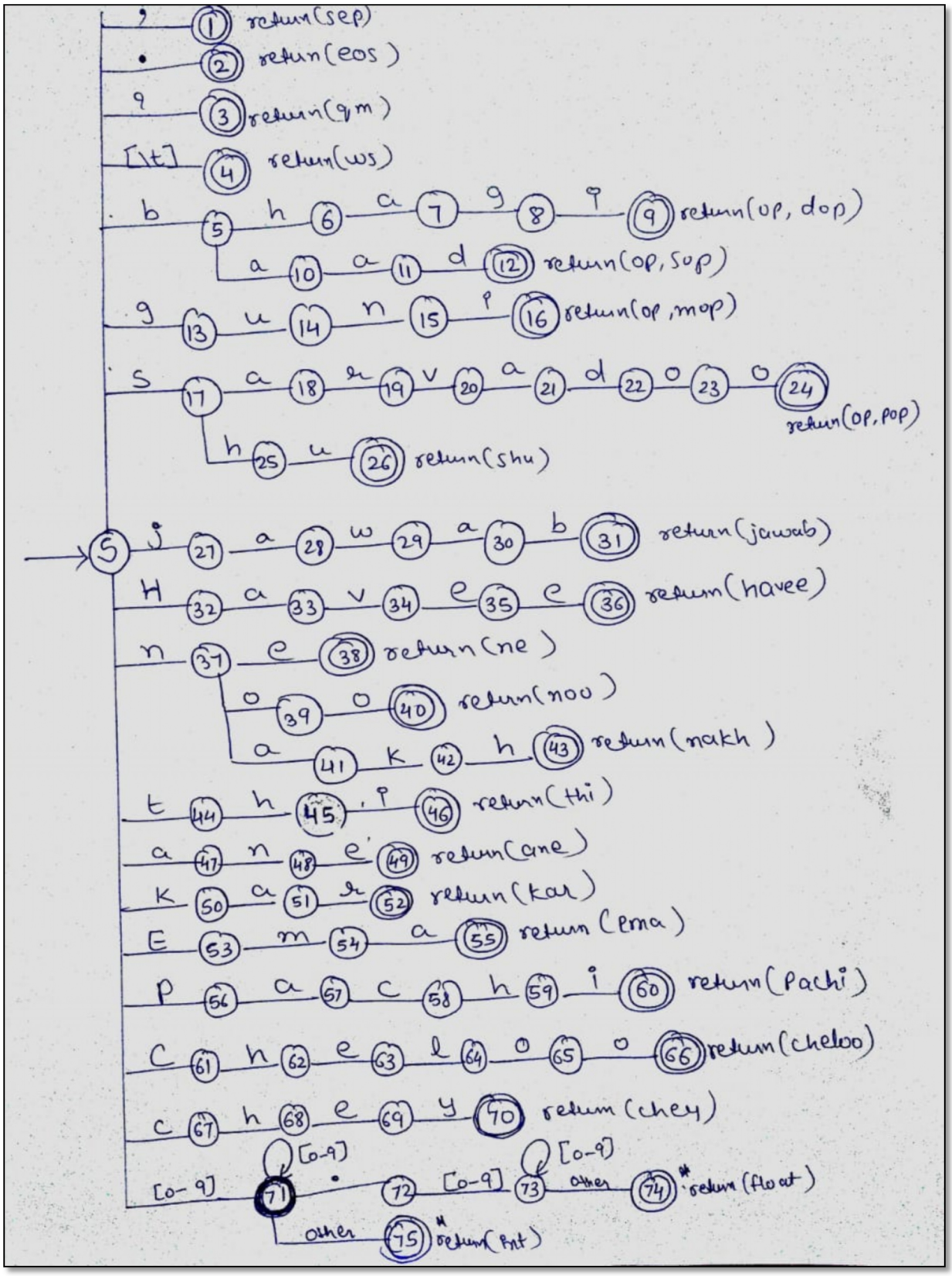
Values type : int and float

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

Delimiters : {. , ? \t}

RE	Token
.	eos
,	sep
?	qm
[\t]	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="S";
      while not eof do (c < len)
        {
         if (f)
        {
            f= false;
         }
        char  ch = nextchar();
         switch (state) {
            case state of
            "S":
               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 ]:
        state = "4";
        ch = nextchar();
        f= true;
           break;

‘b’:
        state = "5";
        ch = nextchar();
          break;

‘g’:
        state = "13";
        ch = nextchar();
          break;

‘s’:
        state = "17";
        ch = nextchar();
           break;

   ‘j’:
        state = "27";
        ch = nextchar();
          break;
   'H’:
        state = "32";
        ch = nextchar();
          break;
   'n':
        state = "37";
```

```
        ch = nextchar();
           break;
  ‘t’:
        state = "44";
        ch = nextchar();
          break;
‘a’:
        state = "47";
        ch = nextchar();
          break;
‘k’:
        state = "50";
        ch = nextchar();
          break;
‘E’:
        state = "53";
        ch = nextchar();
          break;
‘P’:
        state = "56";
        ch = nextchar();
          break;
‘C’:
        state = "61";
        ch = nextchar();
          break;
‘c’:
        state = "67";
        ch = nextchar();
          break;

[0-9]:
        state = "71";
        ch = nextchar();
          break;
```

```
                    }
                default:
                    f= true;
            end case

case state of "5":
                case state of 'h':
                    state = "6";
                    ch = nextchar();
                        break;
                case state of 'a':
                    state = "10";
                    ch = nextchar();
                        break;
                    default:
                    f = true;

case state of "6":
                case state of 'a':
                    state = "7";
                    ch = nextchar();

case state of "7":
                case state of 'g':
                    state = "8";
                    ch = nextchar();

case state of "8":
                case state of 'i':
                    state = "9";
                    ch = nextchar();
                    f= true;

case state of "10":
```

```
                case state of 'a':
                    state = "11";
                    ch = nextchar();

case state of "11":
                case state of 'd':
                    state = "12";
                    ch = nextchar();
                    default:
                    f= true;

case state of "13":
                case state of 'u':
                    state = "14";
                    ch = nextchar();

case state of "14":
                case state of 'n':
                    state = "15";
                    ch = nextchar();

case state of "15":
                case state of 'i':
                    state = "16";
                    ch = nextchar();
                    f= true;

case state of "17":
                case state of 'h':
                    state = "25";
                    ch = nextchar();
                          break;
                  case state of 'a':
                    state = "18";
                    ch = nextchar();
```

```
                    break;

        case state of "18":
                case state of 'r':
                    state = "19";
                    ch = nextchar();

case state of "19":
                case state of 'v':
                    state = "20";
                    ch = nextchar();

case state of "20":
                case state of 'a':
                    state = "21";
                    ch = nextchar();

case state of "21":
                case state of 'd':
                    state = "22";
                    ch = nextchar();

case state of "22":
                case state of 'o':
                    state = "23";
                    ch = nextchar();

case state of "23":
                case state of 'o':
                    state = "24";
                    ch = nextchar();
                        f = true;

case state of "25":
                case state of 'u':
```

```
                state = "26";
                ch = nextchar();
                    f= true;

case state of "27":
            case state of 'a':
                state = "28";
                ch = nextchar();

case state of "28":
            case state of 'w':
                state = "29";
                ch = nextchar();

case state of "29":
            case state of 'a':
                state = "30";
                ch = nextchar();

case state of "30":
            case state of 'b':
                state = "31";
                ch = nextchar();
                f= true;

case state of "32":
            case state of 'a':
                state = "33";
                ch = nextchar();

case state of "33":
            case state of 'v':
                state = "34";
                ch = nextchar();
```

```
case state of "34":
        case state of 'e':
            state = "35";
            ch = nextchar();

case state of "35":
        case state of 'e':
            state = "36";
            ch = nextchar();
            f = true;

case state of "37":
        case state of 'e':
            state = "38";
            ch = nextchar();
            f= true;
                break;
        case state of 'o':
            state = "39";
            ch = nextchar();
                break;
        case state of 'a':
            state = "41";
            ch = nextchar();
                break;

case state of "39":
        case state of 'o':
            state = "40";
            ch = nextchar();
            f = true;

case state of "41":
        case state of 'k':
            state = "2"4;
```

```
                ch = nextchar();

case state of "42":
            case state of 'h':
                state = "43";
                ch = nextchar();
                f = true;

case state of "44":
            case state of 'h':
                state = "45";
                ch = nextchar();

case state of "45":
            case state of 'i':
                state = "46";
                ch = nextchar();
                 f= true;

case state of "47":
            case state of 'n':
                state = "48";
                ch = nextchar();

case state of "48":
            case state of 'e':
                state = "49";
                ch = nextchar();
                 f = true;


case state of "50":
            case state of 'a':
                state = "51";
                ch = nextchar();
```

```
case state of "51":
        case state of 'r':
            state = "52";
            ch = nextchar();
             f = true;

case state of "53":
        case state of 'm':
            state = "54";
            ch = nextchar();

case state of "54":
        case state of 'a':
            state = "55";
            ch = nextchar();
             f = true;

case state of "56":
        case state of 'a':
            state = "57";
            ch = nextchar();

case state of "57":
        case state of 'c':
            state = "58";
            ch = nextchar();

case state of "58":
        case state of 'h':
            state = "59";
            ch = nextchar();

case state of "59":
        case state of 'i':
```

```
                state = "60";
                ch = nextchar();
                f = true;

case state of "61":
            case state of 'h':
                state = "62";
                ch = nextchar();

case state of "62":
            case state of 'e':
                state = "63";
                ch = nextchar();

case state of "63":
            case state of 'l':
                state = "64";
                ch = nextchar();

case state of "64":
            case state of 'o':
                state = "65";
                ch = nextchar();

case state of "65":
            case state of 'o':
                state = "66";
                ch = nextchar();
                f = true;


case state of "67":
            case state of 'h':
                state = "68";
                ch = nextchar();
```

```
case state of "68":
        case state of 'e':
            state = "69";
            ch = nextchar();

case state of "69":
        case state of 'y':
            state = "70";
            ch = nextchar();
             f = true;

case state of "71":
        case state of [0-9]:
            ch = nextchar();
                 break;
        case state of '.':
            state = "72";
            ch = nextchar();
                 break;
        default:
            state = "75";
            f= true;

 case state of "72":
        case state of [0-9]:
            state = "73";
            ch = nextchar();
        default:
           f = true;

 case state of "73":
        case state of [0-9]:
            ch = nextchar();
        default:
```

```
                    state = "74";
                    f = true;
            }
        }
            case state of
              "26"|"31"|”36"|"38"|"40"|"43"|"46"|"49"|"52"|"55"|"60"|"66"|"70":
                 print(" keyword");

              "75":
                 print(" int");

             "74":
                 print(" float");

              "9"|"12"|"16"|"24":
                 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>
%}

Keyword
"jawab"|"Havee"|"ne"|"thi"|"ane"|"noo"|"kar"|"Ema"|"nakh"|"Pachi"|"Cheloo
"|"shu"|"chey"

Op        "sarvadoo"|"baad"|"bhagi"|"guni"
Digit     [0-9]
Int       {Digit}+
Float     {Digit}+(.{Digit})
qm        "?"
ws        [\t]
eos       "."
sep       ","

%%

{Keyword}  {printf("Keyword - %s\n",yytext);}
{Op}         {printf("Operator  - %s\n",yytext);}
{Int}        {printf("Integer - %s\n",yytext);}
{Float}      {printf("Float Number - %s\n",yytext);}
{qm}         {printf("que tag - %s\n",yytext);}
{eos}        {printf("eos - %s\n",yytext);}
{sep}        {printf("sep - %s\n",yytext);}
{ws}         {printf("ws \n",yytext);}

%%
int yywrap(){return 1;}
int main()
{
   yylex();
   return 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:

```
C:\Users\Manisha\Desktop\flex programs>project.exe
2000 ,500 ,400 ane 23 noo sarvadoo kar. Ema thi 300 baad kar.
```

Output:

```
Integer          -      2000
Separator        -      ,
Integer          -      500
Separator        -      ,
Integer          -      400
Keyword          -      ane
Integer          -      23
Keyword          -      noo
Operator         -      sarvadoo
Keyword          -      kar
End of sentence  -      .
Keyword          -      Ema
Keyword          -      thi
Integer          -      300
Operator         -      baad
Keyword          -      kar
End of sentence  -      .
```

Input:

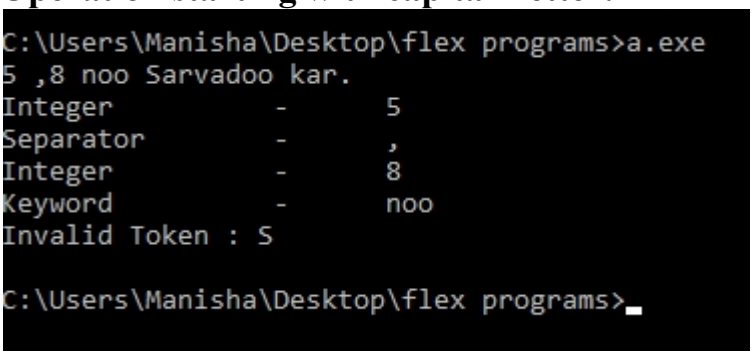
```
C:\Users\Manisha\Desktop\flex programs>project.exe
Havee 40 ne 5 thi guni nakh. Pachi jawab ne 4 thi bhagi nakh.Cheloo jawab shu chey?
```

Output:

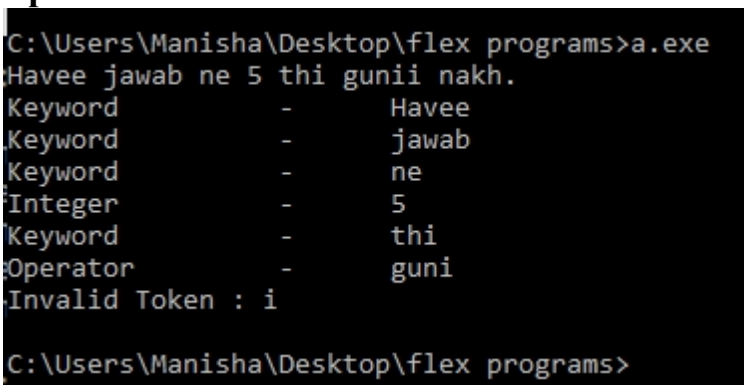
```
Keyword          -      Havee
Integer          -      40
Keyword          -      ne
Integer          -      5
Keyword          -      thi
Operator         -      guni
Keyword          -      nakh
End of sentence  -      .
Keyword          -      Pachi
Keyword          -      jawab
Keyword          -      ne
Integer          -      4
Keyword          -      thi
Operator         -      bhagi
Keyword          -      nakh
End of sentence  -      .
Keyword          -      Cheloo
Keyword          -      jawab
Keyword          -      shu
Keyword          -      chey
End of program   -      ?
```

Invalid tokens:

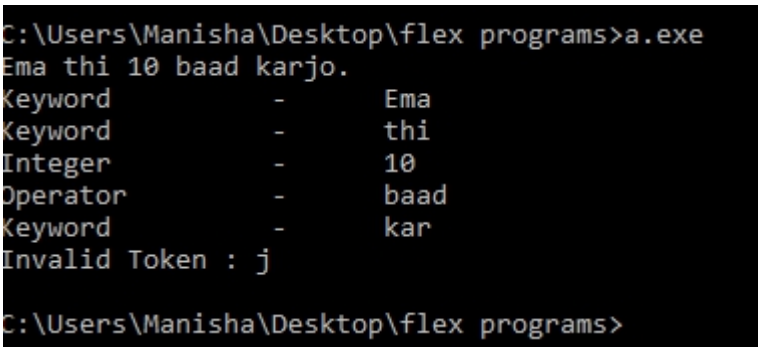
1. **Operation starting with capital Letter:**



2. **Operation is invalid:**



3. **Keyword is invalid:**



3.0 SYNTAX ANALYZER DESIGN

3.0.1 Grammar rules

S -> A | D

B -> OP KEYW DOT P

A -> KEYW A | KEYW C

C -> NUM B | NUM KEYW B

P -> KEYW P | QM | S

D -> NUM SEP D | NUM Q | NUM KEYW NUM Q

Q -> KEYW B

OP-> sarvadoo | baad | bhagi | guni

KEYW-> jawab | Havee | ne | thi | ane | noo | kar | Ema | nakh | Pachi | Cheloo | shu | chey

NUM-> int | float

QM-> **?**

DOT-> **.**

SEP-> **,**

3.0.2 Yacc based imlementation of syntax analyzer

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

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

%}

Keyword  "jawab"|"Havee"|"ne"|"thi"|"ane"|"noo"|"kar"|"Ema"|"nakh"|"Pachi"|
          "Cheloo"|"shu"|"chey"
Op        "sarvadoo"|"baad"|"bhagi"|"guni"
Digit     [0-9]
Int       {Digit}+
Float     {Digit}+(.{Digit})
qm        "?"
ws        [ \t\n]
eos       "."
sep       ","

%%

{Keyword}      {printf("Keyword      -      %s\n",yytext);return KEYWORD;}
{Op}           {printf("Operator     - %s\n",yytext);return OPERATION;}
{Int}          {printf("Integer      -      %s\n",yytext);return NUMBER;}
{Float}        {printf("Float        -      %s\n",yytext);return NUMBER;}
{qm}           {printf("End of program  -   %s\n",yytext);return QUEMARK;}
{eos}          {printf("End of sentence  -  %s\n",yytext);return DOT;}
{sep}          {printf("Separator      -      %s\n",yytext);return COMMA;}
{ws}           {return WHITESPACE;}
.              {printf("Invalid Token : %s\n",yytext); return 0;return *yytext;}

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

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

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

%token KEYWORD OPERATION NUMBER QUEMARK DOT COMMA
        WHITESPACE

%%
S : A
| D { printf("\nThese Sentences are Valid. \n\n"); return 0; }
;
B : OPERATION WHITESPACE KEYWORD DOT P
;
A : KEYWORD WHITESPACE A
| KEYWORD WHITESPACE C
;
C : NUMBER WHITESPACE B
| NUMBER WHITESPACE KEYWORD WHITESPACE B
;
P : KEYWORD WHITESPACE P
| QUEMARK
| S
;
D : NUMBER WHITESPACE COMMA D
| NUMBER WHITESPACE KEYWORD WHITESPACE NUMBER
        WHITESPACE Q
```

```
| NUMBER WHITESPACE Q
;
Q : KEYWORD WHITESPACE B
;
%%

void yyerror(char *err) {
printf("Error: ");
fprintf(stderr, "%s\n", err);
exit(1);
}
int main() {
printf("Enter Gujarati 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 Input with all the possible combinations:**

```
C:\Users\Manisha\Desktop\bison programs>project
Enter Gujarati Sentences:
200 ,800 ,75 ane 50 noo sarvadoo kar.Ema thi 400 baad kar.Cheloo jawab shu chey ?
Integer          -      200
Separator        -      ,
Integer          -      800
Separator        -      ,
Integer          -      75
Keyword          -      ane
Integer          -      50
Keyword          -      noo
Operator         -      sarvadoo
Keyword          -      kar
End of sentence  -      .
Keyword          -      Ema
Keyword          -      thi
Integer          -      400
Operator         -      baad
Keyword          -      kar
End of sentence  -      .
Keyword          -      Cheloo
Keyword          -      jawab
Keyword          -      shu
Keyword          -      chey
End of program   -      ?

These Sentences are Valid.


C:\Users\Manisha\Desktop\bison programs>
```

```
C:\Users\Manisha\Desktop\bison programs>project
Enter Gujarati Sentences:
89 ,40 noo sarvadoo kar.Pachi jawab ne 4 thi bhagi nakh.Cheloo jawab shu chey ?
Integer          -      89
Separator        -      ,
Integer          -      40
Keyword          -      noo
Operator         -      sarvadoo
Keyword          -      kar
End of sentence  -      .
Keyword          -      Pachi
Keyword          -      jawab
Keyword          -      ne
Integer          -      4
Keyword          -      thi
Operator         -      bhagi
Keyword          -      nakh
End of sentence  -      .
Keyword          -      Cheloo
Keyword          -      jawab
Keyword          -      shu
Keyword          -      chey
End of program   -      ?

These Sentences are Valid.


C:\Users\Manisha\Desktop\bison programs>
```

```
C:\Users\Manisha\Desktop\bison programs>project
Enter Gujarati Sentences:
900 ane 500 noo baad kar.Havee jawab ne 4 thi guni nakh.Cheloo jawab shu chey ?
Integer          -      900
Keyword          -      ane
Integer          -      500
Keyword          -      noo
Operator         -      baad
Keyword          -      kar
End of sentence  -      .
Keyword          -      Havee
Keyword          -      jawab
Keyword          -      ne
Integer          -      4
Keyword          -      thi
Operator         -      guni
Keyword          -      nakh
End of sentence  -      .
Keyword          -      Cheloo
Keyword          -      jawab
Keyword          -      shu
Keyword          -      chey
End of program   -      ?

These Sentences are Valid.


C:\Users\Manisha\Desktop\bison programs>
```

```
C:\Users\Manisha\Desktop\bison programs>project
Enter Gujarati Sentences:
800 ane 50.5 noo guni kar.Ema thi 200 baad kar.Havee jawab ane 250 noo sarvadoo kar.
Pachi jawab ne 5 thi bhagi nakh.Cheloo jawab shu chey ?
Integer          -      800
Keyword          -      ane
Float            -      50.5
Keyword          -      noo
Operator         -      guni
Keyword          -      kar
End of sentence  -      .
Keyword          -      Ema
Keyword          -      thi
Integer          -      200
Operator         -      baad
Keyword          -      kar
End of sentence  -      .
Keyword          -      Havee
Keyword          -      jawab
Keyword          -      ane
Integer          -      250
Keyword          -      noo
Operator         -      sarvadoo
Keyword          -      kar
End of sentence  -      .
Keyword          -      Pachi
Keyword          -      jawab
Keyword          -      ne
Integer          -      5
Keyword          -      thi
Operator         -      bhagi
Keyword          -      nakh
End of sentence  -      .
Keyword          -      Cheloo
Keyword          -      jawab
Keyword          -      shu
Keyword          -      chey
End of program   -      ?

These Sentences are Valid.
```

- **Invalid Syntax:**

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

```
C:\Users\Manisha\Desktop\bison programs>project
Enter Gujarati Sentences:
10 ane 20 noo guni kar.
Integer          -      10
Keyword          -      ane
Integer          -      20
Keyword          -      noo
Operator         -      guni
Keyword          -      kar
End of sentence  -      .
Error: syntax error, unexpected WHITESPACE, expecting KEYWORD or NUMBER or QUEMARK

C:\Users\Manisha\Desktop\bison programs>
```

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

```
C:\Users\Manisha\Desktop\bison programs>project
Enter Gujarati Sentences:
50 ,40 noo sarvadoo kar?
Integer          -      50
Separator        -      ,
Integer          -      40
Keyword          -      noo
Operator         -      sarvadoo
Keyword          -      kar
End of program   -      ?
Error: syntax error, unexpected QUEMARK, expecting DOT

C:\Users\Manisha\Desktop\bison programs>
```

3. Two operations are used consecutively in a sentence.

```
C:\Users\Manisha\Desktop\bison programs>project
Enter Gujarati Sentences:
8 ane 15 noo guni bhagi kar.
Integer          -      8
Keyword          -      ane
Integer          -      15
Keyword          -      noo
Operator         -      guni
Operator         -      bhagi
Error: syntax error, unexpected OPERATION, expecting KEYWORD

C:\Users\Manisha\Desktop\bison programs>
```

4. Missing another number or misplacement of a keyword

```
C:\Users\Manisha\Desktop\bison programs>project
Enter Gujarati Sentences:
50 ane noo sarvadoo kar.
Integer          -      50
Keyword          -      ane
Keyword          -      noo
Error: syntax error, unexpected KEYWORD, expecting OPERATION or NUMBER
```

5. should be used to indicate end of program ?

```
C:\Users\Manisha\Desktop\bison programs>project
Enter Gujarati Sentences:
7 ,5 noo sarvadoo kar.Cheloo jawab shu chey.
Integer          -      7
Separator        -      ,
Integer          -      5
Keyword          -      noo
Operator         -      sarvadoo
Keyword          -      kar
End of sentence  -      .
Keyword          -      Cheloo
Keyword          -      jawab
Keyword          -      shu
Keyword          -      chey
End of sentence  -      .
Error: syntax error, unexpected DOT, expecting QUEMARK
```

6. Invalid token

```
C:\Users\Manisha\Desktop\bison programs>project
Enter Gujarati Sentences:
40 ane 80 noo sarvadoo kar.Ema thi 20 baad karj.
Integer          -      40
Keyword          -      ane
Integer          -      80
Keyword          -      noo
Operator         -      sarvadoo
Keyword          -      kar
End of sentence  -      .
Keyword          -      Ema
Keyword          -      thi
Integer          -      20
Operator         -      baad
Keyword          -      kar
Invalid Token : j
Error: syntax error, unexpected $end

C:\Users\Manisha\Desktop\bison programs>
```

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.