

Project Report
on
**Compiler for
<< Time Conversion
Gujarati to English >>**

Developed by

Rohan Pansara - IT086 - 20ITUOS119

Deep Patel - IT094 - 20ITUBS141

Het Patel - IT104 - 20ITUOS130

Jenish Patel - IT105 - 20ITUBS039



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

**DHARMSINH DESAI UNIVERSITY
NADIAD-387001, GUJARAT**



CERTIFICATE

This is to certify that the project entitled “ **Time Conversion Gujarati to English**” is a bonafide report of the work carried out by

- 1) Mr. Rohan Pansara, Student ID No: 20ITUOS119
- 2) Mr. Deep Patel, Student ID No: 20ITUBS141
- 3) Mr. Het Patel, Student ID No: 20ITUOS130
- 4) Mr. Jenish Patel, Student ID No: 20ITUBS039

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 for the subject of “**Language Translator**” during the academic year 2022-2023.

Prof. Nilamba Vala,

(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	
1.1 Project Details (Specification /Explanation of the project)	4
1.2 Project Planning (List of students involved and their roles / responsibilities)	4
2.0. Lexical phase design	
2.1 Regular Expressions	5
2.2 Deterministic Finite Automaton design for lexer	7
2.3 Algorithm of lexer	8
2.4 Implementation of lexer	23
2.5 Execution environment setup...	25
2.6 Output screenshots of lexer	27
3.0 Syntax analyzer design	
3.1 Grammar rules	29
3.2 Yacc based implementation of syntax analyzer	30
3.3 Execution environment setup	33
3.4 Output screenshots of yacc based implementation	34
4.0 Conclusion and future work...	38

Chapter 1- Introduction

1.1 Project Details:

Language Name: Time Conversation Gujarati to English

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 are:-

1) savar na pona 4

2) sanj na pona 3

3) bopor na dodh

Project Planning

List of students with their Responsibilities:

IT086 ROHAN PANSARA: Regular Expression, DFA Design, Algorithm Design and implementation, Scanner phase Implementation, Grammar rules, YACC implementation, Final Report.

IT094 DEEP PATEL: Regular Expression, DFA Design, Algorithm Design and implementation, Scanner phase Implementation, Grammar rules, YACC implementation, Final Report.

IT104 HET PATEL: Regular Expression, DFA Design, Algorithm Design and implementation, Scanner phase Implementation, Grammar rules, YACC implementation, Final Report.

IT105 JENISH PATEL: Regular Expression, DFA Design, Algorithm Design and implementation, Scanner phase Implementation, Grammar rules, YACC implementation, Final Report.

Chapter 2- Lexical Phase Design

2.1 Regular Expression:

Keywords :

RE-Token

savar - savar
bopor – bopor
sanj - sanj
raat - raat
na- na
sava – sava
sada - sada
pona – pona
dodh – dodh
adhi – adhi

Operations :

Values type : int and float

RE Token

[0-9]+ int

[10-12]+ int

Delimiters : {\n \t}

RE Token

[\n] new line

[\t] ws

2.2 DFA Design for LEXER:

Project

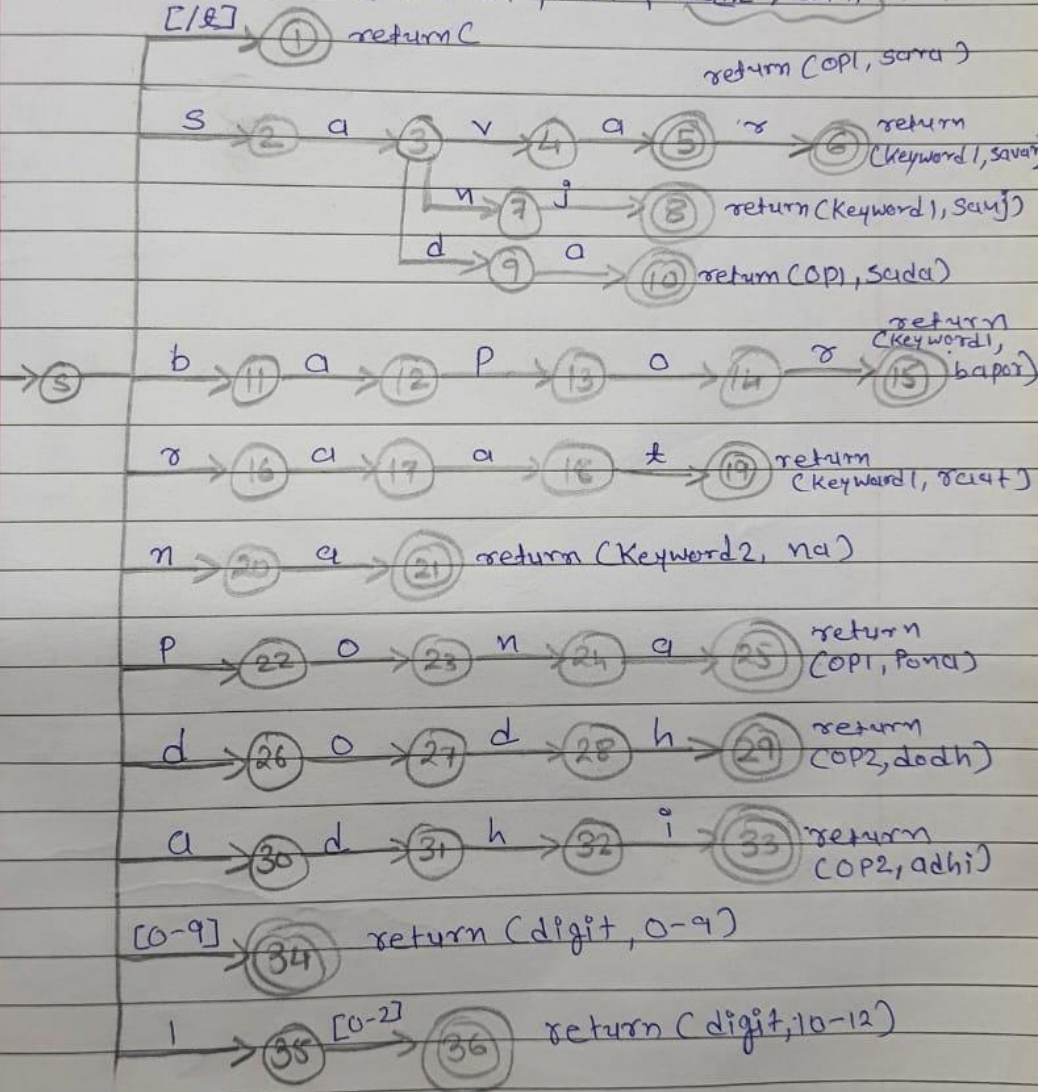
PAGE NO.:

DATE: / /

⇒ Aim & Designing Compiler & Language For Time Conversion Gujarati to English.

⇒ DFA

Keyword 1: Sarar, bapor, sayj, saat, na
 Operation & Keyword 2: sarar, suda, ponar, dath, adhi
 [1-9] return C



2.3 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":

'[\t]':
state = "1";
ch = nextchar();
f = true;
break;

's':
state = "2";
ch = nextchar();
break;

'b':
state = "11";
ch = nextchar();
break;

'r':
state = "16";
ch = nextchar();
break;

'n':
state = "20";
ch = nextchar();
break;

'p':
state = "22";
ch = nextchar();
break;

```



```
'd':  
state = "26";  
ch = nextchar();  
break;
```

```
'a':  
state = "30";  
ch = nextchar();  
break;
```

```
'l':  
state = "35";  
ch = nextchar();  
break;
```

```
'[1-9]':  
state = "34";  
ch = nextchar();  
f = true;  
break;  
}
```

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

```
case state of "2":  
case state of 'a':  
state = "3";  
ch = nextchar();  
break;
```

```
case state of "3":  
case state of 'v':  
state = "4";  
ch = nextchar();  
break;
```

```
case state of "4":  
case state of 'a':  
state = "5";  
ch = nextchar();  
f = true;  
break;
```

```
case state of "5":  
case state of 'r':  
state = "6";  
ch = nextchar();  
f = true;  
break;
```

```
case state of "3":  
case state of 'n':  
state = "7";  
ch = nextchar();  
break;
```

```
case state of "7":  
case state of 'j':  
state = "8";  
ch = nextchar();  
f=true;  
break;
```

```
case state of "3":  
case state of 'd':  
state = "9";  
ch = nextchar();  
break;
```

```
case state of "9":  
case state of 'a':  
state = "10";  
ch = nextchar();  
f=true;  
break;
```

```
case state of "11":  
case state of 'a':  
state = "12";  
ch = nextchar();  
break;
```

```
case state of "12":  
case state of 'p':  
state = "13";  
ch = nextchar();  
break;
```

```
case state of "13":  
case state of 'o':  
state = "14";  
ch = nextchar();  
break;
```

```
case state of "14":  
case state of 'r':  
state = "15";  
ch = nextchar();  
f=true;  
break;
```

```
case state of "16":  
case state of 'a':  
state = "17";  
ch = nextchar();  
break;
```

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

```
case state of "18":  
case state of 't':  
state = "19";  
ch = nextchar();  
f=true;  
break;
```

```
case state of "20":  
case state of 'a':  
state = "21";  
ch = nextchar();  
f=true;  
break;
```

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

```
case state of "23":  
case state of 'n':  
state = "24";  
ch = nextchar();  
break;
```

```
case state of "24":  
case state of 'a':  
state = "25";  
ch = nextchar();  
break;
```

```
case state of "26":  
case state of 'o':  
state = "27";  
ch = nextchar();  
break;
```

```

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

```

```

case state of "28":
case state of 'h':
state = "29";
ch = nextchar();
f=true;
break;

```

```

case state of "30":
case state of 'd':
state = "31";
ch = nextchar();
break;

```

```

case state of "31":
case state of 'h':
state = "32";
ch = nextchar();
break;

```

```

case state of "32":
case state of 'i':
state = "33";
f=true;
ch = nextchar();
break;

```

```

case state of "35":
case state of [0-2]:
state = "36";
ch = nextchar();
f=true;
break;

```

```

}
}

```

```

case state of "1":print("white space");
case state of "6"|"8"|"15"|"19":print("keyword1");
case state of "21" : print("keyword2");
case state of "5"|"6" : print("operator1");
case state of "29"|"33" : print("operator2");
case state of "34" : print("digit 0-9");
case state of "36" : print("digit 10-12");

```

2.4 Implementation of LEXER:**Flex Program:**

```

% {

#include<stdio.h>

% }

Digit [0-9]+

Keyword "savar"|"bopor"|"sanj"|"raat"

keyw1 "na"

Operation "sava"|"sada"|"pona"

Ope1 "dodh"|"adhi"

%%

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

{ keyw1 } { printf("Keyword1: %s\n",yytext);}

{ Operation } { printf("Operation: %s\n",yytext);}

{ Ope1 } { printf("Operation1: %s\n",yytext);}

{ Digit } { printf("Digit: %s\n",yytext);}

.\|n  {}

%%

int yywrap(){

return 0;

}

```

```
int main()
{
    extern FILE *yyin,*yyout;

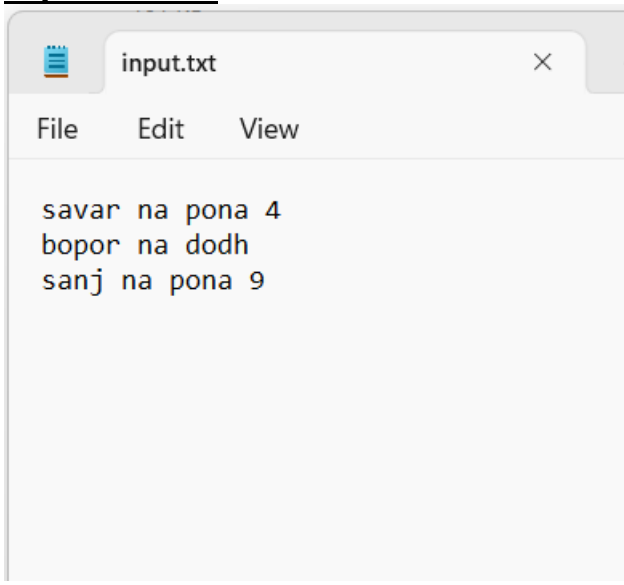
    yyin=fopen("input.txt","r");
    yyout=fopen("output.txt","w");

    yylex();

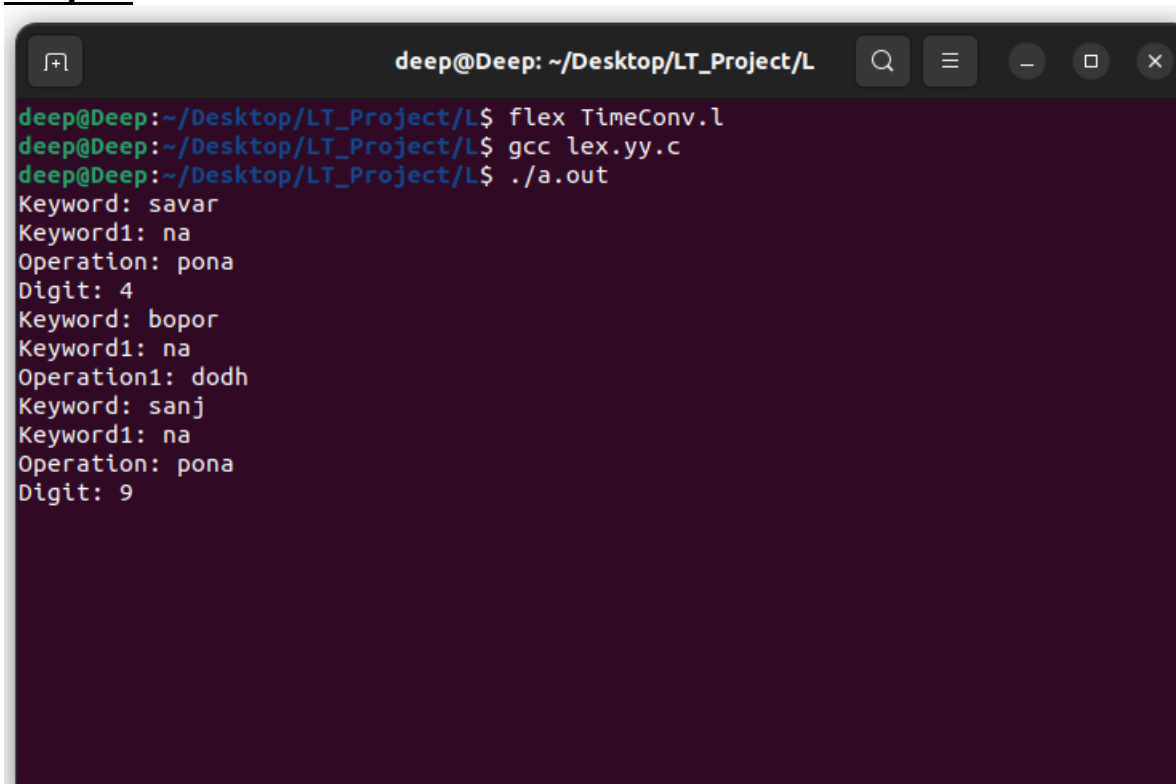
    return 0;
}
```

Output screenshots of lexer :

Input.txt file:



Output:



2.5 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-->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**

Step 6:

- Finished

Chapter 3- Syntax Analyzer Design**3.1 Grammar Rules:**

S-> A | B | C

A-> K K1 OP dg eol

B-> K K1 dg eol

C-> K K1 OP1 eol

K-> sava | bopar | sanj |raat

K1-> na

OP-> sava | sada | pona

OP1-> dodh | adhi

dg-> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12

3.2YACC based implementation of Syntax Analyzer:

Project.1 :

```
% {

/* Definition section */

#include<stdio.h>

#include "time.tab.h"

% }


/* Rule Section */

%%

[0-9]+ {printf("Digit: %s\n",yytext); return dg;}

"savar "|"bopor "|"sanj "|"raat "      {printf("Keyword: %s\n",yytext); return k;}

"na " {printf("Keyword1: %s\n",yytext); return k1;}

"sava "|"sada "|"pona "      {printf("Operation: %s\n",yytext); return op;}

"dodh "|"adhi" {printf("Operation1: %s\n",yytext); return op1;}

\n {return eol;}

.      ;

%%

int yywrap()

{

return 1;

}
```

Project.v:

```

% {

/* Definition section */

#include<stdio.h>

#include<stdlib.h>

% }

%token dg k k1 op op1 eol

/* Rule Section */

%%

S: A {printf("\nThis sentence is valid.\n"); return 0;};

|B {printf("\nThis sentence is valid.\n"); return 0;};

|C {printf("\nThis sentence is valid.\n"); return 0;}; ;

A: k k1 op dg eol ;

B: k k1 dg eol ;

C: k k1 op1 eol ;

%%

void yyerror()

{

printf("Error: Invalid sentence");

exit(1);

}

//driver code

void main()

{

printf("Enter sentence: ");

yyparse();

}

```

3.2 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:

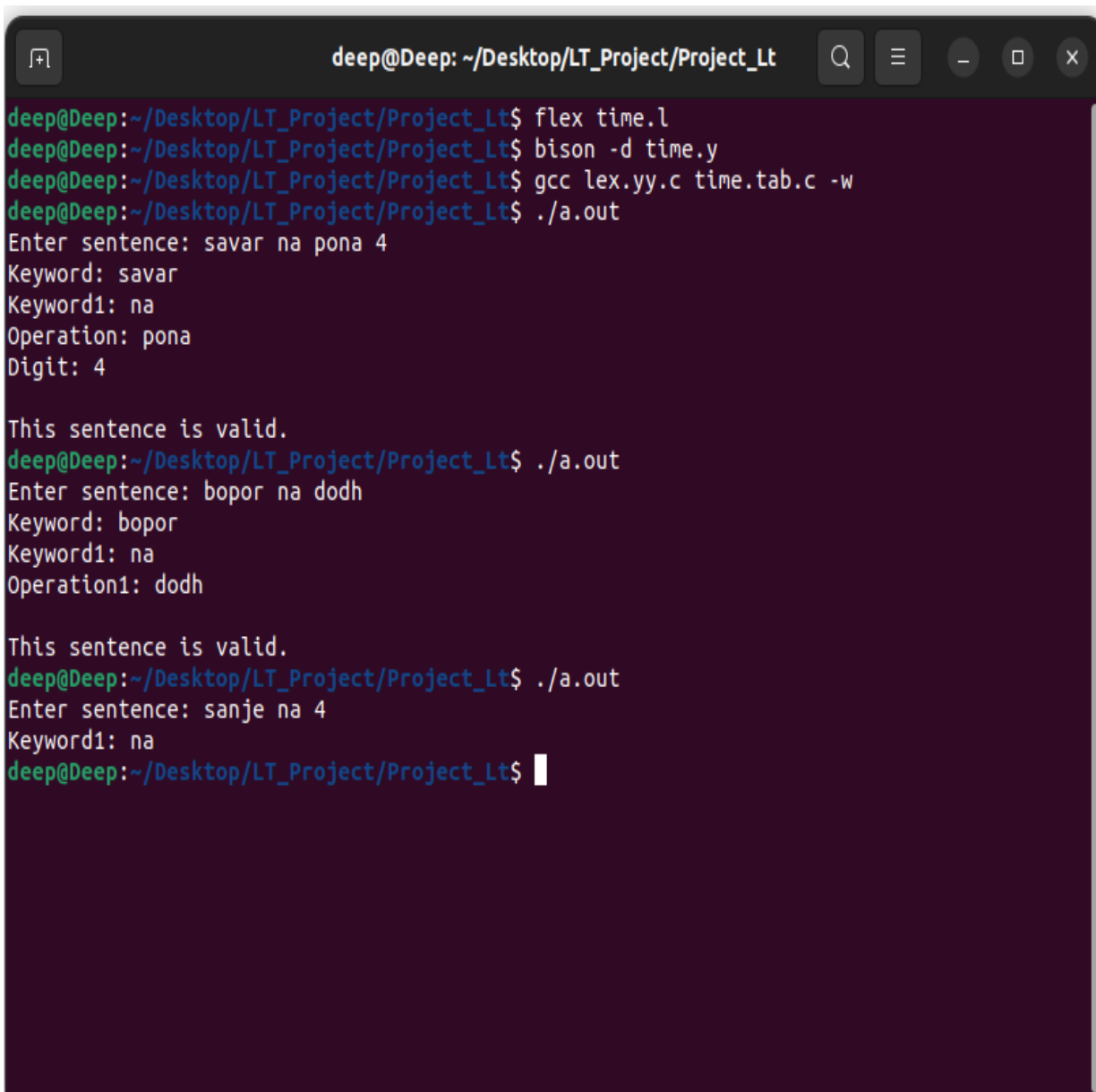
Open a prompt, cd to the directory where your ".l" and ".y" are, and compile them with:

flex time.l

bison -dy time.y

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

3.3 Output Screenshots of YACC based Implementation



```
deep@Deep: ~/Desktop/LT_Project/Project_Lt
deep@Deep:~/Desktop/LT_Project/Project_Lt$ flex time.l
deep@Deep:~/Desktop/LT_Project/Project_Lt$ bison -d time.y
deep@Deep:~/Desktop/LT_Project/Project_Lt$ gcc lex.yy.c time.tab.c -w
deep@Deep:~/Desktop/LT_Project/Project_Lt$ ./a.out
Enter sentence: savar na pona 4
Keyword: savar
Keyword1: na
Operation: pona
Digit: 4

This sentence is valid.
deep@Deep:~/Desktop/LT_Project/Project_Lt$ ./a.out
Enter sentence: bopor na dodh
Keyword: bopor
Keyword1: na
Operation1: dodh

This sentence is valid.
deep@Deep:~/Desktop/LT_Project/Project_Lt$ ./a.out
Enter sentence: sanje na 4
Keyword1: na
deep@Deep:~/Desktop/LT_Project/Project_Lt$
```

C-Program:

```

#include<stdio.h>
#include<string.h>
void main()
{
    FILE* file;
    int i=0,j=0,k=0,l;
    char ch;
    char buff[10][100];
    file = fopen("input.txt","r");
    if(file == NULL)
    {
        printf("Can't access file.\n");
        return;
    }
    while (ch != EOF) {
        ch = fgetc(file);
        if(ch == '\n')
        {
            buff[i][j++] = '\0';
            i++;
            j=0;
        }
        else
            buff[i][j++] = ch;
    }
    for(k=0;k<=i;k++)
    {
        printf("Input: %s\n",buff[k]);
        char* s = strtok(buff[k]," ");
        while(s != NULL)
        {
            if((strcmp(s,"savar")==0) || (strcmp(s,"bopor")==0) || (strcmp(s,"sanj")==0) ||
            (strcmp(s,"raat")==0))
                printf("Keyword: %s\n",s);
            else if(strcmp(s,"na")==0)
                printf("Keyword1: %s\n",s);
            else if((strcmp(s,"sava")==0) || (strcmp(s,"sada")==0) || (strcmp(s,"pona")==0))

```

```

        printf("Operation: %s\n",s);
    else if((strcmp(s,"dodh")==0) || (strcmp(s,"adhi")==0))
        printf("Operation1: %s\n",s);
    else if(isdigit(s)==0)
        printf("Digit: %s\n",s);
    s = strtok(NULL," ");
}
printf("\n");
}
fclose(file);
}

```

Output:

```

C:\Users\Dell\OneDrive\Desktop
Input: savar na pona 4
Keyword: savar
Keyword1: na
Operation: pona
Digit: 4

Input: bopor na dodh
Keyword: bopor
Keyword1: na
Operation1: dodh

Input: sanj na pona 9
Keyword: sanj
Keyword1: na
Operation: pona
Digit: 9

-----
Process exited after 2.033 seconds with return value 0
Press any key to continue . . .

```


Chapter 4- 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 soaked up more knowledge about how different compilers work in the practical world and also how various types of errors are handled.

Link:

https://github.com/Deep1511/LT_PROJECT.git