Project Report on

Compiler for String Operations Using Gujarati Language

Developed by

Aum Rathod-IT001-20ITUBS017

Rushi Bhalodiya-IT012-20ITUOS067

Ravi Bhuva-IT015-20ECUOG078

Department of Information Technology
Faculty of Technology, Dharmsinh Desai University
College Road, Nadiad-387001
2022-2023

DHARMSINH DESAI UNIVERSITY
NADIAD-387001, GUJARAT

CERTIFICATE

This is to certify that the project entitled “ **Compiler for String Operations and Mathemetical Expression using Gujarati language**” is a bonafied report of the work carried out by

1) Aum Rathod, Student ID No: 20ITUBS017
2) Rushi Bhalodiya, Student ID No: 20ITUOS067
3) Ravi Bhauva, Student ID No: 20ECUOG078

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 2022-2023.

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

Date: 17/03/2023

Index

Chatper1- Introduction

1.1 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 are:-

(1)
```
chhapi mar 10
jo(25>12) to 25+1 baki 12
jo(25<12) to 25+1 baki 12%5
jo(23>24) to 234
jo(23<=24) to 234
jo(2>1 ane 5>4) to 23
jo(2>1 kato 5<4) to 23
```

(2)
```
chhapi mar halo
jo(32>=23 ane 5!= 4) to chhapi mar moj baki chhapi mar dukh
jo(5==2+3) to chhapi mar barabar
chhapi mar 5*2+7%2
chhapi mar (5+2)*12
```

1.2 Project Planning

List of students with their Responsibilities:

IT001 Aum Rathod: Scanner phase Implementation, YACC implementation

IT012 Rushi Bhalodiya: Regular Expression,
Grammar rules, Final Report.

IT015 Ravi Bhuva: DFA Design,
Algorithm Design and implementation

Chatper2- Lexical Phase Design

2.1 Regular Expression:

Keywords : RE-Token

Madad
chhapi mar
jo
to
baki
chalo madya
ane
kato

Operations :

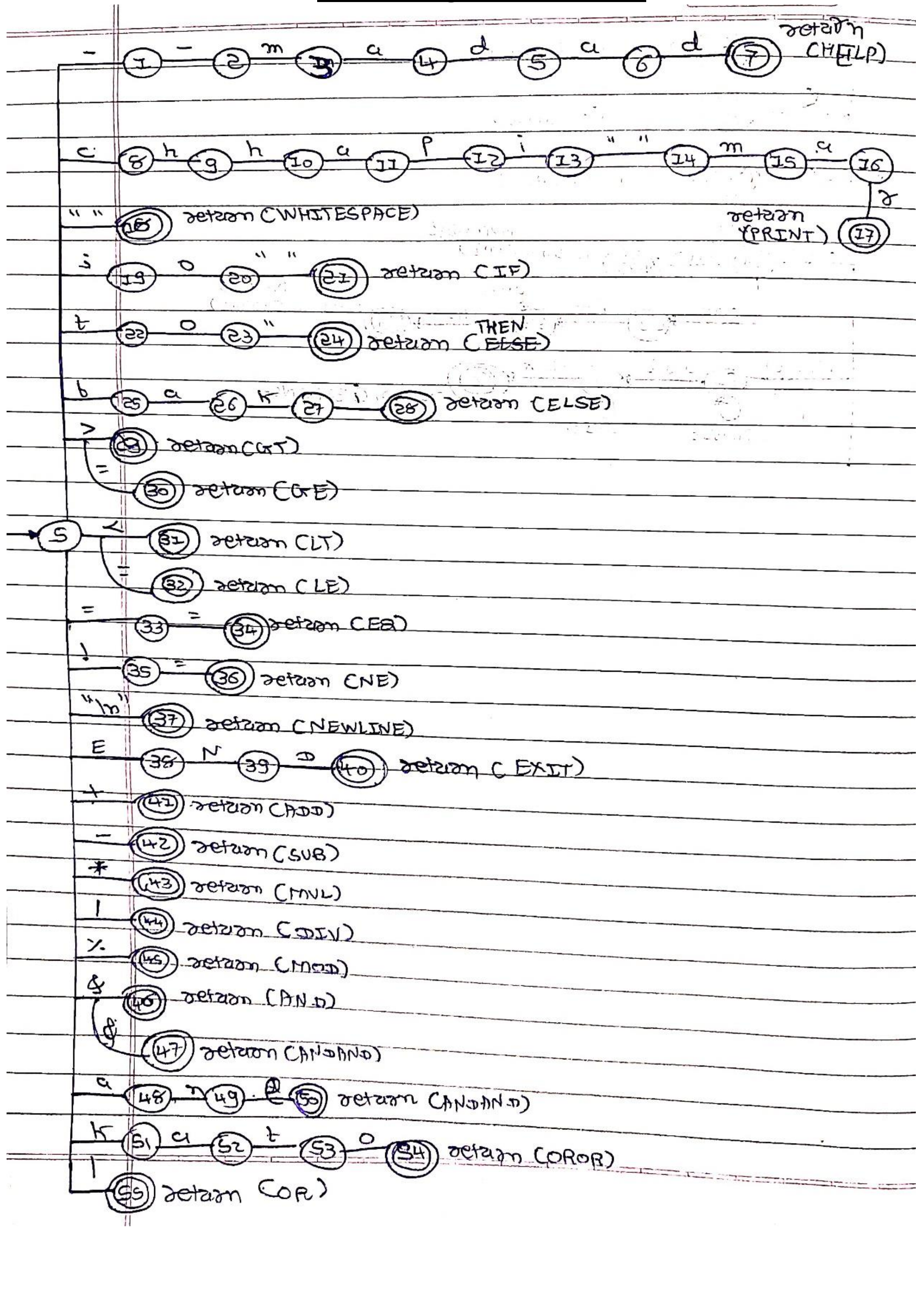
Value Type:int Token and String Token

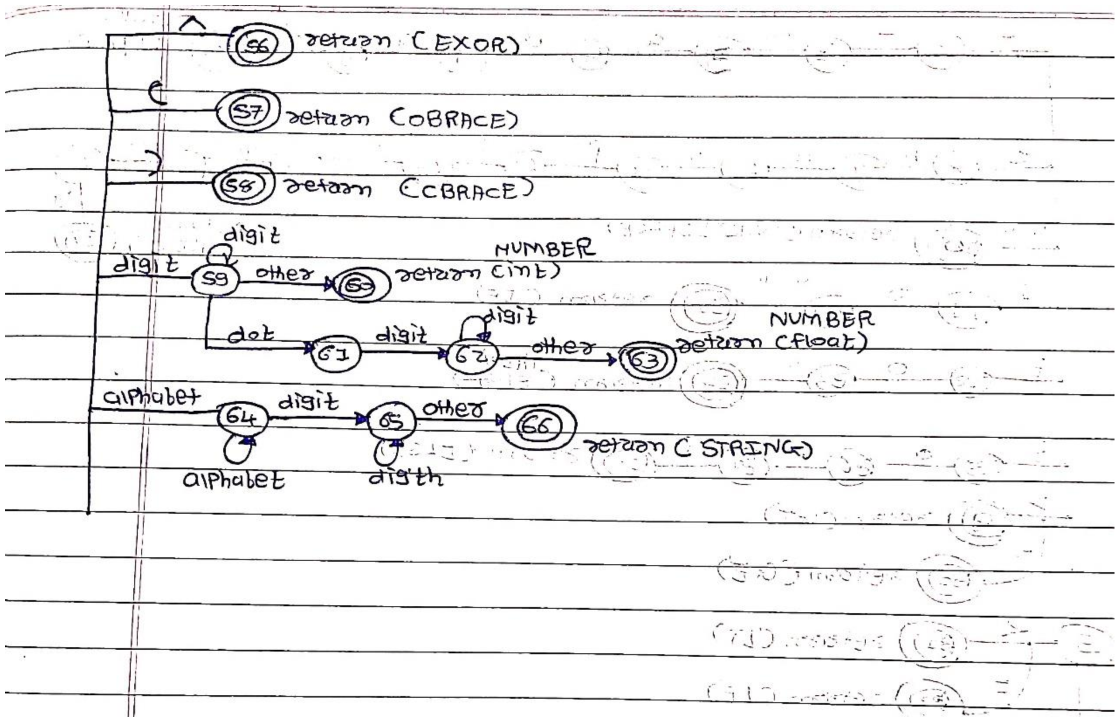
[0-9]+	int
[A-Za-z0-9]*	string

Delimiters:

"("	obrace
")"	cbrace
“ ”	whitespace

2.2 DFA Design for LEXER:




^
S6
return (EXOR)
(
S7
return (OBRACE)
)
S8
return (CBRACE)
digit
digit
S9
other
60
NUMBER
return (int)
dot
61
digit
62
digit
other
63
NUMBER
return (float)
alphabet
64
digit
65
other
66
return (STRING)
alphabet
digth

2.3 Implementation of LEXER:

Flex Program:

```
%{
/* Definition section */
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}

HELP "--madad"
PRINT "chhapi mar"
WHITESPACE " "
IF "jo"
THEN "to"
ELSE "baki"
GT  ">"
LT  "<"
EQ  "=="
GE  ">="
LE  "<="
NE  "!="
NEWLINE "\n"
EXIT "chalo madya"
ADD "+"
SUB "-"
MUL "*"
DIV "/"
MOD "%"
AND "&"
ANDAND  "ane"
OROR    "kato"
OR  "|"
EXOR "^"
OBRACE "("
CBRACE ")"
NUMBER [0-9]+
STRING [A-Za-z0-9]*
/* Rule Section */
%%
```

```
{HELP} {
    printf("--madad : keyword\n");
    return HELP;
}

{ADD} {
    printf("+ : arithmetic operator\n");
    return ADD;
}

{SUB} {
    printf("- : arithmetic operator\n");
    return SUB;
}

{MUL} {
    printf("* : arithmetic operator\n");
    return MUL;
}

{DIV} {
    printf("/ : arithmetic operator\n");
    return DIV;
}

{MOD} {
    printf("% : arithmetic operator\n");
    return MOD;
}

{AND} {
    printf("AND : logical Operator\n");
    return AND;
}

{OR} {
    printf("OR : logical Operator\n");
    return OR;
}

{ANDAND} {
    printf("ane : logical Operator\n");
    return ANDAND;
}

{OROR} {
    printf("kato : logical Operator\n");
    return OROR;
}

{EXOR} {
    printf("EXOR : logical Operator\n");
    return EXOR;
}

{PRINT} {
    printf("chhapi mar : keyword\n");
    return PRINT;
}

{WHITESPACE} {
    return WHITESPACE;
}

{NEWLINE} {
    return NEWLINE;
}

{EXIT} {
    return EXIT;
}

{IF} {
    printf("jo : keyword\n");
    return IF;
}

{THEN} {
    printf("to : keyword\n");
    return THEN;
}

{ELSE} {
    printf("baki : keyword\n");
    return ELSE;
}

{GT}    {
    printf("> : comparison operator\n");
    return GT;
}

{LT}    {
    printf("< : comparison operator\n");
    return LT;
}

{EQ}    {
    printf("= : comparison operator\n");
    return EQ;
}

{GE}    {
    printf(">= : comparison operator\n");
    return GE;
}

{LE}    {
    printf("<= : comparison operator\n");
    return LE;
}

{NE}    {
    printf("!= : comparison operator\n");
    return NE;
}

{OBRACE} {
    printf("( : punctuation\n");
    return OBRACE;
}

{CBRACE} {
    printf(") : punctuation\n");
    return CBRACE;
}
```

```
{NUMBER} {
    printf("%s : number\n",yytext);
    yylval=atoi(yytext);
    return NUMBER;
}
[\t] ;

{STRING} {
    printf("%s : string\n",yytext);
    yylval=yytext;
    return STRING;
}

. return yytext[0];

%%

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

Output

Screenshots of Flex Output:-

1. Chhapi mar 10

   Jo (25>12) to 25+1 baki 12

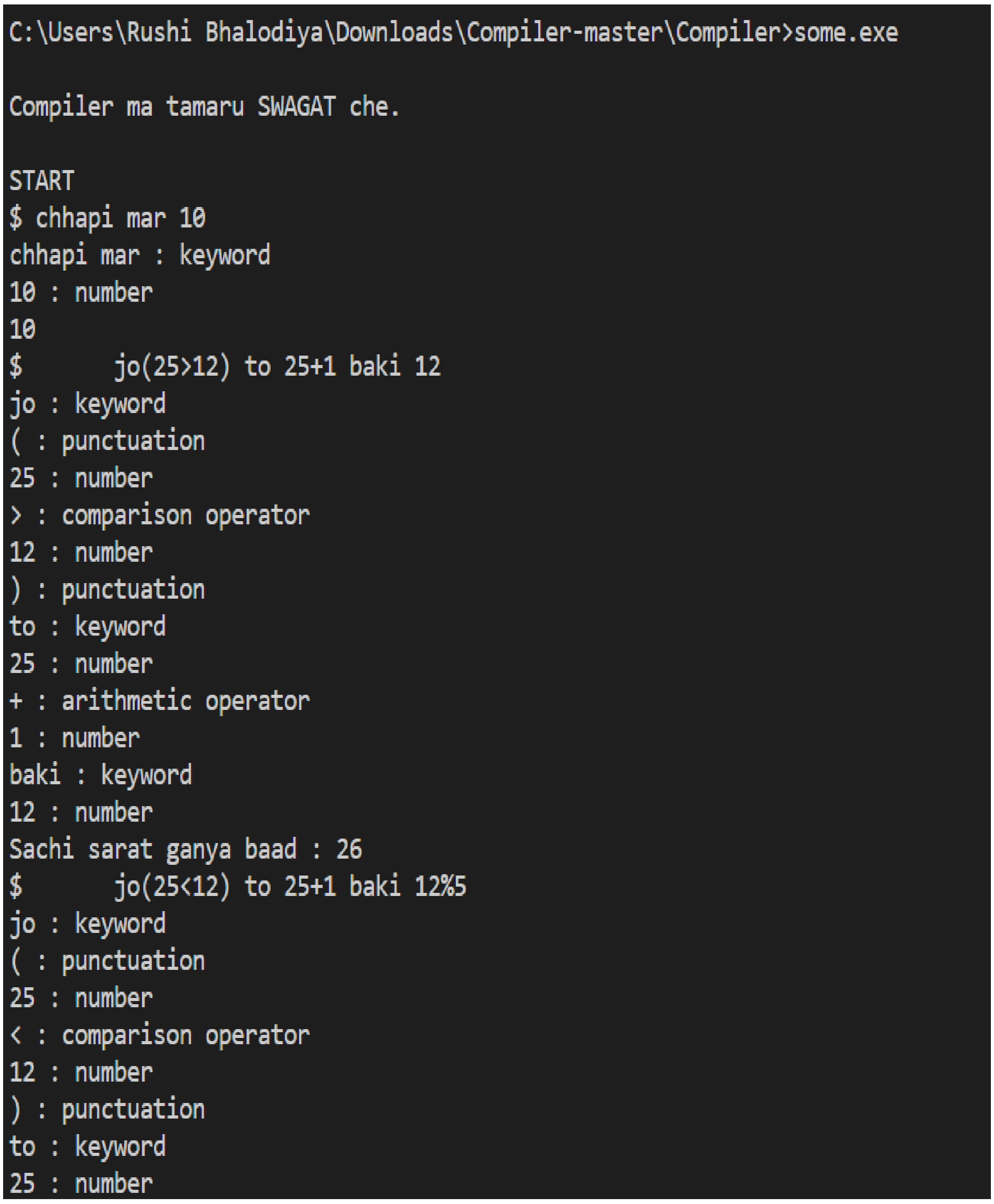
   Jo (25<12) to 25+1 baki 12%5

   Jo (24>23) to 234

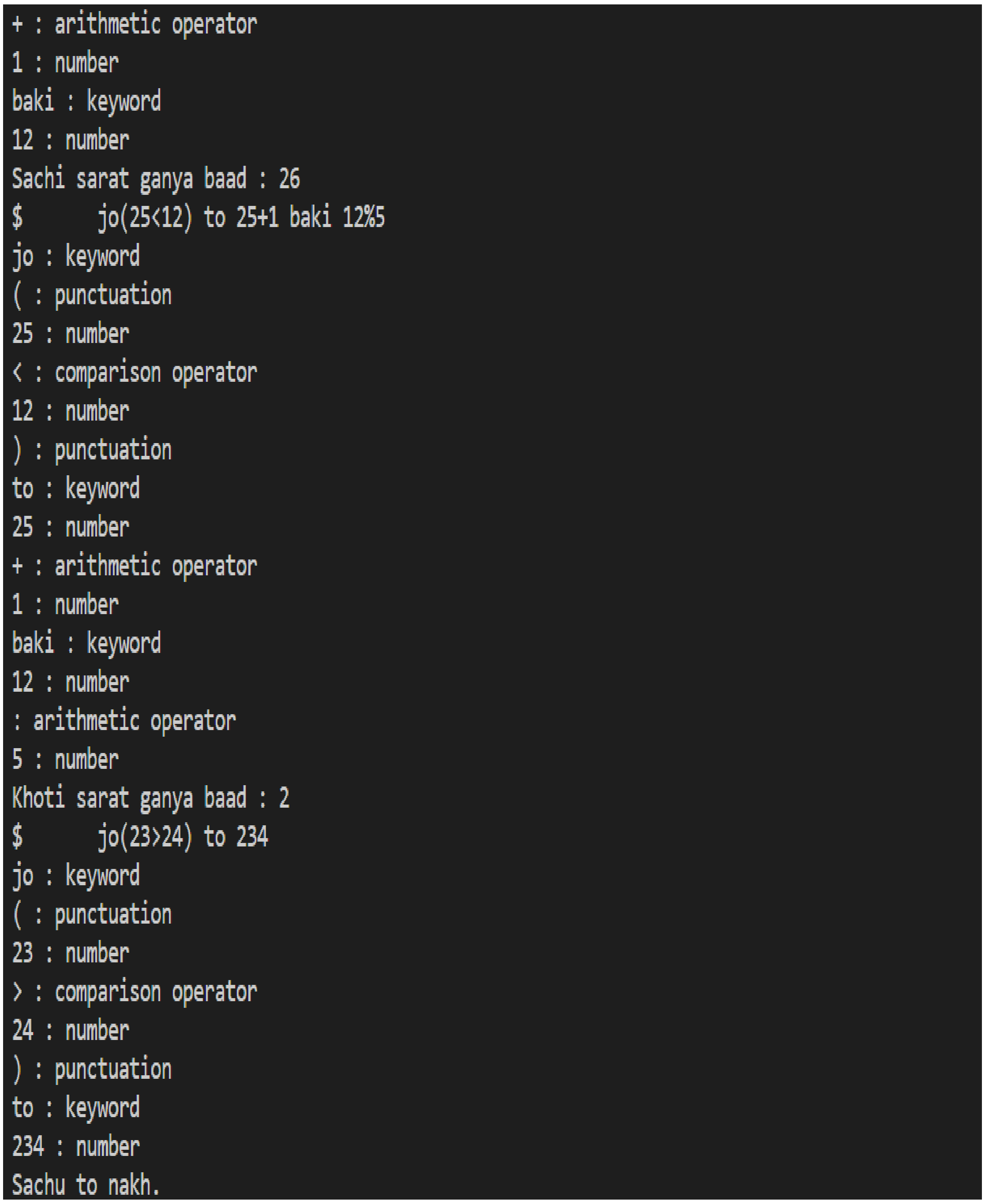
   Jo (23<=24) to 234

   Jo(2>1 ane 5>4) to 23

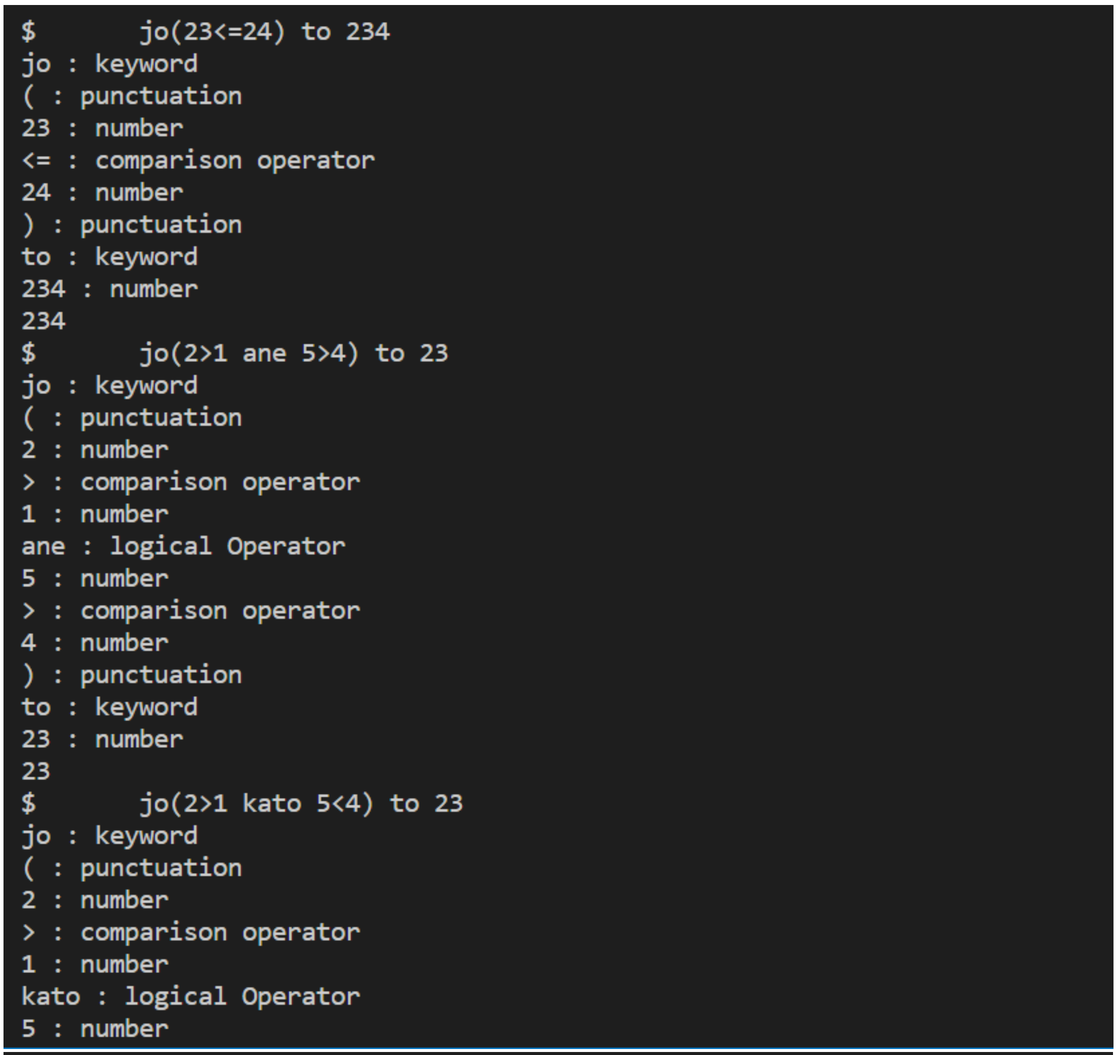
   Jo(2>1 kato 5<4) to 23



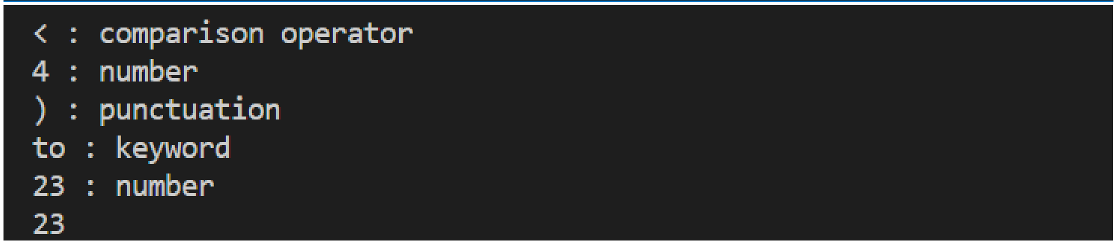
```
C:\Users\Rushi Bhalodiya\Downloads\Compiler-master\Compiler>some.exe

Compiler ma tamaru SWAGAT che.

START
$ chhapi mar 10
chhapi mar : keyword
10 : number
10
$       jo(25>12) to 25+1 baki 12
jo : keyword
( : punctuation
25 : number
> : comparison operator
12 : number
) : punctuation
to : keyword
25 : number
+ : arithmetic operator
1 : number
baki : keyword
12 : number
Sachi sarat ganya baad : 26
$       jo(25<12) to 25+1 baki 12%5
jo : keyword
( : punctuation
25 : number
< : comparison operator
12 : number
) : punctuation
to : keyword
25 : number
```



```
+ : arithmetic operator
1 : number
baki : keyword
12 : number
Sachi sarat ganya baad : 26
$       jo(25<12) to 25+1 baki 12%5
jo : keyword
( : punctuation
25 : number
< : comparison operator
12 : number
) : punctuation
to : keyword
25 : number
+ : arithmetic operator
1 : number
baki : keyword
12 : number
: arithmetic operator
5 : number
Khoti sarat ganya baad : 2
$       jo(23>24) to 234
jo : keyword
( : punctuation
23 : number
> : comparison operator
24 : number
) : punctuation
to : keyword
234 : number
Sachu to nakh.
```



```
$       jo(23<=24) to 234
jo : keyword
( : punctuation
23 : number
<= : comparison operator
24 : number
) : punctuation
to : keyword
234 : number
234
$       jo(2>1 ane 5>4) to 23
jo : keyword
( : punctuation
2 : number
> : comparison operator
1 : number
ane : logical Operator
5 : number
> : comparison operator
4 : number
) : punctuation
to : keyword
23 : number
23
$       jo(2>1 kato 5<4) to 23
jo : keyword
( : punctuation
2 : number
> : comparison operator
1 : number
kato : logical Operator
5 : number
```



```
< : comparison operator
4 : number
) : punctuation
to : keyword
23 : number
23
```

2.4 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!

<u>Step 3</u>: /*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*/**

<u>Step 4:</u>

- 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*/**

<u>Step 5</u>: /*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**

<u>Step 6:</u>

- Finished

Chapter3 – Synatx analyzer design

3.1 Grammar Rules:

```
Start: {printf("$ ");} Expression1 ;


Expression1-> k7 S
            | K7 N
            | K1 C K2 E
            | K1 C K2 E K3 E
            | E
            ;
K1-> “jo”
K2-> “to”
K3-> “baki”
K4-> “chalo madya”
K5-> “ane”
K6-> “kato”
K7-> “chhapi mar”
S -> String
N -> Number
E  -> Expression1
C -> Condition
```

3.2 YACC based implementation of Syntax Analyzer:

Flex.l:-

```
%{
/* Definition section */
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}

HELP "--madad"
PRINT "chhapi mar"
WHITESPACE " "
IF "jo"
THEN "to"
ELSE "baki"
GT  ">"
LT  "<"
EQ  "=="
GE  ">="
LE  "<="
NE  "!="
NEWLINE "\n"
EXIT "chalo madya"
ADD "+"
SUB "-"
MUL "*"
DIV "/"
MOD "%"
AND "&"
ANDAND  "ane"
OROR    "kato"
OR  "|"
EXOR "^"
OBRACE "("
CBRACE ")"
NUMBER [0-9]+
STRING [A-Za-z0-9]*
/* Rule Section */
%%
```

```
{HELP} {
    printf("--madad : keyword\n");
    return HELP;
}

{ADD} {
    printf("+ : arithmetic operator\n");
    return ADD;
}

{SUB} {
    printf("- : arithmetic operator\n");
    return SUB;
}

{MUL} {
    printf("* : arithmetic operator\n");
    return MUL;
}

{DIV} {
    printf("/ : arithmetic operator\n");
    return DIV;
}

{MOD} {
    printf("% : arithmetic operator\n");
    return MOD;
}

{AND} {
    printf("AND : logical Operator\n");
    return AND;
}

{OR} {
    printf("OR : logical Operator\n");
    return OR;
}

{ANDAND} {
    printf("ane : logical Operator\n");
    return ANDAND;
}

{OROR} {
    printf("kato : logical Operator\n");
    return OROR;
}

{EXOR} {
    printf("EXOR : logical Operator\n");
    return EXOR;
}

{PRINT} {
    printf("chhapi mar : keyword\n");
    return PRINT;
}

{WHITESPACE} {
    return WHITESPACE;
}

{NEWLINE} {
    return NEWLINE;
}

{EXIT} {
    return EXIT;
}

{IF} {
    printf("jo : keyword\n");
    return IF;
}

{THEN} {
    printf("to : keyword\n");
    return THEN;
}

{ELSE} {
    printf("baki : keyword\n");
    return ELSE;
}
```

```
{GT}    {
    printf("> : comparison operator\n");
    return GT;
}

{LT}    {
    printf("< : comparison operator\n");
    return LT;
}

{EQ}    {
    printf("= : comparison operator\n");
    return EQ;
}

{GE}    {
    printf(">= : comparison operator\n");
    return GE;
}

{LE}    {
    printf("<= : comparison operator\n");
    return LE;
}

{NE}    {
    printf("!= : comparison operator\n");
    return NE;
}

{OBRACE} {
    printf("( : punctuation\n");
    return OBRACE;
}

{CBRACE} {
    printf(") : punctuation\n");
    return CBRACE;
}

{NUMBER} {
    printf("%s : number\n",yytext);
    yylval=atoi(yytext);
    return NUMBER;
}
[\t] ;

{STRING} {
    printf("%s : string\n",yytext);
    yylval=yytext;
    return STRING;
}

. return yytext[0];

%%

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

yacc.y:

```
%{
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int flag=0;
%}

%token NUMBER PRINT WHITESPACE NEWLINE EXIT HELP STRING OBRACE CBRACE
%token IF THEN ELSE GT LT EQ GE LE NE AND OR EXOR ADD SUB MUL DIV MOD
ANDAND OROR
%left ADD SUB
%left MUL DIV MOD
%left  OBRACE CBRACE
%%

COMMAND: {printf("$ ");} Expression1 ;

Expression1: errorPrint
            | exit
            | printString
            | printNumber
            | help
            | ifCondition
            | ifElseCondition
            | E
            ;

errorPrint: STRING NEWLINE
        {
            printf("Amanya aadesh.Madad mate '--madad' vapro.\n\n");
        } COMMAND;

exit: EXIT NEWLINE
        {
            printf("\nPacha avjo.\n");
            exit(0);
        };

printString: PRINT WHITESPACES STRING NEWLINE
                                        {
                                            printf("%s",$3);
                                        } COMMAND;

printNumber: PRINT WHITESPACES E NEWLINE
                                        {
                                            printf("%d\n", $3);
                                        } COMMAND;

WHITESPACES: WHITESPACE WHITESPACES | ;

ifCondition: IF OBRACE CONDITION CBRACE WHITESPACE THEN WHITESPACE
Expression1 NEWLINE{
    if($3)
    {
        printf("%d\n",$8);
    }
    else
    {
        printf("Sachu to nakh.\n");
    }
} COMMAND;
```

```
ifElseCondition: IF OBRACE CONDITION CBRACE WHITESPACE THEN WHITESPACE
Expression1 WHITESPACE ELSE WHITESPACE Expression1 NEWLINE {
    if($3)
    {
        printf("Sachi sarat ganya baad : %d\n",$8);
    }
    else
    {
        printf("Khoti sarat ganya baad : %d\n",$12);
    }
} COMMAND;

help: HELP NEWLINE
                {
                    printf("chhapva mate\t:\tchhapi mar string | chhapi
mar expression\n");
                    printf("expression\t:\tnum+num || num-num || num*num
|| num/num || num%cnum || num&num || num|num || num^num ||
(Expression)\n",'%');
                    printf("jo\t\t:\tjo(condition) to expression\n");
                    printf("jo-to-baki\t:\tjo(condition) to expression
baki expression\n");
                    printf("madad\t\t:\t--madad\n");
                    printf("bandh krva mate\t:\tEND\n");
                } COMMAND;

E:E ADD E {$$=$1+$3;}
|E SUB E {$$=$1-$3;}
|E MUL E {$$=$1*$3;}
|E DIV E {$$=$1/$3;}
|E MOD E {$$=$1%$3;}
|OBRACE E CBRACE {$$=$2;}
|E AND E {$$=$1&$3;}
|E OR E {$$=$1|$3;}
|E EXOR E {$$=$1^$3;}
| NUMBER {$$=$1;}
;

CONDITION:E GT E {$$ = $1>$3;}
|    E LT E {$$ = $1<$3;}
|    E EQ E {$$ = ($1==$3);}
|    E GE E {$$ = ($1>=$3);}
|    E LE E {$$ = ($1<=$3);}
|   E NE E {$$ = ($1!=$3);}
|   CONDITION WHITESPACE ANDAND WHITESPACE CONDITION {$$ = ($1&&$3);}
|   CONDITION WHITESPACE OROR WHITESPACE CONDITION {$$ = ($1||$3);}
|   E {$$=$1;}
;

%%

void main()
{
    printf("\nCompiler ma tamaru SWAGAT che.\n\nSTART\n");
    yyparse();
    getch();
}

void yyerror()
{
   yyparse();
}
```

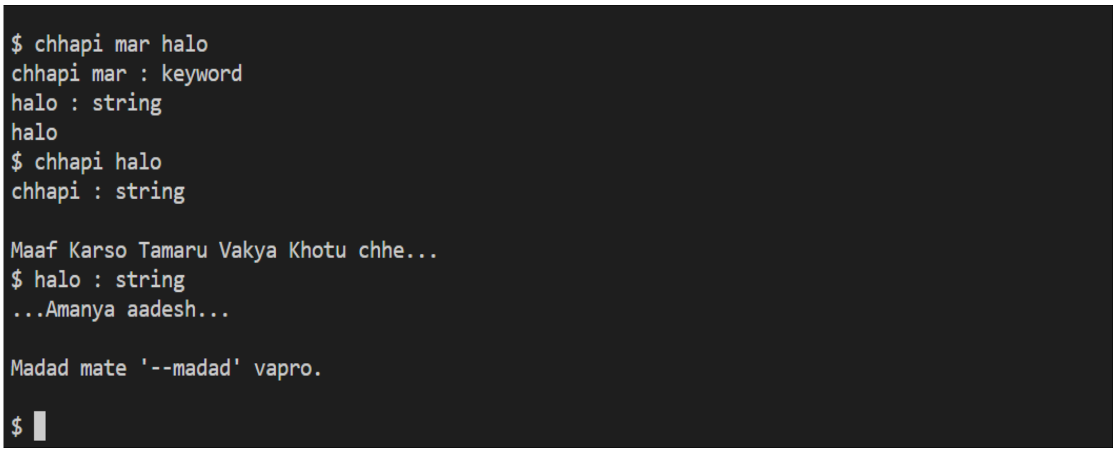
3.3 Execution Environment Setup:

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 lex.l
bison -dy yacc.y
gcc -w lex.yy.c y.tab.c -o some.exe

3.4 Output Screenshots of YACC based Implementation:

(1) Chhapi mar halo

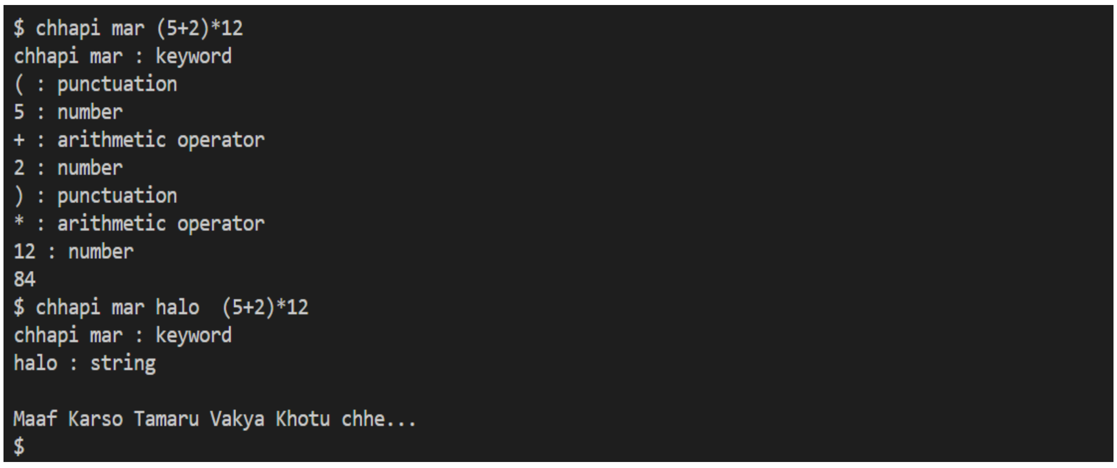
-Valid and Invalid output:-



```
$ chhapi mar halo
chhapi mar : keyword
halo : string
halo
$ chhapi halo
chhapi : string

Maaf Karso Tamaru Vakya Khotu chhe...
$ halo : string
...Amanya aadesh...

Madad mate '--madad' vapro.

$ 
```

(2) Chhapi mar (5+2)*7

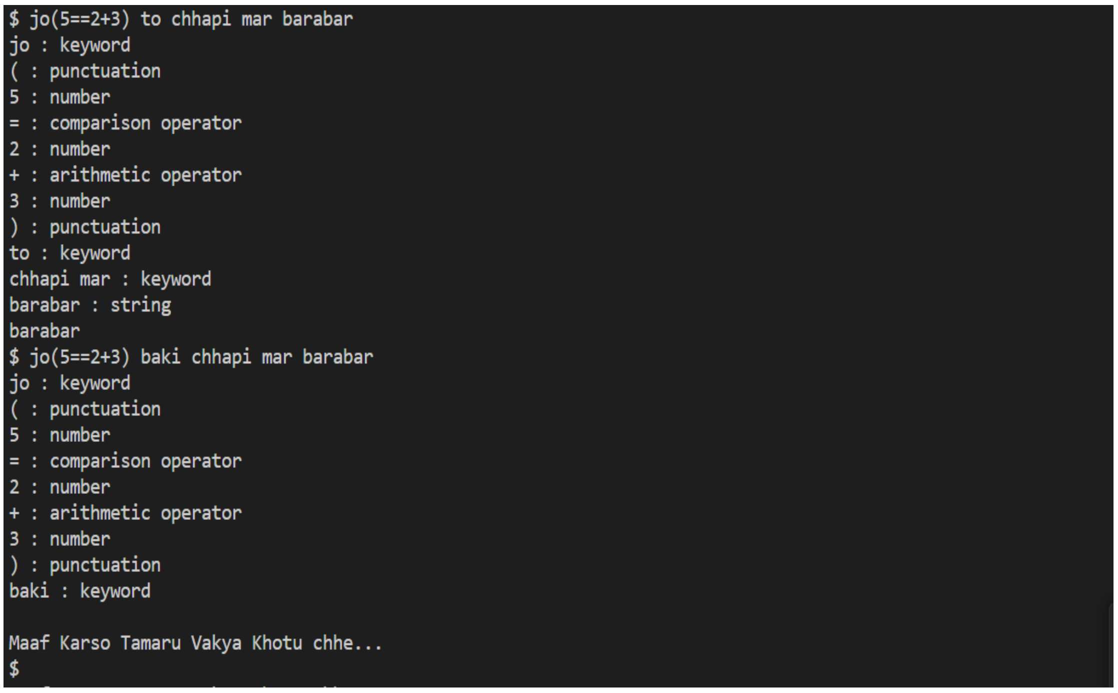
-Valid and Invalid output:-



```
$ chhapi mar (5+2)*12
chhapi mar : keyword
( : punctuation
5 : number
+ : arithmetic operator
2 : number
) : punctuation
* : arithmetic operator
12 : number
84
$ chhapi mar halo  (5+2)*12
chhapi mar : keyword
halo : string

Maaf Karso Tamaru Vakya Khotu chhe...
$
```

(3) Jo (5==2+3) to chhapi mar barabar

-Valid and Invalid output:-



```
$ jo(5==2+3) to chhapi mar barabar
jo : keyword
( : punctuation
5 : number
= : comparison operator
2 : number
+ : arithmetic operator
3 : number
) : punctuation
to : keyword
chhapi mar : keyword
barabar : string
barabar
$ jo(5==2+3) baki chhapi mar barabar
jo : keyword
( : punctuation
5 : number
= : comparison operator
2 : number
+ : arithmetic operator
3 : number
) : punctuation
baki : keyword

Maaf Karso Tamaru Vakya Khotu chhe...
$
```

Chatper4- 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.

GITHUB LINK :-
https://github.com/AumRathod/Compiler/