

PROJECT REPORT ON:
Compiler for
<<SIMPLE INTEREST CALCULATOR USING STRINGS
>>

Developed by

Mufaddal Suratwala-IT164-19ITUOS119

Devansh Talati-IT165-19ITUOS110

Janshi Thaker-IT166-20ITUOD023

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 calculating simple interest using strings ” is a bonafied report of the work carried out by

- 1) Mr.Mufaddal Suratwala Student ID No: 19ITUOS119
- 2) Mr. Devansh Talati Student ID No: 19ITUOS110
- 3) Miss Janshi Thaker Student ID No:20ITUOD023

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 the academic year 2020-2021.

Prof. N.P. Desai
(Lab Incharge)
Department of Information Technology,
Faculty of Technology,
Dharmsinh Desai University, Nadiad
Date: 7th March 2022

Prof. (Dr.)V K Dabhi,
Head , Department of Information Technology,
Faculty of Technology,
Dharmsinh Desai University, Nadiad
Date: 7th March 2022

Index

<u>1.0.1 Project Details.....</u>	<u>4</u>
<u>1.0.2 Project Planning.....</u>	<u>4</u>
<u>2.0 Lexical phase design</u>	
<u>2.0.1 Regular Expressions.....</u>	<u>5</u>
<u>2.0.2 Deterministic Finite Automaton design for lexer.....</u>	<u>6</u>
<u>2.0.3 Algorithm of lexer.....</u>	<u>7</u>
<u>2.0.4 Implementation of lexer.....</u>	<u>21</u>
<u>2.0.5 Execution environment setup.....</u>	<u>22</u>
<u>2.0.6 Output screenshots of lexer.....</u>	<u>25</u>
<u>3.0 Syntax analyzer design</u>	
<u>3.0.1 Yacc based implementation of syntax analyzer.....</u>	<u>31</u>
<u>3.0.2 Execution environment setup.....</u>	<u>34</u>
<u>3.0.3 Output screenshots of yacc based implementation.....</u>	<u>35</u>
<u>4.0 Conclusion.....</u>	<u>38</u>
<u>5.0 Additional Work.....</u>	<u>39</u>

Introduction

1.0.1 Project Details

Language name: Simple interest calculator using strings.

Language Description:

Write an appropriate language using which simple interest can be calculated for the given string.

Example:

principal is 12390 rate of interest is 10 time is 10 then SI.

1.0.2 Project Planning

List of Students with their Roles/Responsibilities:

IT164 Mufaddal Suratwala: Scanner phase implementation, Grammar rules, YACC implementation

IT165 Devansh Talati: DFA design, Algorithm design and implementation

IT166 Janshi Thaker: Regular expression, Final Report

2.0 LEXICAL PHASE DESIGN

2.0.1 Regular Expression:

Keywords:

RE	Token
principal	principal
is	is
rate	rate
of	of
interest	interest
time	time
then	then
SI	SI

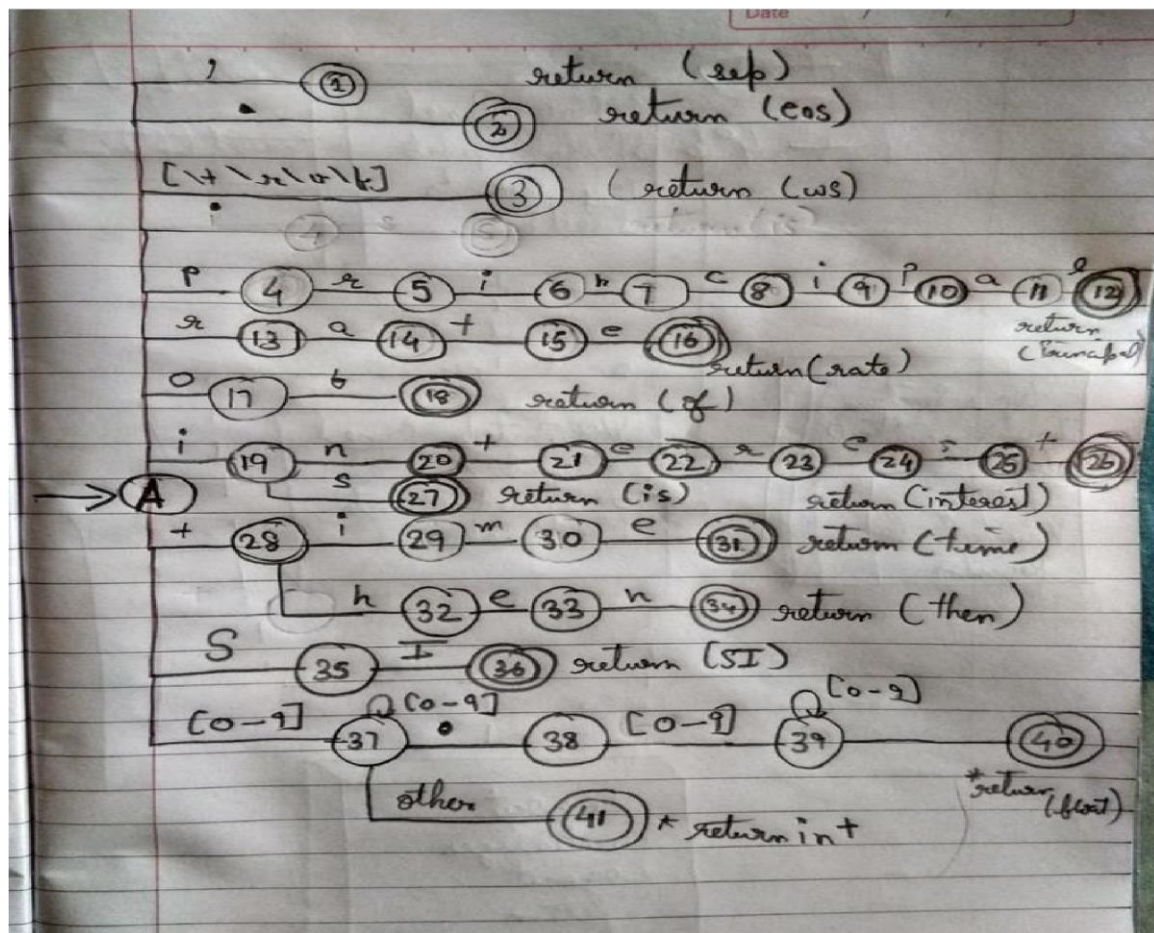
Values type: int and float

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 for 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;

['\t\r\v\f \.]:

state = "3";

ch=

nextchar();

f= true;

break;

'p':

state = "4";

ch=nextchar();

break;

'r':

state =

"13"; ch =

nextchar();

break;

'o':

state = "17";

ch =


```
nextchar();
```

```
break;
```

```
'i':
```

```
state =
```

```
"19"; ch =
```

```
nextchar();
```

```
break;
```

```
't':
```

```
state = "28";
```

```
ch
```

```
=nextchar();
```

```
break;
```

```
'S':
```

```
state = "35";
```

```
ch
```

```
=nextchar();
```

```
break;
```

```
[0-9]:
```

```
state =
```

```
"37"; ch
```

```
=nextchar();
```

```
break;
```

```
}
```

default:

f= true;

end case

case state of

"4": case

state of 'r':

state = "5";

ch =

nextchar();

break;

case state of '5':

case state of

'i': state =

"6"; ch=

nextchar();

break;

default:

f = true;

case state of

"6": case

state of 'n':

state = "7";

ch =

nextchar();

case state of

"7": case

state of 'c':

state = "8";

ch =

nextchar();

case state of

"8": case

state of 'i':

state = "9";

ch =

nextchar();

case state of

"9": case

state of 'p':

state = "10";

ch =

```
nextchar(); f=  
true;
```

```
case state of  
"10": case  
state of 'a':  
state = "11";  
ch      =  
nextchar();  
case state of  
"11": case  
state of 'l':  
state = "12";  
ch      =  
nextchar();  
default: f=  
true;
```

```
case state of  
"13": case  
state of 'a':  
state = "14";  
ch =
```

```
nextchar();  
case state of  
"14": case  
state of 't':  
state = "15";  
ch =  
nextchar();  
case state of  
"15": case  
state of 'e':  
state = "16";  
ch =  
nextchar();
```

```
default:  
f=true; case  
state of "17":  
case state of  
'f': state =  
"18"; ch =  
nextchar();  
f=true;
```

```
case state of
"19":    case
state of 's':
state = "27";
ch      =
nextchar();
break;   case
state of "19":
case state of
'n':    state =
"20";   ch =
nextchar();
case state of
"20":    case
state of 't':
state = "21";
ch      =
nextchar();
break;
```

```
case state of
"21":    case
state of 'e':
state = "22";
```

```
ch = nextchar();
```

```
case state of
```

```
"22": case
```

```
state of 'r':
```

```
state = "23";
```

```
ch =
```

```
nextchar();
```

```
case state of
```

```
"23": case
```

```
state of 'e':
```

```
state = "24";
```

```
ch =
```

```
nextchar();
```

```
case state of
```

```
"24": case
```

```
state of 's':
```

```
state = "25";
```

```
ch =
```

```
nextchar();
```

```
case state of
"25":    case
state of 't':
state = "26";
ch      =
nextchar(); f =
true;
```

```
case state of
"28":    case
state of 'i':
state = "29";
ch      =
nextchar();
case state of
"28":    case
state of 'h':
state = "32";
ch      =
nextchar();
```


case state of

"29": case

state of 'm':

state = "30";

ch =

nextchar();

case state of

"30": case

state of 'e':

state = "31";

ch =

nextchar();

case state of

"32": case

state of 'e':

state = "33";

ch =

nextchar();

case state of

"33": case

state of 'h':

state = "34";

ch =

nextchar();

f=true;

case state of

"35": case

state of 'l':

state = "36";

ch = nextchar();

f=true;

case state

of "37":

case state

of [0-9] :

state = "38";

ch =

nextchar();

case state of

':': state =

"38"; ch

=nextchar();

case state of

"37": case

state of [0-9] :

state = "41";

ch =

nextchar(); f=

true;

case state of "38":

case state of [0-9]:

state = "39";

ch =

nextchar();

case state of

"39": case

state of [0-9]:

state = "40";

```
ch =
nextchar();  f
= true;

}
}
case state of
"12"|"16"|"18"|"26"|"27"|"31"|"34"|"36":print("Keyword");
"40": print("float");
"41": print("int");
"1": print(" sep");
"2": print("
eos"); "3":
print(" ws ");
default:

print("invalid
input"); ch :=
nextchar(); end
case;

}
}
```

2.0.4 Implementation of lexer

Flex Code:

```
%option noyywrap

%{
    #include<stdio.h>

}%

Keyword "principal"|"is"|"rate"|"of"|"interest"|"time"|"SI"

Op      "shu"

Digit   [0-
100000]  Int

{Digit}+  qm

"?" ws    [\t]

eos      "." sep

" ,"

%%

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

{Op}      {printf("Operator - %s\n",yytext);}

{Int}     {printf("Integer - %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 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

Language Translator (IT608) LEXICAL PHASE DESIGN

19

- 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

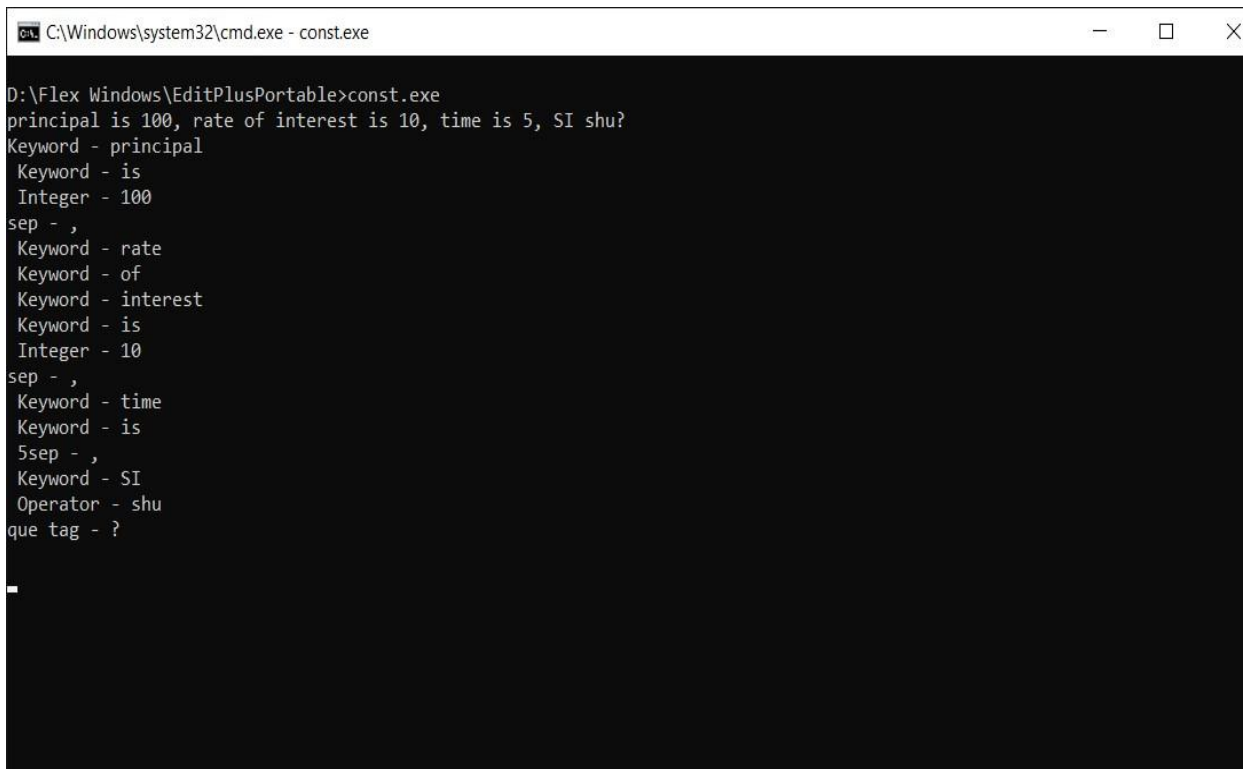
- Execute/Run for windows command prompt :- a.exe

Step 6

- Finished

2.0.6 Output Screenshots of lexer.

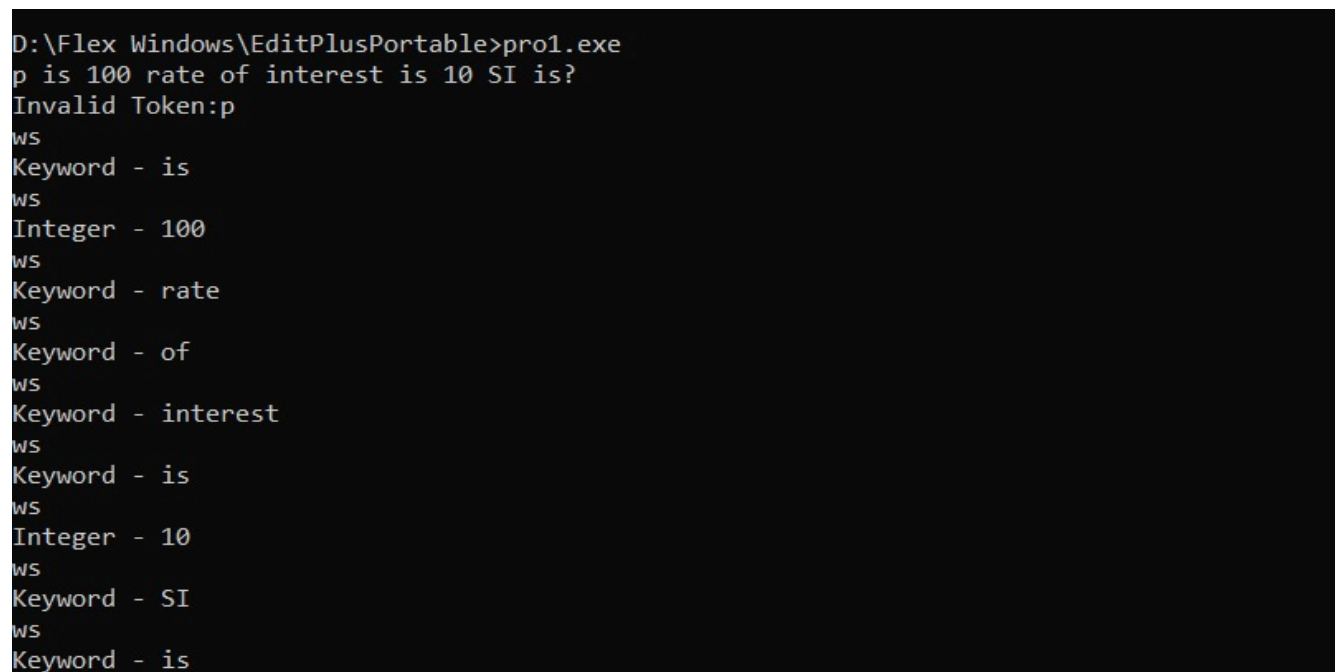
Valid Tokens:



```
C:\Windows\system32\cmd.exe - const.exe

D:\Flex Windows\EditPlusPortable>const.exe
principal is 100, rate of interest is 10, time is 5, SI shu?
Keyword - principal
Keyword - is
Integer - 100
sep - ,
Keyword - rate
Keyword - of
Keyword - interest
Keyword - is
Integer - 10
sep - ,
Keyword - time
Keyword - is
5
sep - ,
Keyword - SI
Operator - shu
que tag - ?
```

Invalid tokens:



```
D:\Flex Windows\EditPlusPortable>pro1.exe
p is 100 rate of interest is 10 SI is?
Invalid Token:p
ws
Keyword - is
ws
Integer - 100
ws
Keyword - rate
ws
Keyword - of
ws
Keyword - interest
ws
Keyword - is
ws
Integer - 10
ws
Keyword - SI
ws
Keyword - is
```

Scanner phase implementation in “C” language

CODE:

```
#include <stdbool.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

bool isDelimiter(char ch)
{
    if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||
        ch == '/' || ch == ',' || ch == ';' || ch == '>' || ch == '<' ||
        ch == '=' || ch == '(' || ch == ')' || ch == '[' || ch
        == ']' || ch == '{' || ch == '}')
        return (true);
    return (false);
}

// Returns 'true' if the string is a KEYWORD.
bool isKeyword(char* str)
{
    if (!strcmp(str, "principal") || !strcmp(str, "is") || !strcmp(str, "rate")
        || !strcmp(str, "of") || !strcmp(str, "interest") ||
        !strcmp(str, "time") || !strcmp(str, "then") || !strcmp(str, "SI")
    )
```

Language Translator (IT608)

```
        return (true);

        return (false);
    }

// Returns 'true' if the string is an INTEGER. bool
isInteger(char* str)

{    int i, len =
strlen(str);

    if (len == 0)
        return (false);

        for (i = 0; i < len; i++)

        {        if (str[i] != '0' && str[i] != '1' && str[i] != '2'
&& str[i] != '3' && str[i] != '4' && str[i] != '5' && str[i] != '6'
&& str[i] != '7' && str[i] != '8' && str[i] != '9' )
            return (false);

        }

    return (true);
}

// Extracts the SUBSTRING.
char* subString(char* str, int left, int right)
{    int i;

    char* subStr = (char*)malloc(sizeof(char) * (right - left + 2));
```

```
        for (i = left; i <= right; i++)
            subStr[i - left] = str[i];
subStr[right - left + 1] = '\0';

        return (subStr);
}

// Parsing the input STRING.
void parse(char* str)
{
    int left = 0, right =
0;

    int len = strlen(str);

    while (right <= len && left <= right) {
        if (isDelimiter(str[right]) == false)
            right++;

        if (isDelimiter(str[right]) == true && left == right) {
            if (isKeyword(str) == true)
                printf("%s' IS A KEYWORD\n", str);

            right++;
            left = right;
        }
    }
}
```

```
if (isDelimiter(str[right]) == true && left != right
    || (right == len && left != right)) {
    char* subStr = subString(str, left, right - 1);

    if (isKeyword(subStr) == true)
        printf("%s' IS A KEYWORD\n", subStr);

    else if (isInteger(subStr) == true)
        printf("%s' IS AN INTEGER\n", subStr);

    left = right;
}
}
return;
}


// DRIVER FUNCTION
int main()
{
    char str[1000] = "principal is 10000 rate of interest is 10 time is 5 then SI is";

    parse(str); // calling the parse function

    return (0);
}
```

}

OUTPUT:

 D:\Lan Translator\lab4.exe

```
'principal' IS A KEYWORD
'is' IS A KEYWORD
'10000' IS AN INTEGER
'rate' IS A KEYWORD
'of' IS A KEYWORD
'interest' IS A KEYWORD
'is' IS A KEYWORD
'10' IS AN INTEGER
'time' IS A KEYWORD
'is' IS A KEYWORD
'5' IS AN INTEGER
'then' IS A KEYWORD
'SI' IS A KEYWORD
'is' IS A KEYWORD
```

```
-----
Process exited after 0.1458 seconds with return value 0
Press any key to continue . . .
```

3.0 SYNTAX ANALYZER DESIGN

3.0.1 Yacc based Implementation of syntax analyser

1) project.y(yacc code)

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

%token KEYWORD NUMBER WHITESPACE DOT

%%
S : A { printf("\nThese Sentences are Valid. \n\n"); exit(0); }
;
A : E E D | E G Q
;
D : F A | K
;
K : A
;
E : KEYWORD WHITESPACE
;
G : KEYWORD
;
F : NUMBER WHITESPACE
;
Q : DOT |
```

```
;
%%

void yyerror(char *err) {
printf("Error: Invalid sentence");
exit(1);
}

int main() {
printf("Enter sentences to calculate simple interest:\n");
yyparse();
}
```

2) project.l(Lex file)

```
%{
    #include<stdio.h>
    #include<stdlib.h>
    #include<string.h>
    #include "y.tab.h";
}%
Keyword "principal"|"is"|"rate"|"of"|"interest"|"time"|"SI"|"then"
Digit  [0-9]+
Int    {Digit}+
eos    "."
ws     [ \t\n]
%%
{Keyword} {printf("Keyword - %s\n",yytext);return KEYWORD;}
{Int}     {printf("Integer - %s\n",yytext);return NUMBER;}
{eos}     {printf("End of sentence - %s\n",yytext);return DOT;}
{ws}      {return WHITESPACE;}
.         {printf("Invalid Token : %s\n",yytext); return *yytext;}
%%
```



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

3.0.2 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/orwelldvcpp/> 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
```

3.0.3 Output screenshots of yacc based implementation

- SOME POSSIBLE VALID INPUTS

```
D:\Flex Windows\EditPlusPortable>project1.exe
Enter sentences to calculate simple interest:
principal is 10 rate of interest is 10 time is 10 then SI.
Keyword - principal
Keyword - is
Integer - 10
Keyword - rate
Keyword - of
Keyword - interest
Keyword - is
Integer - 10
Keyword - time
Keyword - is
Integer - 10
Keyword - then
Keyword - SI
End of sentence - .

These Sentences are Valid.
```

```
D:\Flex Windows\EditPlusPortable>project1.exe
Enter sentences to calculate simple interest:
rate of interest is 10 principal is 1087 time is 10 then SI.
Keyword - rate
Keyword - of
Keyword - interest
Keyword - is
Integer - 10
Keyword - principal
Keyword - is
Integer - 1087
Keyword - time
Keyword - is
Integer - 10
Keyword - then
Keyword - SI
End of sentence - .

These Sentences are Valid.
```

```
D:\Flex Windows\EditPlusPortable>project1.exe
Enter sentences to calculate simple interest:
time is 10 principal is 9089 rate of interest is 10 then SI.
Keyword - time
Keyword - is
Integer - 10
Keyword - principal
Keyword - is
Integer - 9089
Keyword - rate
Keyword - of
Keyword - interest
Keyword - is
Integer - 10
Keyword - then
Keyword - SI
End of sentence - .

These Sentences are Valid.
```

- **INVALID SYNTAX**

1) Misplaced keywords:

```
D:\Flex Windows\EditPlusPortable>project1.exe
Enter sentences to calculate simple interest:
principal is 2122 rate 10
Keyword - principal
Keyword - is
Integer - 2122
Keyword - rate
Integer - 10
Error: Invalid sentence
```

2) Missing number:

```
D:\Flex Windows\EditPlusPortable>project.exe
Enter sentences to calculate simple interest:
principal is rate
Keyword - principal
Keyword - is
Keyword - rate
Error: Invalid sentence
```

3) Beginning with number:

```
D:\Flex Windows\EditPlusPortable>project1.exe
Enter sentences to calculate simple interest:
10 is principal
Integer - 10
Error: Invalid sentence
```

4) Beginning with dot

```
D:\Flex Windows\EditPlusPortable>project1.exe
Enter sentences to calculate simple interest:
. principal is 10
End of sentence - .
Error: Invalid sentence
```

5) Incorrect end of statement

```
Enter sentences to calculate simple interest:
principal is 10.
Keyword - principal
Keyword - is
Integer - 10
End of sentence - .
Error: Invalid sentence
```

6) Missing end of statement

```
D:\Flex Windows\EditPlusPortable>project1.exe
Enter sentences to calculate simple interest:
principal is 1232112 rate of interest is 10 time is 10 then SI
Keyword - principal
Keyword - is
Integer - 1232112
Keyword - rate
Keyword - of
Keyword - interest
Keyword - is
Integer - 10
Keyword - time
Keyword - is
Integer - 10
Keyword - then
Keyword - SI
Error: Invalid sentence
```

4.0 CONCLUSION

The project has been implemented based on what we were taught in our college curriculum and many resources on the web. After the completion of the project and the implementation of our very own language our understanding regarding the functioning and error handling of compilers has improved greatly. At the end our appreciation for the compilers developed by various developers throughout history has increased greatly through the efforts we put in to develop our own compiler.

5.0 ADDITIONAL WORK

(Here we have divided the keywords into two parts MKEYWORD and LKEYWORD)

MKEYWORDS always need to be followed by LKEYWORDS

MKEYWORDS : principal , rate ,time, interest, SI

LKEYWORDS: of, is, then

In the above language :

is principal 122 is time 10 of interest rate is 10 SI then.

Would be accepted as a valid string but by dividing the keywords into a much more stricter form such sentences would be considered invalid in this language.

YACC CODE:

project2.y

```
%{
```

```
#include<stdlib.h>
```

```
#define YYERROR_VERBOSE 1
```

```
void yyerror(char* err);
```

```
%}
```

```
%token MKEYWORD LKEYWORD NUMBER WHITESPACE DOT
```

```
%%
```

```
S : A { printf("\nThese Sentences are Valid. \n\n"); exit(0); }  
  
;  
  
A : B B C H G Q  
  
;  
  
B : E H F  
  
;  
  
C : E H E H F  
  
;  
  
E : MKEYWORD WHITESPACE  
  
;  
  
H : LKEYWORD WHITESPACE  
  
;  
  
G : MKEYWORD  
  
;  
  
F : NUMBER WHITESPACE  
  
;  
  
Q : DOT |  
  
;  
  
%%
```

```
void yyerror(char *err) {  
    printf("Error: Invalid sentence");  
    exit(1);  
}
```



```
}
```

```
int main() {  
printf("Enter sentences to calculate simple interest:\n");  
yyparse();  
}
```

LEX CODE:

```
%{  
    #include<stdio.h>  
    #include<stdlib.h>  
    #include<string.h>  
    #include "y.tab.h";  
%}  
  
MKeyword "principal"|"rate"|"interest"|"time"|"SI"  
LKeyword "is"|"of"|"then"  
Digit  [0-9]+  
Int    {Digit}+  
eos    "."  
ws     [ \t\n]  
%%  
  
{MKeyword} {printf("MKeyword - %s\n",yytext);return MKEYWORD;}  
{LKeyword} {printf("LKeyword - %s\n",yytext);return LKEYWORD;}
```

```
{Int}    {printf("Integer - %s\n",yytext);return NUMBER;}
{eos}    {printf("End of sentence - %s\n",yytext);return DOT;}
{ws}     {return WHITESPACE;}
.        {printf("Invalid Token : %s\n",yytext); return *yytext;}
%%
```

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

Valid syntax:

```
D:\Flex Windows\EditPlusPortable>project2.exe
Enter sentences to calculate simple interest:
principal is 112321 time is 10 rate of interest is 10 then SI.
MKeyword - principal
LKeyword - is
Integer - 112321
MKeyword - time
LKeyword - is
Integer - 10
MKeyword - rate
LKeyword - of
MKeyword - interest
LKeyword - is
Integer - 10
LKeyword - then
MKeyword - SI
End of sentence - .

These Sentences are Valid.
```

Invalid syntax:

```
D:\Flex Windows\EditPlusPortable>project2.exe
Enter sentences to calculate simple interest:
is principal 10.
LKeyword - is
Error: Invalid sentence
D:\Flex Windows\EditPlusPortable>
```

```
D:\Flex Windows\EditPlusPortable>project2.exe
Enter sentences to calculate simple interest:
is principal 122 is time 10 of interest rate is 10 SI then.
LKeyword - is
Error: Invalid sentence
D:\Flex Windows\EditPlusPortable>
```

Comparision:

Previous compiler:

```
D:\Flex Windows\EditPlusPortable>project1.exe
Enter sentences to calculate simple interest:
is principal 122 is time 10 of interest rate is 10 SI then.
Keyword - is
Keyword - principal
Integer - 122
Keyword - is
Keyword - time
Integer - 10
Keyword - of
Keyword - interest
Keyword - rate
Keyword - is
Integer - 10
Keyword - SI
Keyword - then
End of sentence - .

These Sentences are Valid.
```

New compiler:

```
D:\Flex Windows\EditPlusPortable>project2.exe
Enter sentences to calculate simple interest:
principal is 112321 time is 10 rate of interest is 10 then SI.
MKeyword - principal
LKeyword - is
Integer - 112321
MKeyword - time
LKeyword - is
Integer - 10
MKeyword - rate
LKeyword - of
MKeyword - interest
LKeyword - is
Integer - 10
LKeyword - then
MKeyword - SI
End of sentence - .

These Sentences are Valid.
```