Project Report

on

Compiler for SIMPLE AND FLOAT NUMERICAL OPERATIONS

Developed by

Parth Bandel - IT004 - 19ITUOS013

Dev Bhagat - IT005 - 20ITUOD019

Kaushal Bhalaiya - IT006 - 19ITUBS123

Guided By:

Prof. Nikita P. Desai

Dept. of Information Technology

Department of Information Technology

Faculty of Technology

Dharmsinh Desai University

College Road, Nadiad-387001

2022

DHARMSINH DESAI UNIVERSITY

NADIAD-387001, GUJARAT

This is to certify that the project entitled "**Simple and Float Numerical Operations"** is a bonafied report of the work carried out by

1) Mr. Parth Bandel, Student ID No: 19ITUOS013
2) Mr. Dev Bhagat, Student ID No: 20ITUOD019
3) Mr. Kaushal Bhalaiya, Student ID No: 19ITUBS123

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 In charge)

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

Grammar Name: Simple and Float number operations with binary and unary assignment operators.

Grammar Rules:

1. X++: used to indicate two increments to “X”
2. X--: used to indicate two decrements to “X” integer variable.
3. X**: used to indicate X variable should be doubled.
4. X=+Y12: Does the addition of X and Y12 and stores value back in “X”.
5. X=-Y: Does the addition of X and Y and stores value back in “X”.
6. X=<integer or float value>: To assign some value to the variable
7. <datatype> <variable name list separated by semicolon>: To define the variables with a particular datatype.

● Variables can be of type: “simple” or “float”

● Variable names should start with capital alphabet and can be followed by maximum 2 digits.

● Sentence end is indicated by “:”

1.0.2 Project Planning

List of Students with their Roles/Responsibilities:

IT004-Parth Bandel:

- Generate Token
- Algorithm
- C++ implementation

IT005-Dev Bhagat:

- DFA design
- Regular Expressions

IT006-Kaushal Bhalaiya:

- Grammar Rules
- Complete parser with Flex and Bison (Yacc) code
- Testing with different test cases
- Project Report

2.0 LEXICAL PHASE DESIGN

2.0.1 Regular Expression

Regular Expression:

cletter → [A-Z]

letter → [A-Za-z]

digit → [0-9]

op → ++ | -- | **

assignop → =+ | =- | =

simple → simple

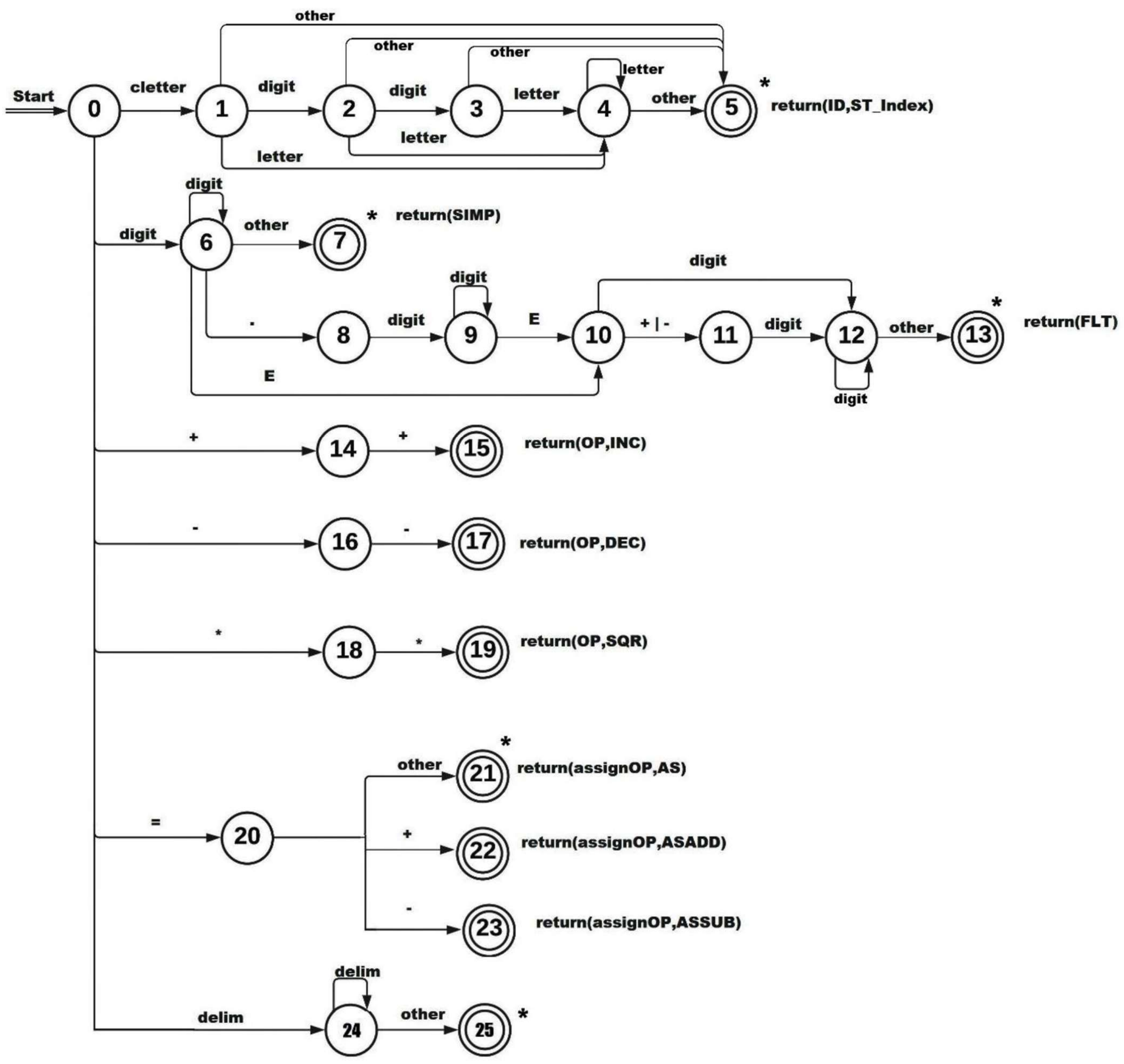
float → float

id → cletter(digit?)(digit?)(letter)*

num → digit+((.)(digit+))?

ws → (blank | tab | newline) +

2.0.2 Deterministic Finite Automaton design for lexer

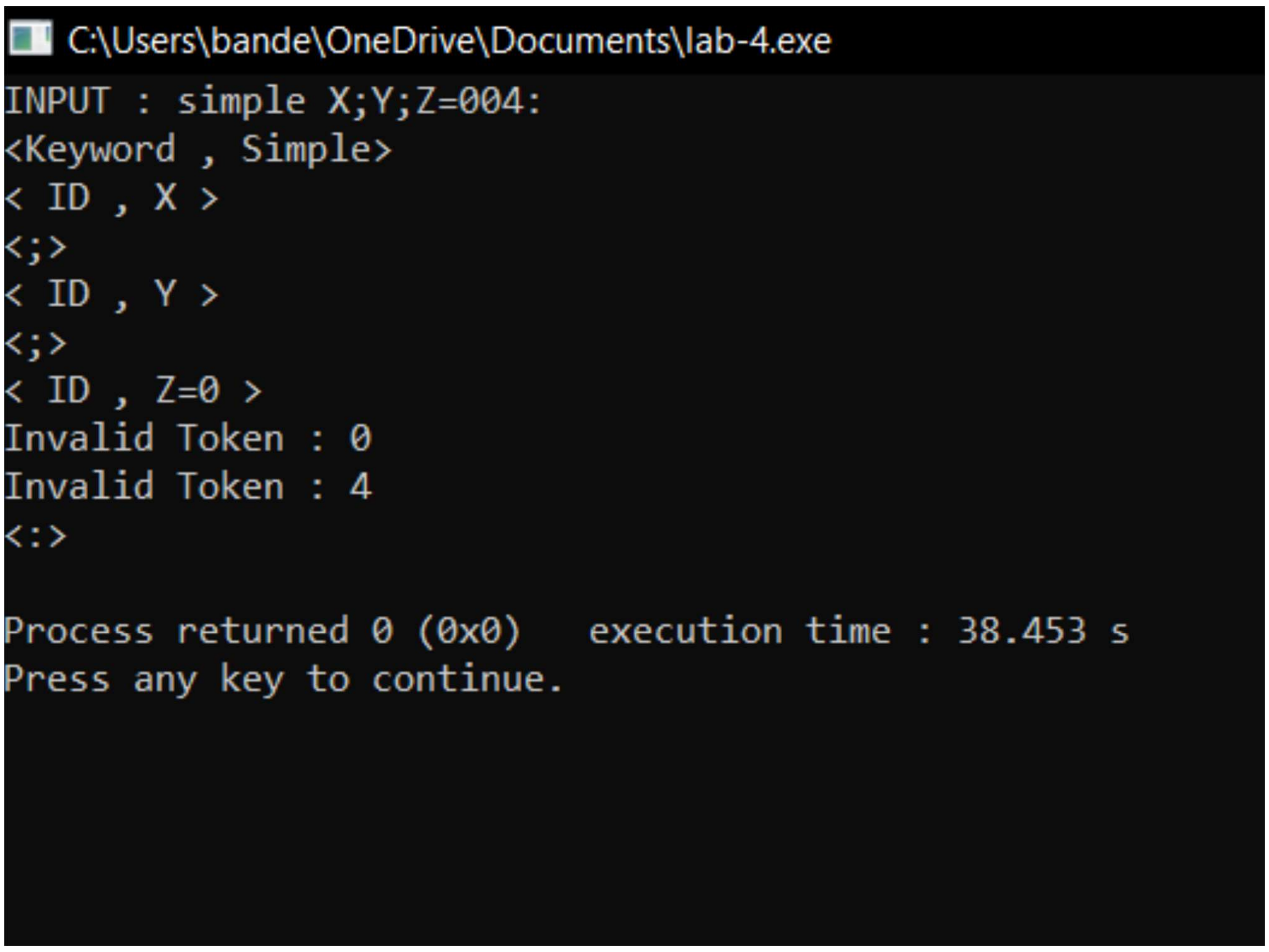


2.0.3 Algorithm of lexer

```
#include<bits/stdc++.h>


using namespace std;

void identify(string s) {

 int length = s.length();

 int i = 0;

 while (i < length) {

  if (s[i] == 's') {

   i++;

   if (s[i] == 'i' && s[i + 1] == 'm' && s[i + 2] == 'p' && s[i + 3] == 'l' && s[i +

     4] == 'e') {

    cout << "<Keyword , Simple>\n";

    i += 5;

   } else {

    i--;

    //goto var;

   }

  } else if (s[i] == 'f') {

   i++;

   if (s[i] == 'l' && s[i + 1] == 'o' && s[i + 2] == 'a' && s[i + 3] == 't') {

    cout << "<Keyword , Float>\n";
    i += 4;
  } else i--;
} else if (isupper(s[i])) {
  //var:
  vector < char > v;
  v.push_back(s[i++]);
  int len = 1, dig = 0;
  if (isdigit(s[i]) || isdigit(s[i + 1])) {
    v.push_back(s[i++]);
    v.push_back(s[i++]);
    loop1:
      if ((s[i] >= 'A' && s[i] <= 'Z') || (s[i] >= 'a' && s[i] <= 'z')) {
        v.push_back(s[i++]);
        goto loop1;
      }
  } else {
    loop2: if ((s[i] >= 'A' && s[i] <= 'Z') || (s[i] >= 'a' && s[i] <= 'z')) {
      v.push_back(s[i++]);
      goto loop2;
    }
  }
  if (len != 0) {
    cout << "< ID , ";
    for (auto ch: v) cout << ch;
    cout << " >\n";
  }
} else if (s[i] == '=') {
  if (s[i + 1] == '+') {
    cout << "<ASSIGNOP, =+>\n";
    i += 2;
  } else if (s[i + 1] == '-') {
    cout << "<ASSIGNOP, =->\n";
    i += 2;
  } else {
    cout << "<ASSIGNOP, =>\n";
    i++;
  }
} else if (s[i] == '+') {
  if (s[i + 1] == '+') {
    cout << "<OP, ++>\n";
    i += 2;
  } else goto invalid;
} else if (s[i] == '-') {
  if (s[i + 1] == '-') {
    cout << "<OP, -->\n";
    i += 2;
  } else goto invalid;
} else if (s[i] == '*') {
      if (s[i + 1] == '*') {
        cout << "<OP, *>\n";
        i += 2;
      } else goto invalid;
    } else if (s[i] == ';') {
      cout << "<;>\n";
      i++;
    } else if (s[i] == ':') {
      cout << "<:>\n";
      i++;
    } else if ((int) s[i] == 32) {
      i++;
    } else if (s[i] == '\t') {
      i += 4;
    } else {
      invalid: cout << "Invalid Token : " << s[i++] << endl;
    }
  }
}
int main() {
  string s;
  cout << "INPUT : ";
  getline(cin, s);
  identify(s);
```

}

OUTPUT-



```
C:\Users\bande\OneDrive\Documents\lab-4.exe
INPUT : simple X;Y;Z=004:
<Keyword , Simple>
< ID , X >
<;>
< ID , Y >
<;>
< ID , Z=0 >
Invalid Token : 0
Invalid Token : 4
<:>

Process returned 0 (0x0)   execution time : 38.453 s
Press any key to continue.
```

2.0.4 Implementation of lexer

Flex Program:

```
%{

#include <stdio.h>

%}

printf("\n\n");

%%

";"                     printf("VS            --->\t%s\n",yytext);

":"                     printf("EOS           --->\t%s\n",yytext);

"simple"|"float"            printf("datatype      --->\t%s\n",yytext);

"++"|"--"|"**"              printf("OP            --->\t%s\n",yytext);

"="|"=+"|"=-"               printf("ASSIGNOP      --->\t%s\n",yytext);

[A-Z][[0-9]?[0-9]?]?[a-z|A-Z]*  printf("variable      --->\t%s\n",yytext);

[+-]?([0-9]+([.][0-9]+)?)     printf("NUM           --->\t%s\n",yytext);

[ \t\r]+          /*NOP*/   print("");

%%

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**

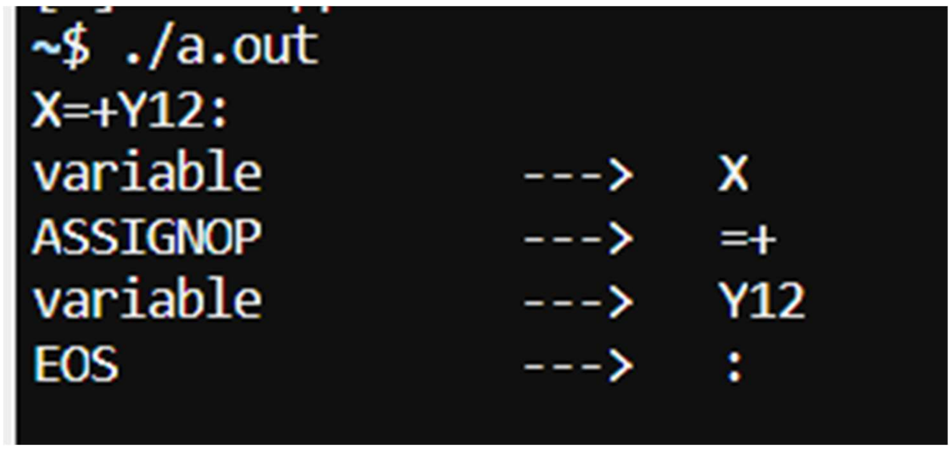
- Type in command :- **gcc lex.yy.c**

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

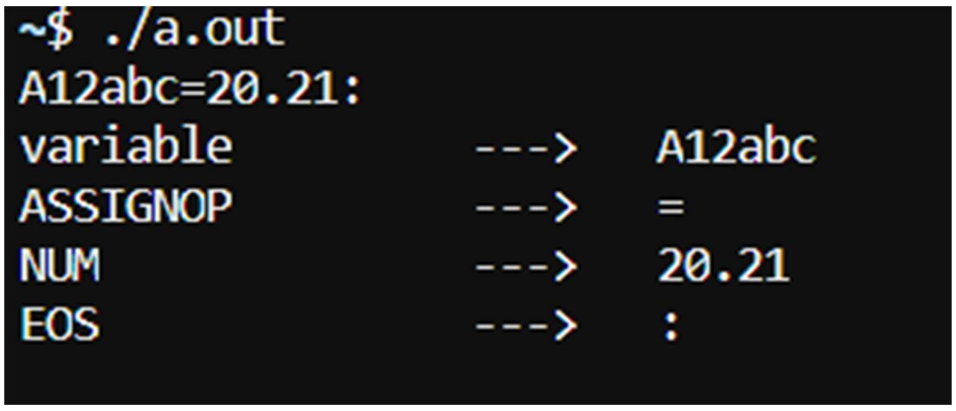
Step 6

- Finished

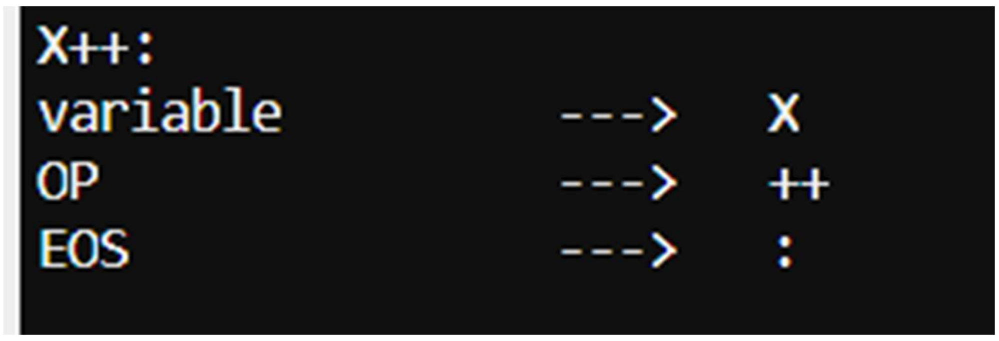
2.0.6 Output screenshots of lexer



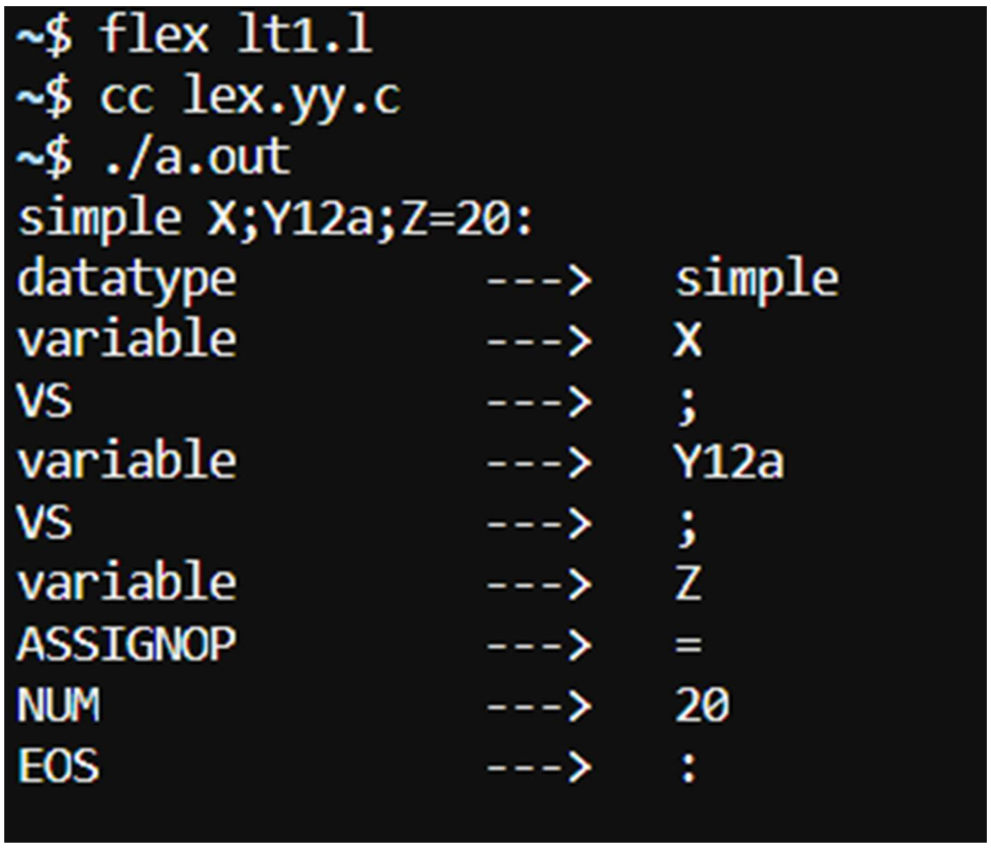
```
~$ ./a.out
X=+Y12:
variable            --->    X
ASSIGNOP            --->    =+
variable            --->    Y12
EOS                 --->    :
```



```
~$ ./a.out
A12abc=20.21:
variable            --->    A12abc
ASSIGNOP            --->    =
NUM                 --->    20.21
EOS                 --->    :
```



```
X++:
variable            --->    X
OP                  --->    ++
EOS                 --->    :
```



```
~$ flex lt1.l
~$ cc lex.yy.c
~$ ./a.out
simple X;Y12a;Z=20:
datatype          --->   simple
variable          --->   X
VS                --->   ;
variable          --->   Y12a
VS                --->   ;
variable          --->   Z
ASSIGNOP          --->   =
NUM               --->   20
EOS               --->   :
```

3.0 SYNTAX ANALYZER DESIGN

3.0.1 Grammar rules

S -> TYPE XID EOL | ASSSTMT EOL | EXPR EOL

EXPR -> ID UOP | ID BOP ID

XID -> YID | YID SEMI XID

YID -> ID | ASSSTMT

ASSSTMT -> ID AOP VALUE

AOP -> =

BOP -> =+ | =-

UOP -> ++ | -- | **

ID -> id

TYPE -> simple | float

EOL -> :

SEMI -> ;

Tokens that I’ve used:

LEXEMES	**TOKENS**	**DISCRIPTION**
simple	SIMP	Simple Datatype
float	FLT	Float Datatype
++	UOP	Unary increment
--	UOP	Unary decrement
**	UOP	Unary double
=	AOP	Assignment operator
=+	BOP	Binary addition operator
=-	BOP	Binary subtraction operator
;	SEMI	To Separate variable list
:	EOL	Sentence end
[A-Z][[0-9]?[0-9]?]?[a-zA-Z]*	ID	Variable name (identifier)
[+-]?[0-9]+	NUM_SIMP	Simple numbers
[+-]?([0-9]+([.][0-9]*)?|[.][0-9]+)	NUM_FLT	Float numbers
Whitespace (‘\0’)	WS	space

FIRST:

Non-Terminal Symbol	**First Set**
AOP	=
UOP	++, --, **
BOP	=+, =-
ID	id
EOL	:
SEMI	;
TYPE	simple, float
S	simple, float, id
EXPR	id
YID	id
ASSSTMT	id
XID	id

FOLLOW:

Non-Terminal Symbol	**Follow Set**
S	$
EXPR	:
XID	:
YID	;, :
ASSSTMT	:, ;
AOP	VALUE
UOP	:
BOP	id
ID	=, ++, --, **, =+, =-, ;, :
EOL	$
SEMI	id
TYPE	id

3.0.2 Yacc based imlementation of syntax analyzer

Project.yacc

```
%{

#include <stdio.h>

#define YYERROR_VERBOSE 1

void yyerror(char *err);

%}

%token ID UOP BOP AOP SIMP FLT NUM_SIMP NUM_FLT EOL SEMI ERR WS

%%

E : X { return 0;}

X : S EOL { printf("\nEntered expression is VALID according to grammar rules.\n\n"); return 0; }

;

S : TYPE WS XID

| ASSSTMT

| EXPR

;

EXPR : ID UOP

| ID WS BOP WS ID

;

XID : YID

| YID SEMI WS XID;

YID : ID| ASSSTMT;

ASSSTMT : ID WS AOP WS VALUE

;

VALUE : NUM_SIMP

| NUM_FLT
```

```
;

TYPE : SIMP

| FLT

;

%%

void yyerror(char *err) {

printf("Error: ");

fprintf(stderr, "%s\n", err);

exit(1);

}

int main() {

while(1){

printf("Enter the expression:\n");

yyparse();

}

}
```

Project.lex

```
%option noyywrap

%{

#include <stdio.h>

#include "y.tab.h";

%}

%%

; { printf(" SEMI ---> %s \n",yytext); return SEMI; }

: { printf(" EOL ---> %s \n",yytext); return EOL; }

"simple" { printf(" SIMP ---> %s \n",yytext); return SIMP; }
```

```
"float" { printf(" FLT ---> %s \n",yytext); return FLT; }

"++" { printf(" UOP ---> %s \n",yytext); return UOP;}

"--" { printf(" UOP ---> %s \n",yytext); return UOP;}

"**" { printf(" UOP ---> %s \n",yytext); return UOP;}

"=" { printf(" AOP ---> %s \n",yytext); return AOP;}

"=+" { printf(" BOP ---> %s \n",yytext); return BOP;}

"=-" { printf(" BOP ---> %s \n",yytext); return BOP;}

[A-Z][[0-9]?[0-9]?]?[a-zA-Z]* { printf(" ID ---> %s \n",yytext); return ID; }

[+-]?[0-9]+ { printf(" NUM_SIMP ---> %s \n",yytext); return NUM_SIMP; }

[+-]?([0-9]+([.][0-9]*)?|[.][0-9]+) { printf(" NUM_FLT ---> %s \n",yytext); return NUM_FLT; }

[ ] { printf("< WS >\n"); return WS; }

[\t\r]+ /* nop */

. { printf("------- INVALID TOKEN ------- %s \n",yytext);return *yytext; }

%%
```

3.0.3 Execution environment setup

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

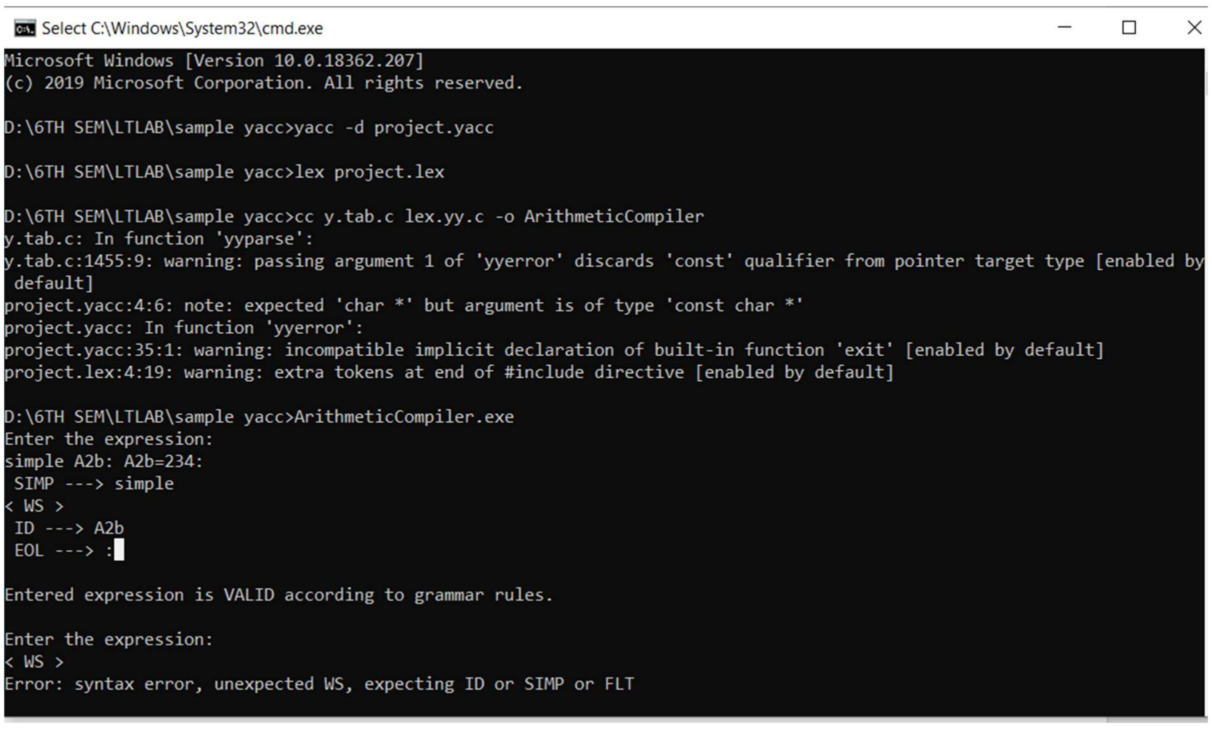
https://sourceforge.net/projects/orwelldevcpp/

set environment varialble and then run program

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

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

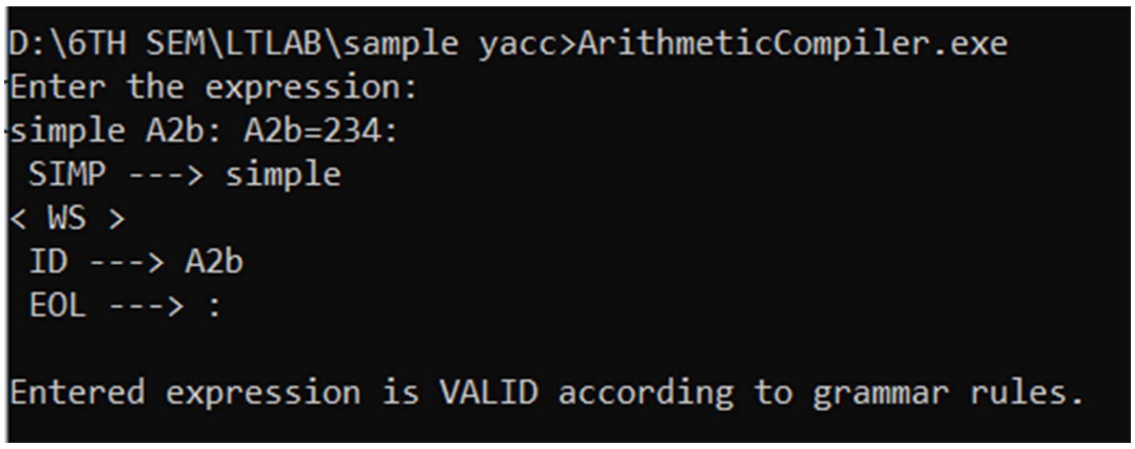
3.0.4 Output screenshots of yacc based implementation



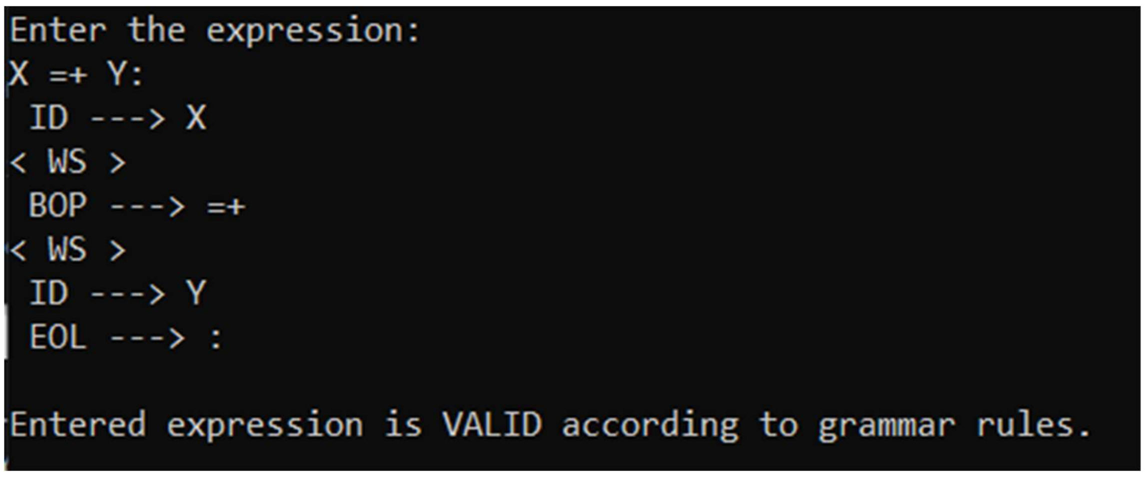
Select C:\Windows\System32\cmd.exe

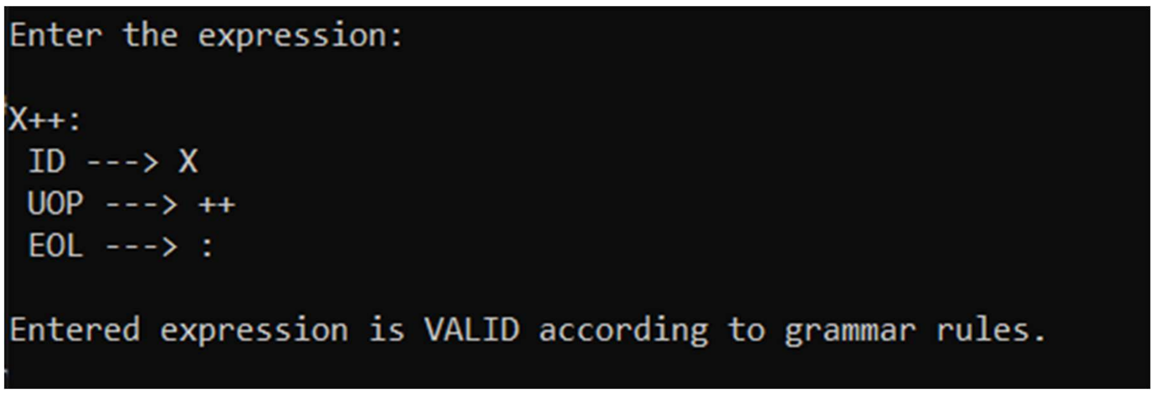
```
Microsoft Windows [Version 10.0.18362.207]
(c) 2019 Microsoft Corporation. All rights reserved.

D:\6TH SEM\LTLAB\sample yacc>yacc -d project.yacc

D:\6TH SEM\LTLAB\sample yacc>lex project.lex

D:\6TH SEM\LTLAB\sample yacc>cc y.tab.c lex.yy.c -o ArithmeticCompiler
y.tab.c: In function 'yyparse':
y.tab.c:1455:9: warning: passing argument 1 of 'yyerror' discards 'const' qualifier from pointer target type [enabled by
 default]
project.yacc:4:6: note: expected 'char *' but argument is of type 'const char *'
project.yacc: In function 'yyerror':
project.yacc:35:1: warning: incompatible implicit declaration of built-in function 'exit' [enabled by default]
project.lex:4:19: warning: extra tokens at end of #include directive [enabled by default]

D:\6TH SEM\LTLAB\sample yacc>ArithmeticCompiler.exe
Enter the expression:
simple A2b: A2b=234:
 SIMP ---> simple
< WS >
 ID ---> A2b
 EOL ---> :

Entered expression is VALID according to grammar rules.

Enter the expression:
< WS >
Error: syntax error, unexpected WS, expecting ID or SIMP or FLT
```

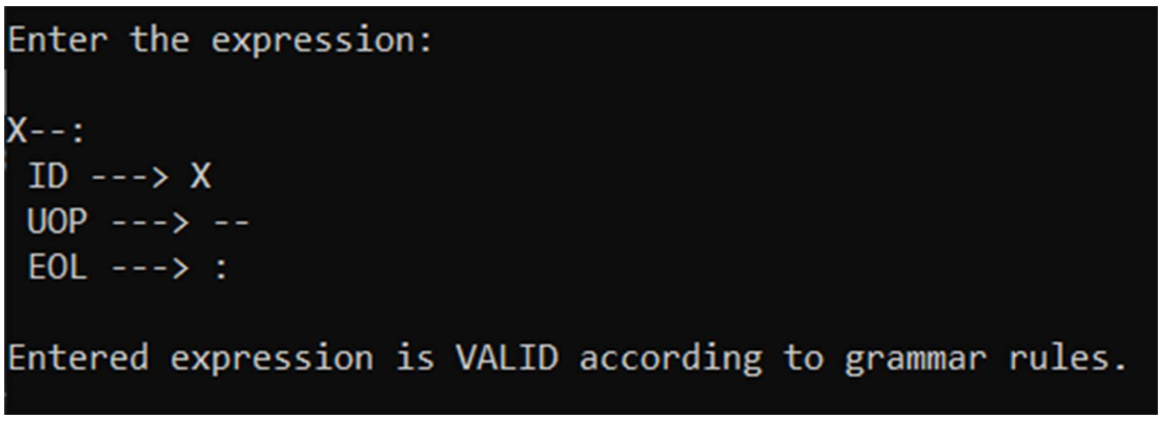
VALID TESTCASES:

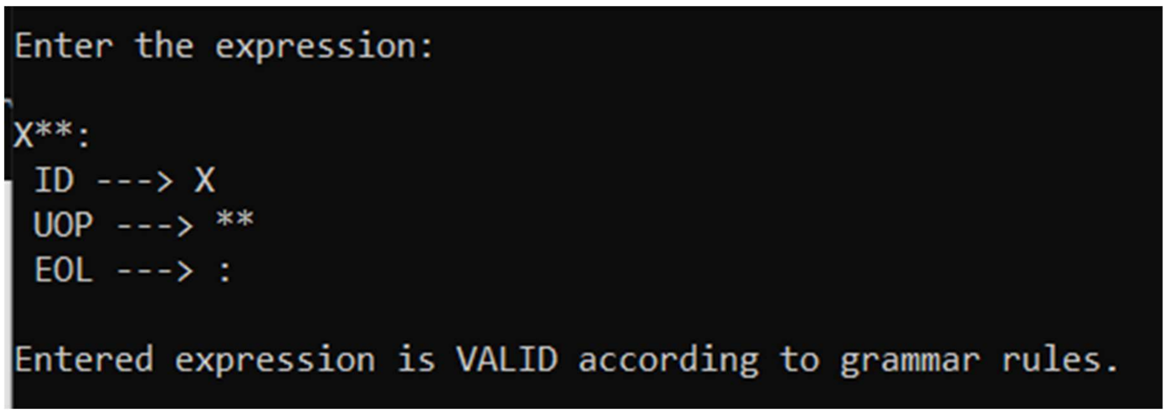


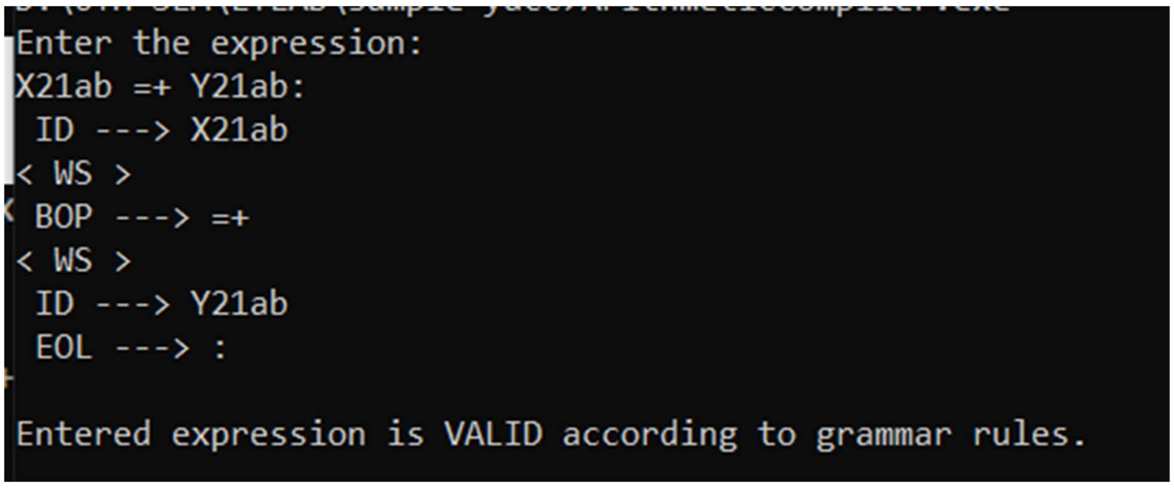
```
D:\6TH SEM\LTLAB\sample yacc>ArithmeticCompiler.exe
Enter the expression:
simple A2b: A2b=234:
 SIMP ---> simple
< WS >
 ID ---> A2b
 EOL ---> :

Entered expression is VALID according to grammar rules.
```


```
Enter the expression:
X =+ Y:
 ID ---> X
< WS >
 BOP ---> =+
< WS >
 ID ---> Y
 EOL ---> :

Entered expression is VALID according to grammar rules.
```

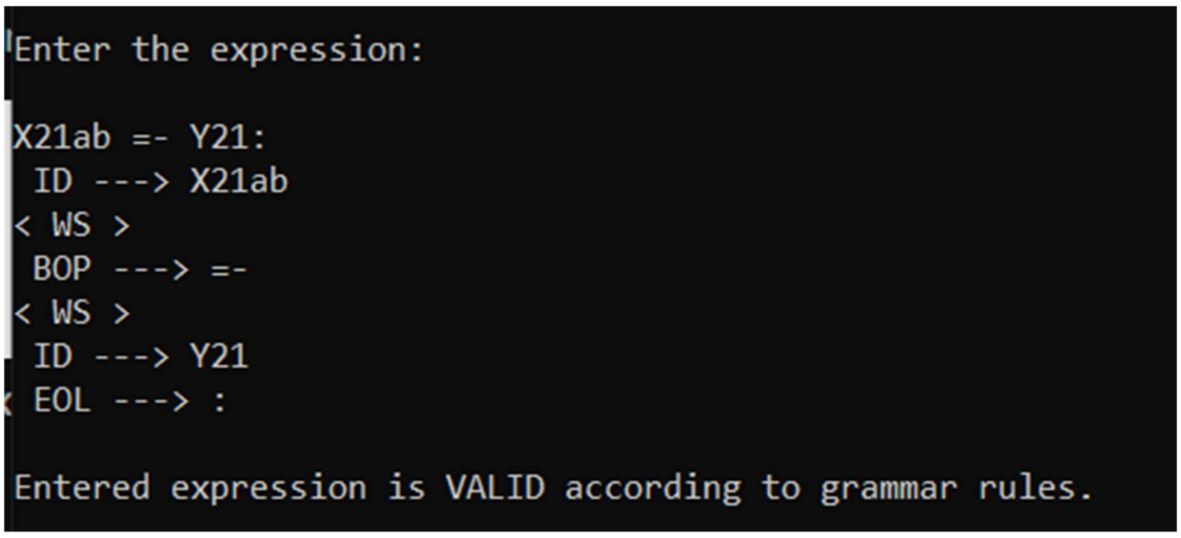

```
Enter the expression:

X++:
 ID ---> X
 UOP ---> ++
 EOL ---> :

Entered expression is VALID according to grammar rules.
```


```
Enter the expression:

X--:
 ID ---> X
 UOP ---> --
 EOL ---> :

Entered expression is VALID according to grammar rules.
```

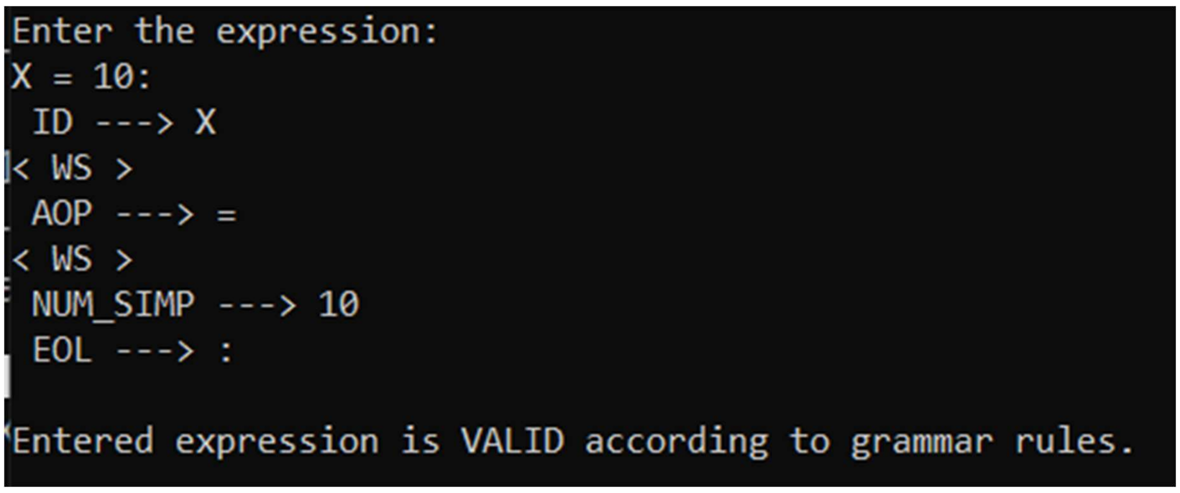

```
Enter the expression:

X**:
 ID ---> X
 UOP ---> **
 EOL ---> :

Entered expression is VALID according to grammar rules.
```



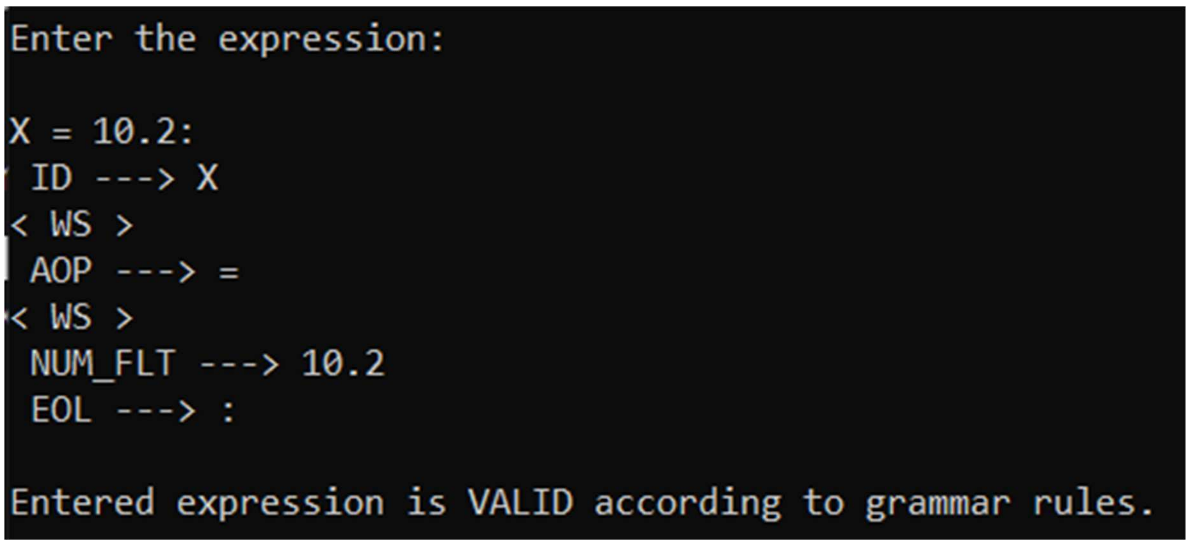
```
Enter the expression:
X21ab =+ Y21ab:
 ID ---> X21ab
< WS >
 BOP ---> =+
< WS >
 ID ---> Y21ab
 EOL ---> :

Entered expression is VALID according to grammar rules.
```



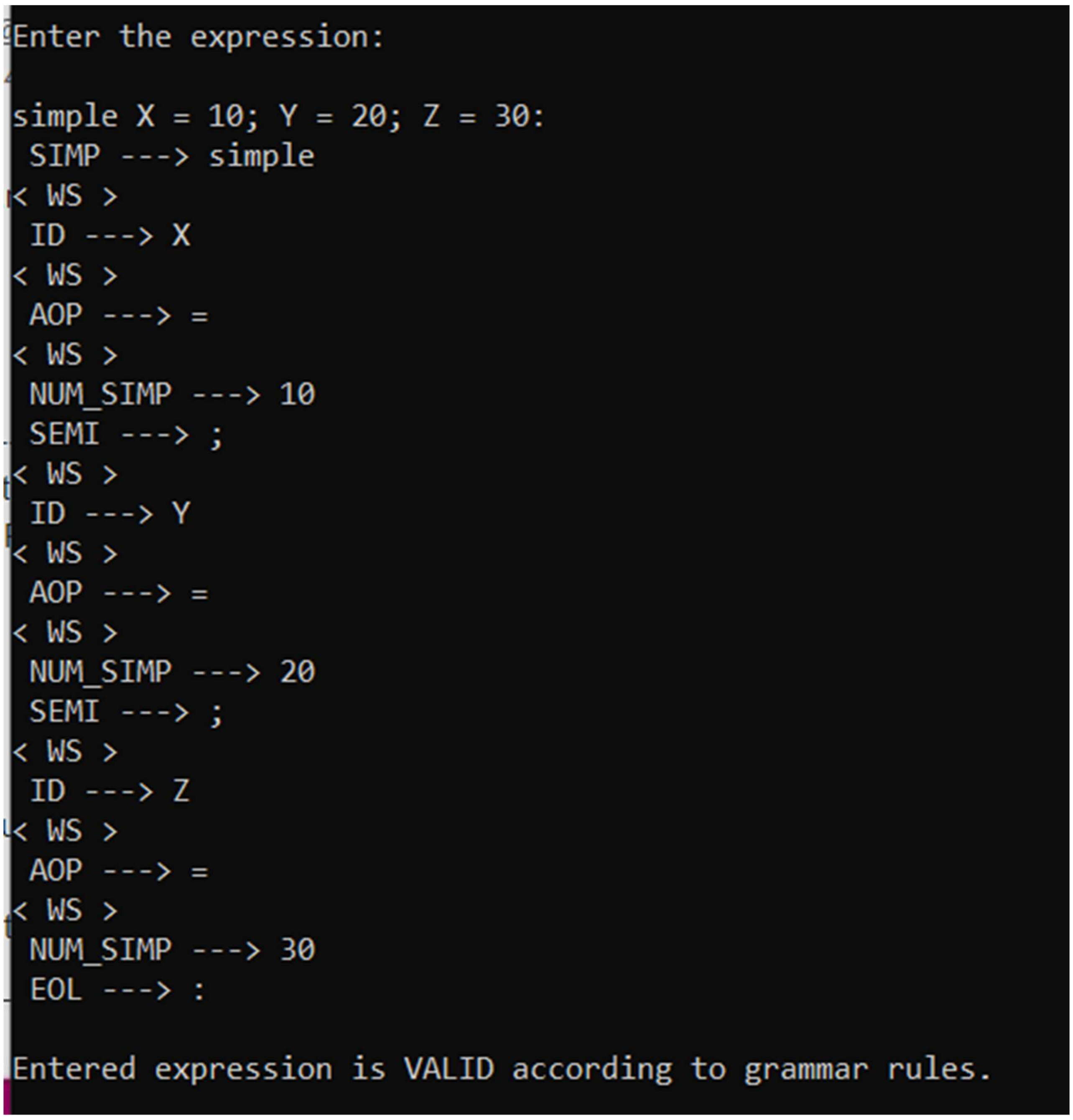
```
Enter the expression:

X21ab =- Y21:
 ID ---> X21ab
< WS >
 BOP ---> =-
< WS >
 ID ---> Y21
 EOL ---> :

Entered expression is VALID according to grammar rules.
```



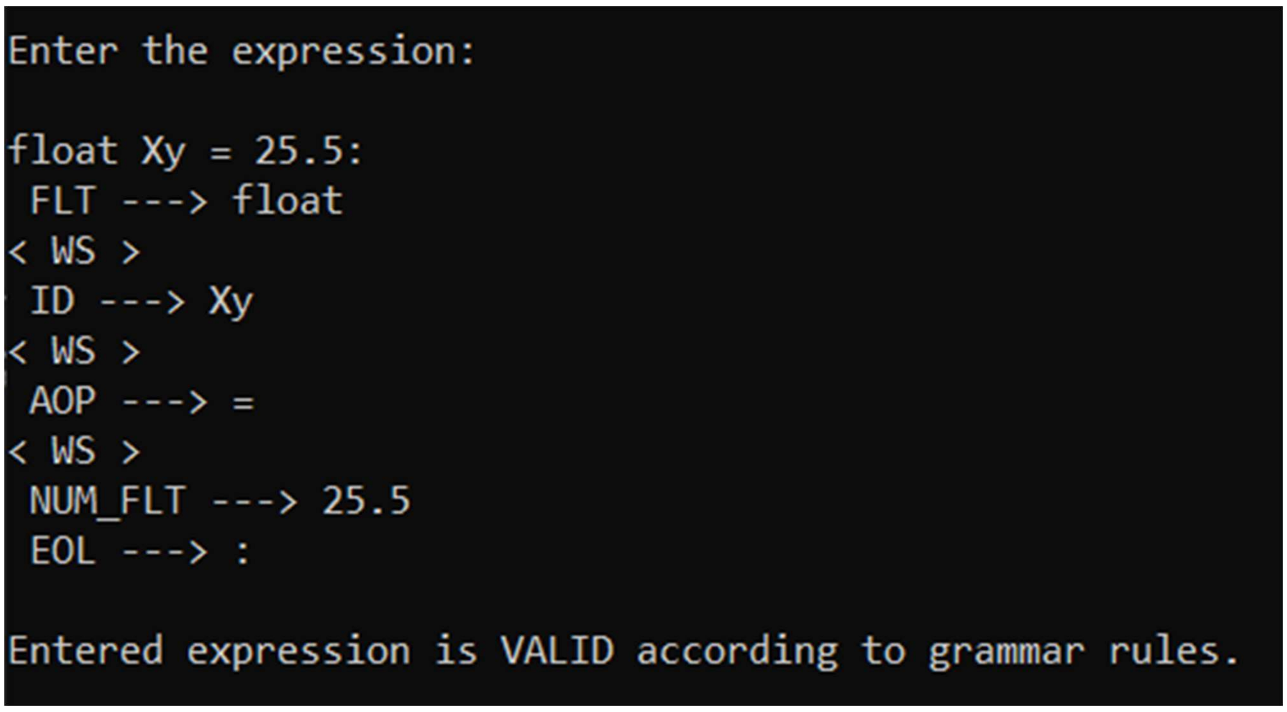
```
Enter the expression:
X = 10:
 ID ---> X
< WS >
 AOP ---> =
< WS >
 NUM_SIMP ---> 10
 EOL ---> :

Entered expression is VALID according to grammar rules.
```



```
Enter the expression:

X = 10.2:
 ID ---> X
< WS >
 AOP ---> =
< WS >
 NUM_FLT ---> 10.2
 EOL ---> :

Entered expression is VALID according to grammar rules.
```

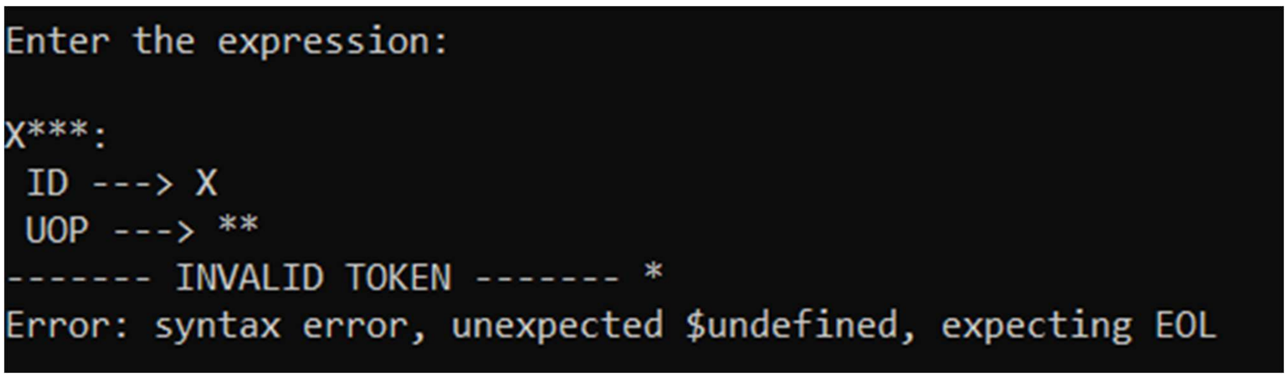


```
Enter the expression:

simple X = 10; Y = 20; Z = 30:
 SIMP ---> simple
< WS >
 ID ---> X
< WS >
 AOP ---> =
< WS >
 NUM_SIMP ---> 10
 SEMI ---> ;
< WS >
 ID ---> Y
< WS >
 AOP ---> =
< WS >
 NUM_SIMP ---> 20
 SEMI ---> ;
< WS >
 ID ---> Z
< WS >
 AOP ---> =
< WS >
 NUM_SIMP ---> 30
 EOL ---> :

Entered expression is VALID according to grammar rules.
```

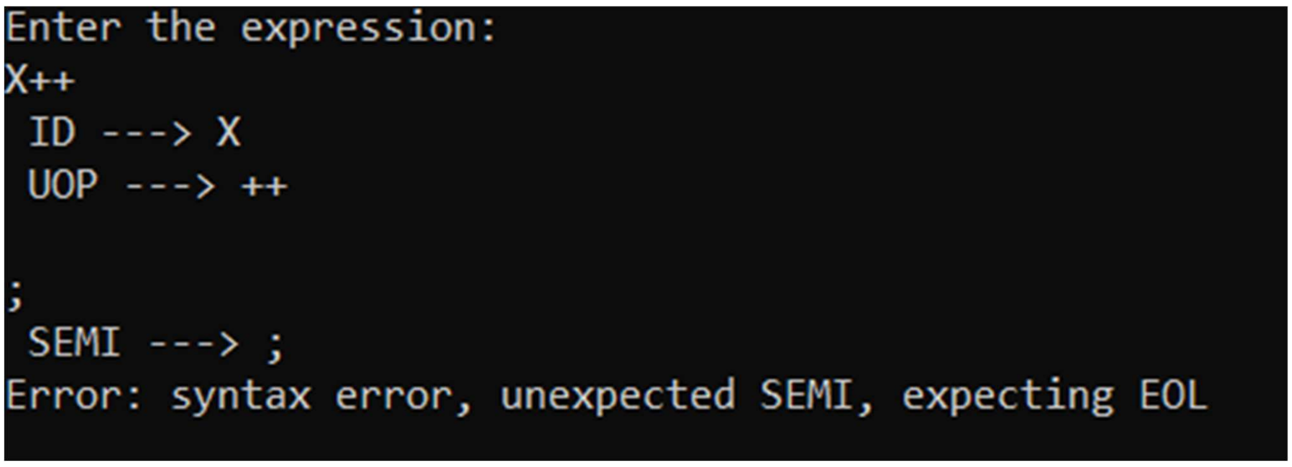


```
Enter the expression:

float Xy = 25.5:
 FLT ---> float
< WS >
 ID ---> Xy
< WS >
 AOP ---> =
< WS >
 NUM_FLT ---> 25.5
 EOL ---> :

Entered expression is VALID according to grammar rules.
```

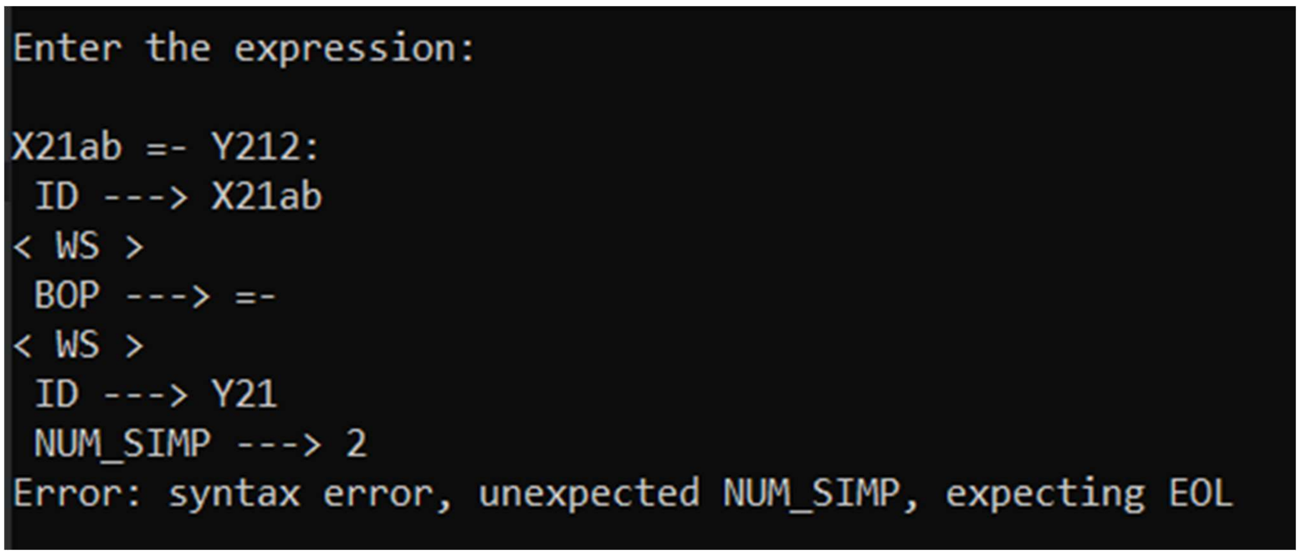
INVALID TESTCASES:



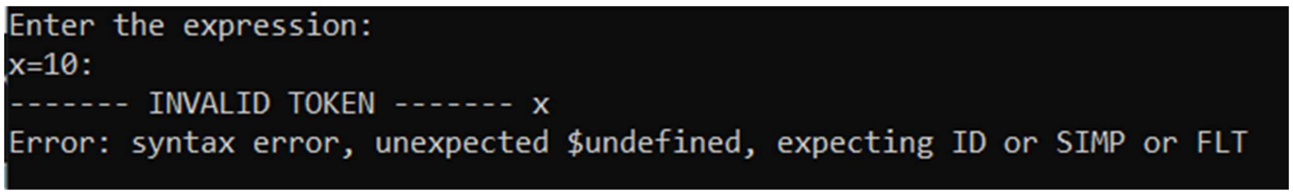
```
Enter the expression:

X***:
 ID ---> X
 UOP ---> **
------- INVALID TOKEN ------- *
Error: syntax error, unexpected $undefined, expecting EOL
```



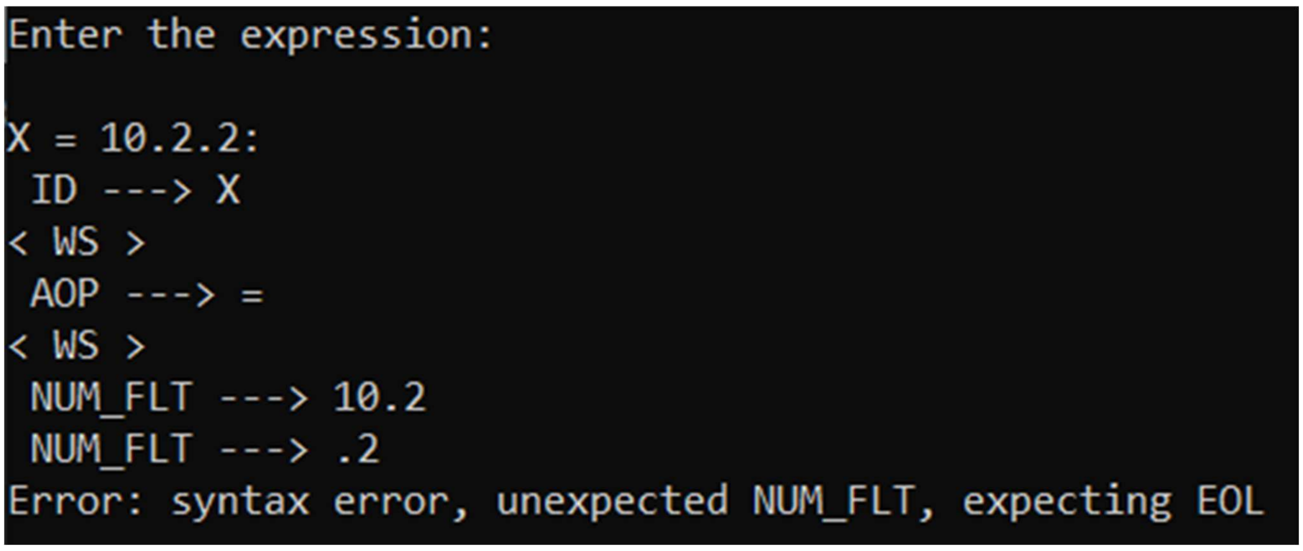
```
Enter the expression:
X++
 ID ---> X
 UOP ---> ++


;
 SEMI ---> ;
Error: syntax error, unexpected SEMI, expecting EOL
```



```
Enter the expression:

X21ab =- Y212:
 ID ---> X21ab
< WS >
 BOP ---> =-
< WS >
 ID ---> Y21
 NUM_SIMP ---> 2
Error: syntax error, unexpected NUM_SIMP, expecting EOL
```



```
Enter the expression:
x=10:
------- INVALID TOKEN ------- x
Error: syntax error, unexpected $undefined, expecting ID or SIMP or FLT
```



```
Enter the expression:

X = 10.2.2:
 ID ---> X
< WS >
 AOP ---> =
< WS >
 NUM_FLT ---> 10.2
 NUM_FLT ---> .2
Error: syntax error, unexpected NUM_FLT, expecting EOL
```

4.0 CONCLUSION

As an IT student we have learned and worded with many programming languages but we have hardly thought the process that is running behind the picture. There is very complex logic that is implemented to make this language that we are using right now. But the subject of Language Translator makes us understand this complex mechanism.

By implementing this project, we have learnt the lexical and parsing phase of compiler design. This project has been using the knowledge that we have gained from college curriculum and from the internet. After doing this project we conclude that we have got more knowledge about how different compilers are working and handling the errors.

We would like to thank Prof. Nikita P. Desai for teaching this interesting subject, for the guidance in the project and to give chance to know how compilers works.