Pantho Haque
Roll: 1907075
Group : B1
Year : 3rd
Semester: 2nd

Writing a program in flex to tokenize the source file containing the source code of my language

Experiment No: 03
Date of performance : 19-09-23
Date of Submission : 03-10-23

Course code: CSE 3212

Computer Science and Engineering

Khulna University of Engineering and Technology, Khulna - 9203

Objective:

1. Learn to perform lexical analysis on our code snippet.
2. Identify and categorize various language constructs.
3. Learn to design our own syntax of a programming language.

Introduction:

Lexical analysis is the first phase of the compilation process, which involves scanning the source code to identify tokens and their attributes. We will perform lexical analysis on a code snippet to gain insights into its structure and content. Determining the comments, variable declarations, conditional statements, various types of operators , functions etc.

Basic Structure:

== == indicating a single line comment

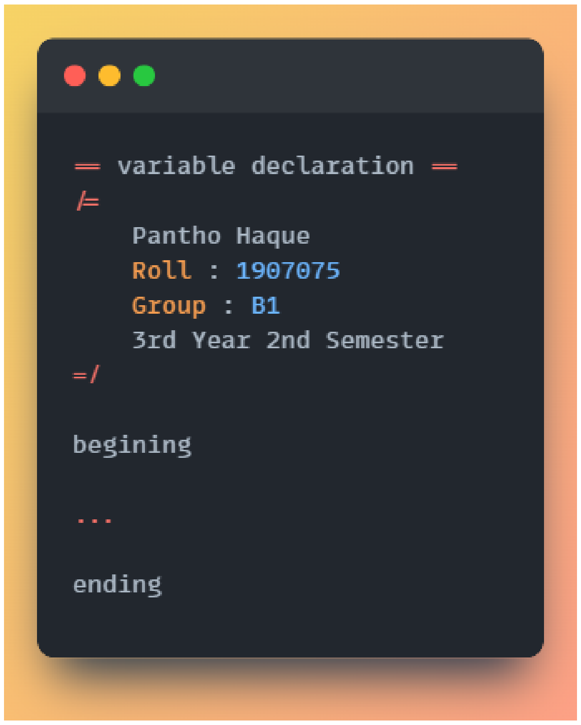
/=

... indicating a multiline comment

=/

beginning the start of the code

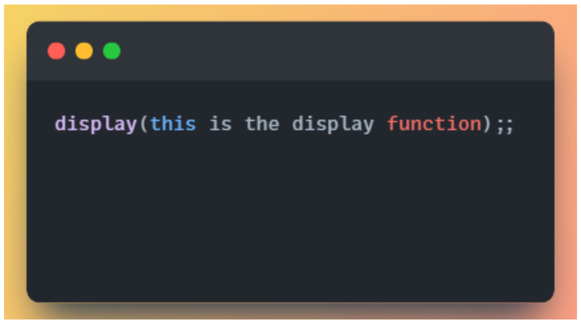
ending the end of the code



```
== variable declaration ==
/=
    Pantho Haque
    Roll : 1907075
    Group : B1
    3rd Year 2nd Semester
=/

begining

...

ending
```

Display:

This function displays any data we give to it.

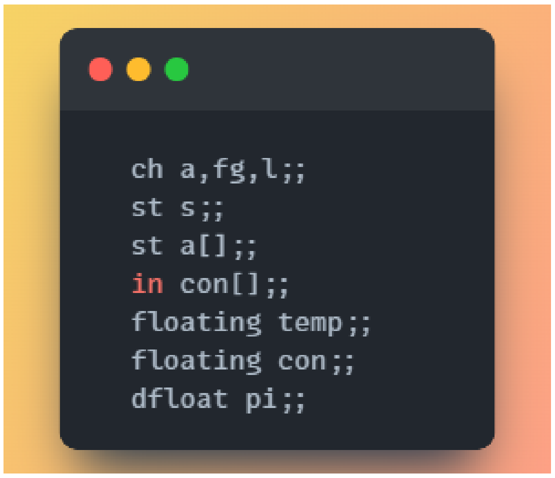


```
display(this is the display function);;
```

ID and Type Design:

Variables have 5 data types .

1. **ch** will store a single character.
2. **st** will store a string.
3. **in** will store integer
4. **floating** will store floating.
   point number
5. **dfloat** will store double.
   Number



```
ch a,fg,l;;
st s;;
st a[];;
in con[];;
floating temp;;
floating con;;
dfloat pi;;
```

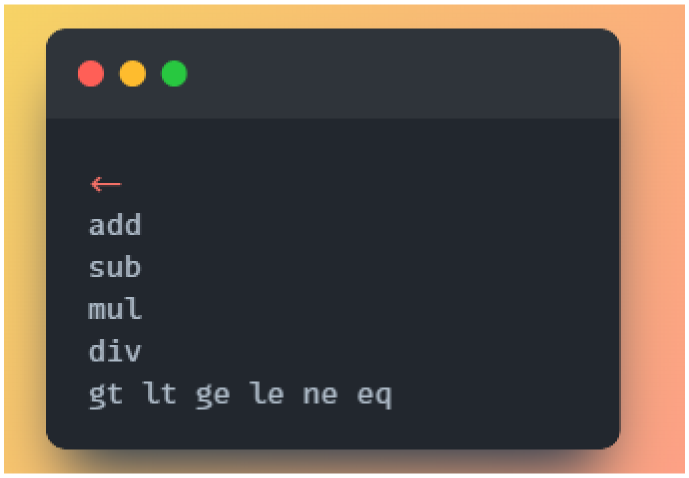
Multiple variables can be declared in a single line separated by **comma**.

Array can be declared by adding **[]** after the variable name.

No duplicate variables are allowed here.

Operator Handling:

- Assignment operator (<-)
- Arithmetic Operators
  1.**add** for Addition
  2.**sub** for Subtraction
  3.**mul** for Multiplication
  4.**div** for Division
- Relational Operators
  1. **gt** for greater than
  2.**lt** for less than
  3. **ge** for greater equal
  4.**le** for less equal
  5.**ne** for not equal
  6.**eq** for equal to



```
←
add
sub
mul
div
gt lt ge le ne eq
```

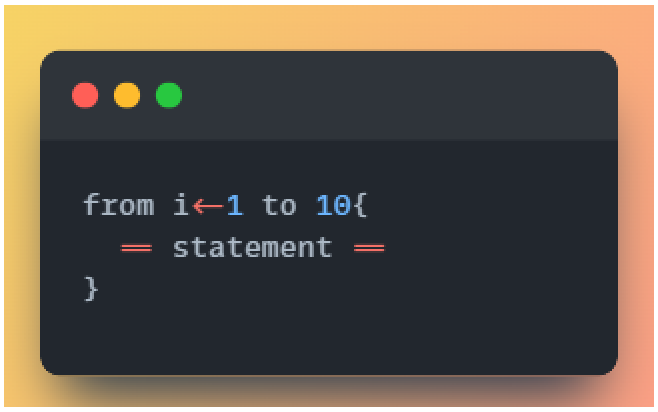
from-to Loop:

Loop is used to make an iteration through a data structure like array or string

from and **to** keywords are used to maintain the iteration sequence in loop

RegExp:

```
from[ ]{var}[ ]*<-[0-9]+[ ]+to[ ]+[0-9]+\{[^}]*\}
```



```
from i←1 to 10{
  == statement ==
}
```

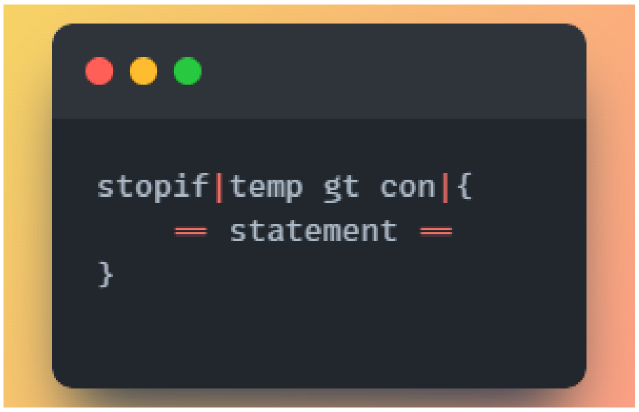
stopif Loop:

This loop is used to go through the statements again and again till a specific condition is satisfied. Once the condition returns false the loop will stop.

| | is holding the condition of this loop.

RegExp:

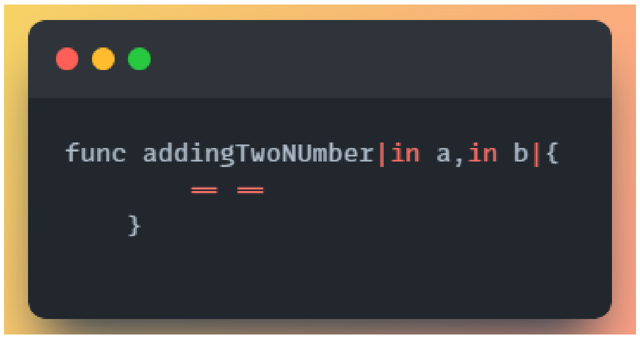
```
stopif\|{var}[ ]{relop}[ ]{var}\|\{[^\}]*\}
```



```
stopif|temp gt con|{
     == statement ==
}
```

Function:

To make our code less redundance we use function which is identified by **func** keyword **| |** holding the parameters of our function



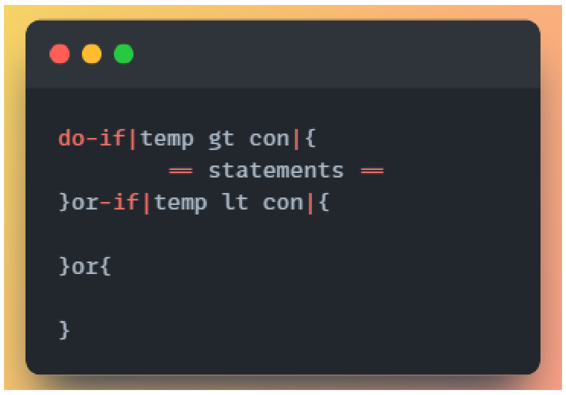
```
func addingTwoNUmber|in a,in b|{
         == ==
    }
```

RegExp:

```
func[ ]+{var}[ ]*\|({dataT}[ ]+{var})(,{dataT}[ ]+{var})*\|\{[^\}]*\}
```

do-or-if conditional statement:

To do some action after taking a decision, we use conditional statement. The first condition will be initiated using **do-if** keyword. Then we use **or-if** keyword to check further conditions. Lastly the default action will be under **or** block.



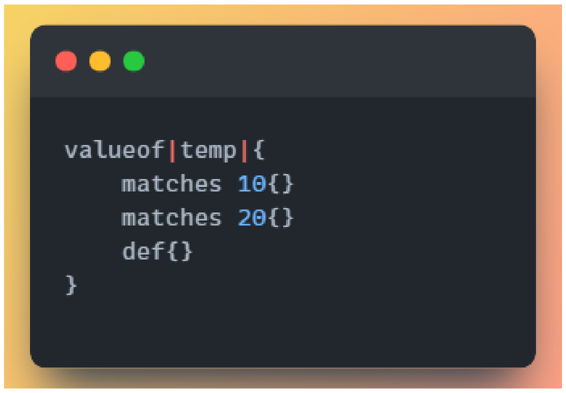
```
do-if|temp gt con|{
        == statements ==
}or-if|temp lt con|{

}or{

}
```

RegExp:

```
do-if\|{var}[]{relop}[]{var}(\|\{)[^\}]*\}(or-
if\|{var}[]{relop}[]{var}\|\{[^\}]*\})?(or\{[^\}]*\})?
```

Value-of conditional statement:

The value of conditional statement matches the each value given with the variable encapsulated between **||** **.** Block with **def** keyword will run default if no value has been matched.



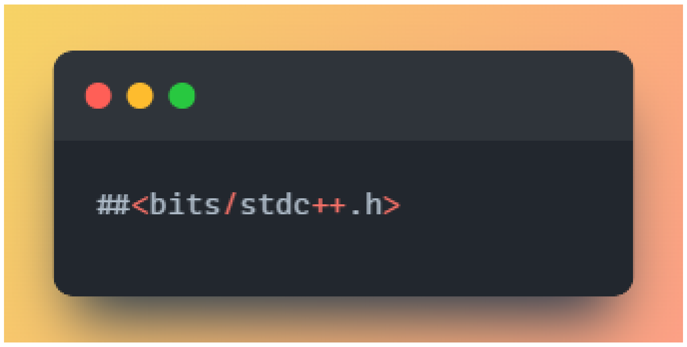
```
valueof|temp|{
    matches 10{}
    matches 20{}
    def{}
}
```

RegExp:

```
valueof\|{var}\|\{{ws}(matches[][0-
9]+\{[^}]*\}{ws})+def\{[^}]*\}{ws}\}
```

Directives:

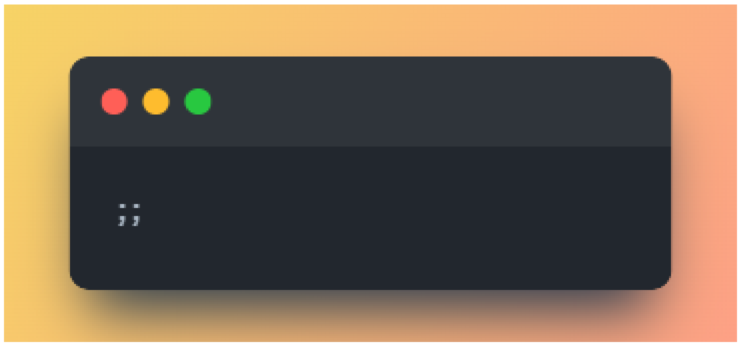
##<> indicates importing a header file to our program.



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

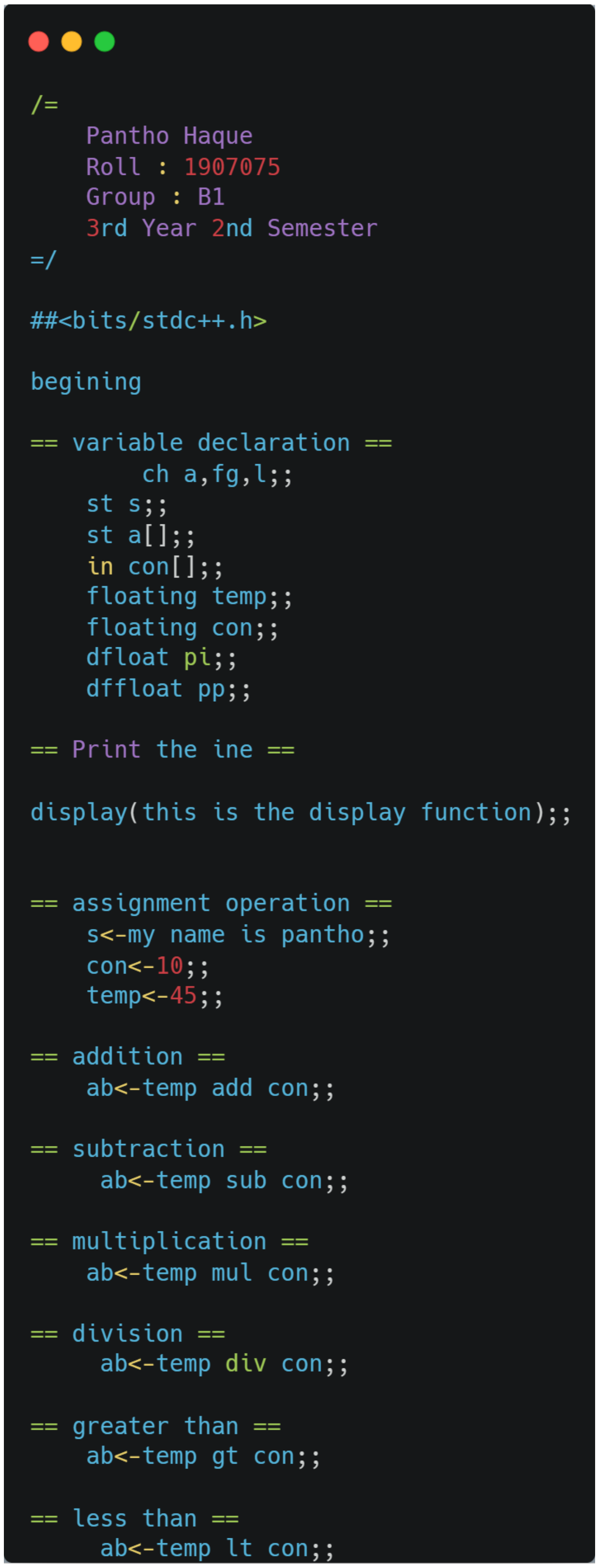
End of a statement:

;; will indicate the end of each statement

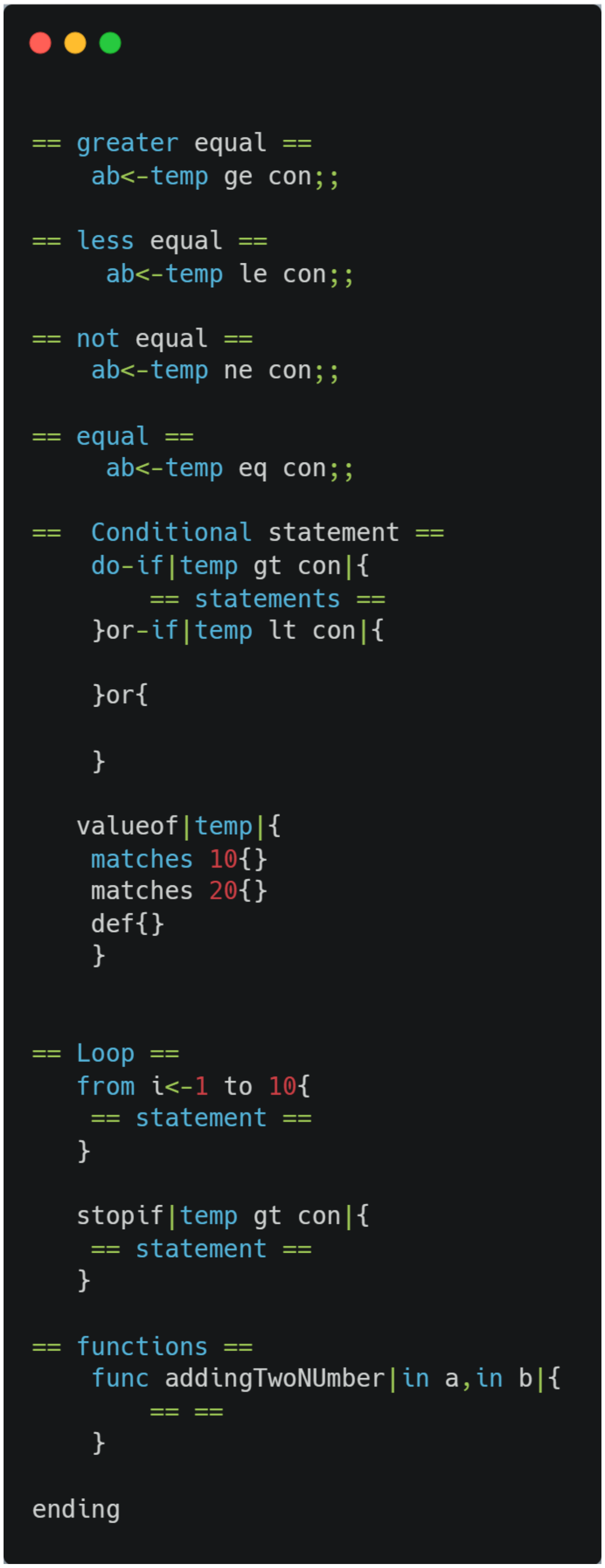


```
;;
```

Source code:

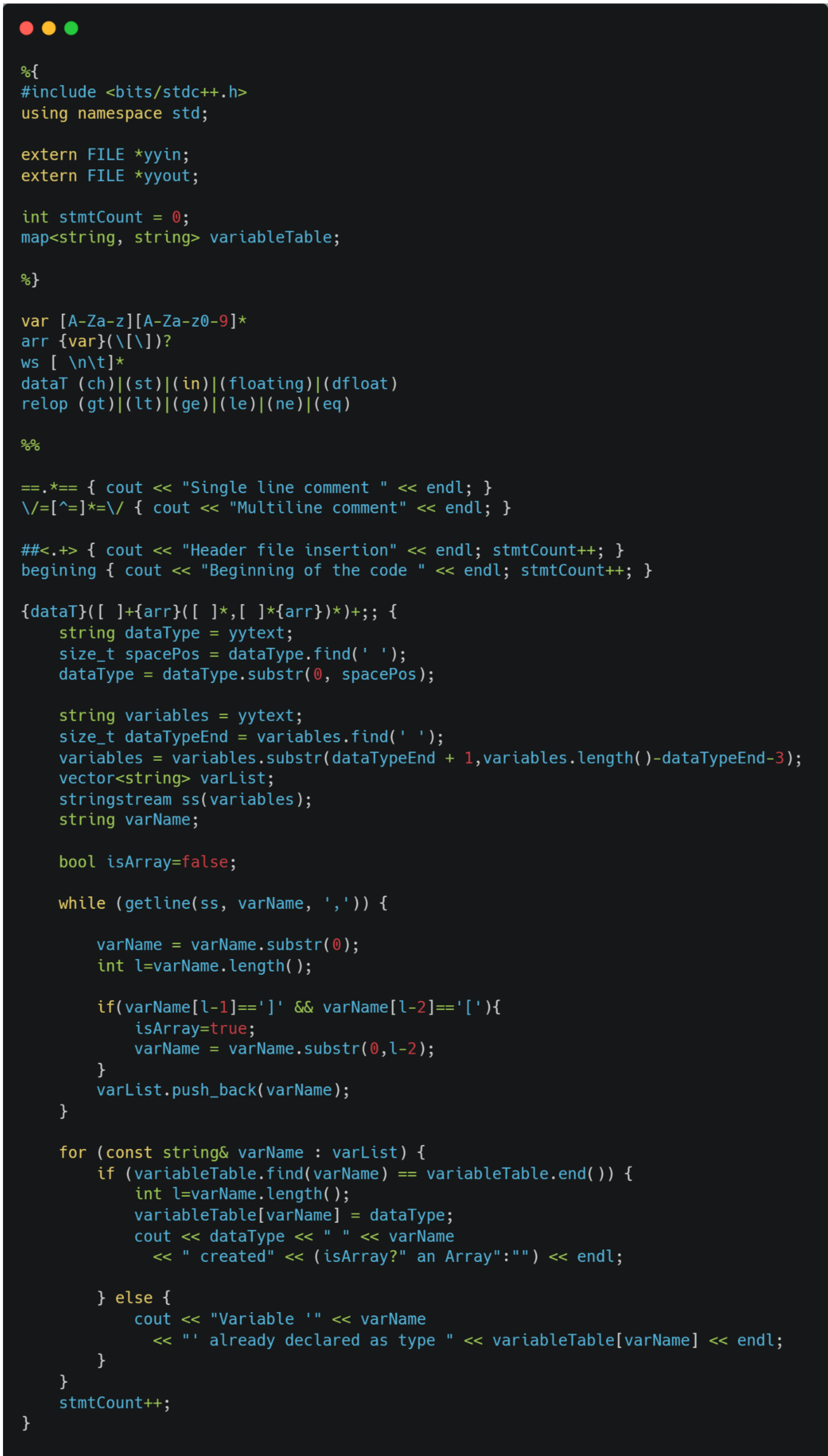


```
/=
    Pantho Haque
    Roll : 1907075
    Group : B1
    3rd Year 2nd Semester
=/

##<bits/stdc++.h>

begining

== variable declaration ==
        ch a,fg,l;;
    st s;;
    st a[];;
    in con[];;
    floating temp;;
    floating con;;
    dfloat pi;;
    dffloat pp;;

== Print the ine ==

display(this is the display function);;


== assignment operation ==
    s<-my name is pantho;;
    con<-10;;
    temp<-45;;

== addition ==
    ab<-temp add con;;

== subtraction ==
     ab<-temp sub con;;

== multiplication ==
    ab<-temp mul con;;

== division ==
     ab<-temp div con;;

== greater than ==
    ab<-temp gt con;;

== less than ==
     ab<-temp lt con;;
```

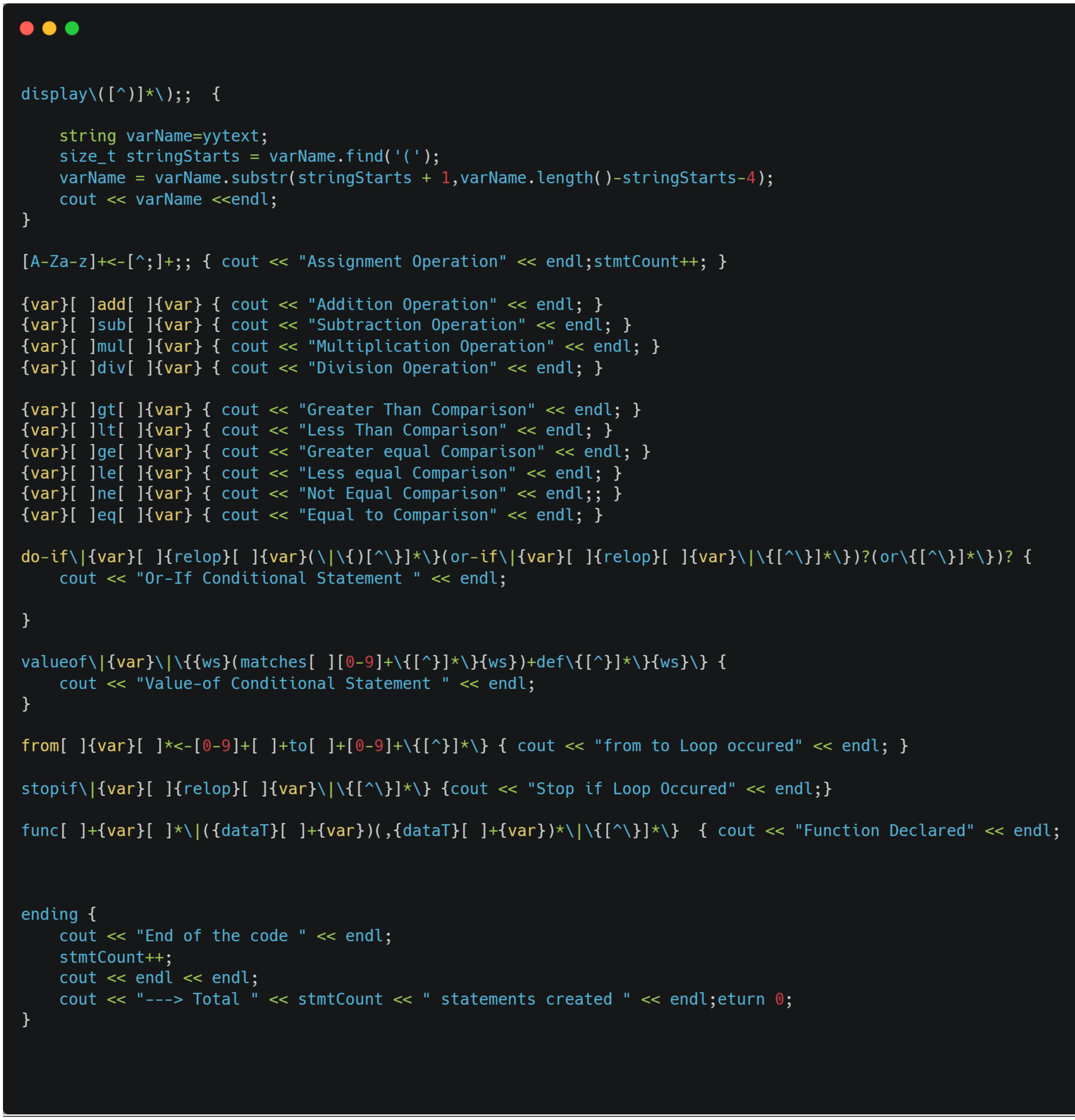


```
== greater equal ==
    ab<-temp ge con;;

== less equal ==
     ab<-temp le con;;

== not equal ==
    ab<-temp ne con;;

== equal ==
     ab<-temp eq con;;

==  Conditional statement ==
    do-if|temp gt con|{
        == statements ==
    }or-if|temp lt con|{

    }or{

    }

   valueof|temp|{
    matches 10{}
    matches 20{}
    def{}
    }


== Loop ==
   from i<-1 to 10{
    == statement ==
   }

   stopif|temp gt con|{
    == statement ==
   }

== functions ==
    func addingTwoNUmber|in a,in b|{
        == ==
    }

ending
```

Flex(.l) file:

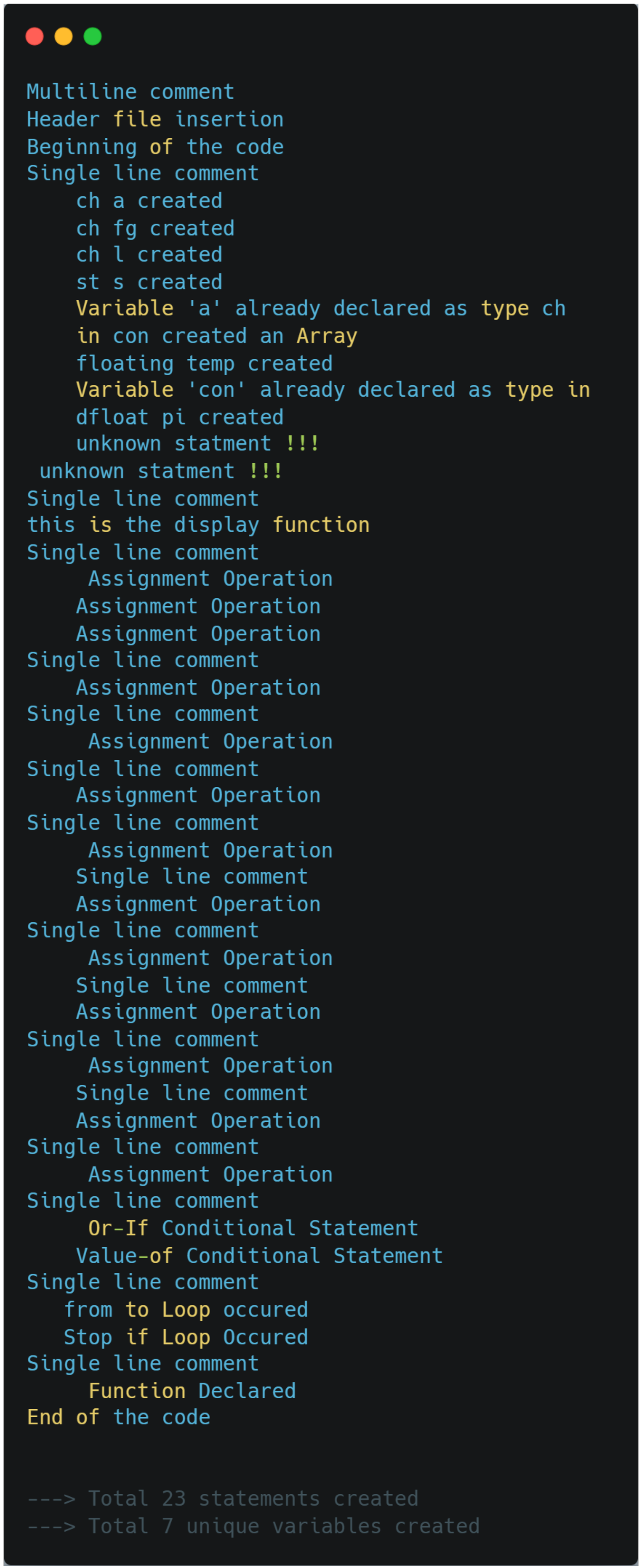


```
%{
#include <bits/stdc++.h>
using namespace std;

extern FILE *yyin;
extern FILE *yyout;

int stmtCount = 0;
map<string, string> variableTable;

%}

var [A-Za-z][A-Za-z0-9]*
arr {var}(\[\])?
ws [ \n\t]*
dataT (ch)|(st)|(in)|(floating)|(dfloat)
relop (gt)|(lt)|(ge)|(le)|(ne)|(eq)

%%

==.*== { cout << "Single line comment " << endl; }
\/=[^=]*=\/ { cout << "Multiline comment" << endl; }

##<.+> { cout << "Header file insertion" << endl; stmtCount++; }
begining { cout << "Beginning of the code " << endl; stmtCount++; }

{dataT}([ ]+{arr}([ ]*,[ ]*{arr})*)+;; {
    string dataType = yytext;
    size_t spacePos = dataType.find(' ');
    dataType = dataType.substr(0, spacePos);

    string variables = yytext;
    size_t dataTypeEnd = variables.find(' ');
    variables = variables.substr(dataTypeEnd + 1,variables.length()-dataTypeEnd-3);
    vector<string> varList;
    stringstream ss(variables);
    string varName;

    bool isArray=false;

    while (getline(ss, varName, ',')) {

        varName = varName.substr(0);
        int l=varName.length();

        if(varName[l-1]==']' && varName[l-2]=='['){
            isArray=true;
            varName = varName.substr(0,l-2);
        }
        varList.push_back(varName);
    }

    for (const string& varName : varList) {
        if (variableTable.find(varName) == variableTable.end()) {
            int l=varName.length();
            variableTable[varName] = dataType;
            cout << dataType << " " << varName
              << " created" << (isArray?" an Array":"") << endl;

        } else {
            cout << "Variable '" << varName
              << "' already declared as type " << variableTable[varName] << endl;
        }
    }
    stmtCount++;
}
```



```
display\([^)]*\);;  {

    string varName=yytext;
    size_t stringStarts = varName.find('(');
    varName = varName.substr(stringStarts + 1,varName.length()-stringStarts-4);
    cout << varName <<endl;
}

[A-Za-z]+<-[^;]+;; { cout << "Assignment Operation" << endl;stmtCount++; }

{var}[ ]add[ ]{var} { cout << "Addition Operation" << endl; }
{var}[ ]sub[ ]{var} { cout << "Subtraction Operation" << endl; }
{var}[ ]mul[ ]{var} { cout << "Multiplication Operation" << endl; }
{var}[ ]div[ ]{var} { cout << "Division Operation" << endl; }

{var}[ ]gt[ ]{var} { cout << "Greater Than Comparison" << endl; }
{var}[ ]lt[ ]{var} { cout << "Less Than Comparison" << endl; }
{var}[ ]ge[ ]{var} { cout << "Greater equal Comparison" << endl; }
{var}[ ]le[ ]{var} { cout << "Less equal Comparison" << endl; }
{var}[ ]ne[ ]{var} { cout << "Not Equal Comparison" << endl;; }
{var}[ ]eq[ ]{var} { cout << "Equal to Comparison" << endl; }

do-if\|{var}[ ]{relop}[ ]{var}(\|\{)[^\}]*\}(or-if\|{var}[ ]{relop}[ ]{var}\|\{[^\}]*\})?(or\{[^\}]*\})? {
    cout << "Or-If Conditional Statement " << endl;

}

valueof\|{var}\|\{{ws}(matches[ ][0-9]+\{[^}]*\}{ws})+def\{[^}]*\}{ws}\} {
    cout << "Value-of Conditional Statement " << endl;
}

from[ ]{var}[ ]*<-[0-9]+[ ]+to[ ]+[0-9]+\{[^}]*\} { cout << "from to Loop occured" << endl; }

stopif\|{var}[ ]{relop}[ ]{var}\|\{[^\}]*\} {cout << "Stop if Loop Occured" << endl;}

func[ ]+{var}[ ]*\|({dataT}[ ]+{var})(,{dataT}[ ]+{var})*\|\{[^\}]*\}  { cout << "Function Declared" << endl;



ending {
    cout << "End of the code " << endl;
    stmtCount++;
    cout << endl << endl;
    cout << "---> Total " << stmtCount << " statements created " << endl;eturn 0;
}
```

Output File:



```
Multiline comment
Header file insertion
Beginning of the code
Single line comment
    ch a created
    ch fg created
    ch l created
    st s created
    Variable 'a' already declared as type ch
    in con created an Array
    floating temp created
    Variable 'con' already declared as type in
    dfloat pi created
    unknown statment !!!
 unknown statment !!!
Single line comment
this is the display function
Single line comment
     Assignment Operation
    Assignment Operation
    Assignment Operation
Single line comment
    Assignment Operation
Single line comment
     Assignment Operation
Single line comment
    Assignment Operation
Single line comment
     Assignment Operation
    Single line comment
    Assignment Operation
Single line comment
     Assignment Operation
    Single line comment
    Assignment Operation
Single line comment
     Assignment Operation
    Single line comment
    Assignment Operation
Single line comment
     Assignment Operation
Single line comment
     Or-If Conditional Statement
    Value-of Conditional Statement
Single line comment
   from to Loop occured
   Stop if Loop Occured
Single line comment
     Function Declared
End of the code


---> Total 23 statements created
---> Total 7 unique variables created
```

Discussion:

The lexical analysis phase source code is scanned and divided into distinct tokens. These tokens form the foundation for subsequent phases, including parsing and semantic analysis. Here we have covered token identification and categorization like variables, Data types, Operators, Control Structures, Comments, Functions. We must deal with the whitespaces and indentations.

Conclusion:

The lexical analysis phase is a vital component of the compilation process. Our implementation successfully identified and categorized various tokens, including variables, data types, operators, comments, and control structures. It provided valuable feedback to the user, promoting code quality and understanding.