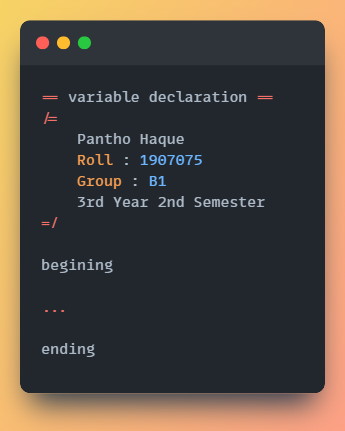
****

**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

**/=**

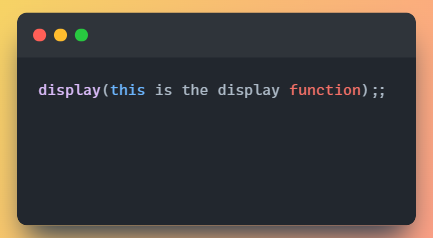
… indicating a multiline

comment

**=/**

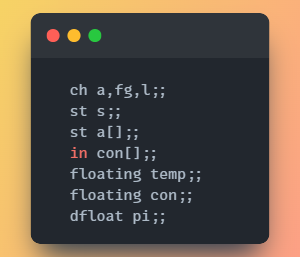
**beginning** the start of the code

**ending** the end of the code



**Display:**

This function displays any data we give to it.

**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

1. **dfloat** will store double.

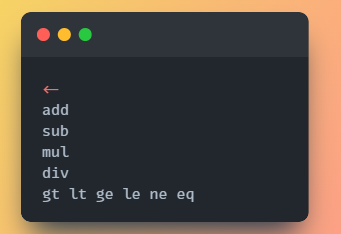
Number

*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

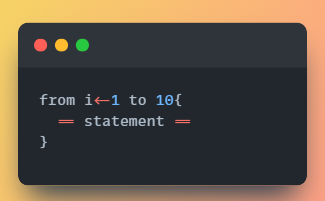
2.**lt** for less than

3. **ge** for greater equal

4.**le** for less equal

5.**ne** for not equal

6.**eq** for equal to

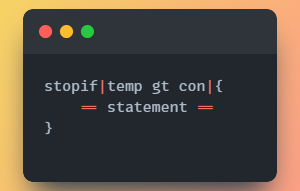
**from-to Loop:**

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]+\{[^}]\*\}

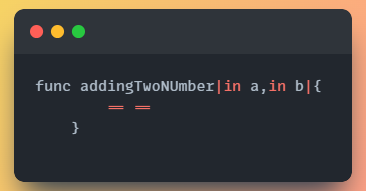
**stopif Loop:**

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:**

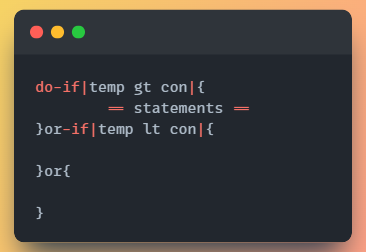
stopif\|{var}[ ]{relop}[ ]{var}\|\{[^\}]\*\}

**Function:**

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

**RegExp:**

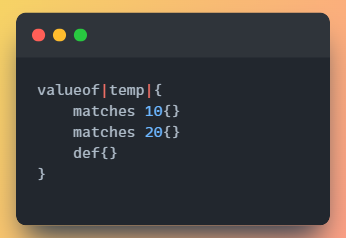
func[ ]+{var}[ ]\*\|({dataT}[ ]+{var})(,{dataT}[ ]+{var})\*\|\{[^\}]\*\}

**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.

**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.

**RegExp:**

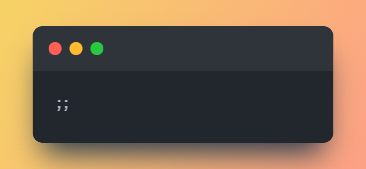
valueof\|{var}\|\{{ws}(matches[][0-9]+\{[^}]\*\}{ws})+def\{[^}]\*\}{ws}\}

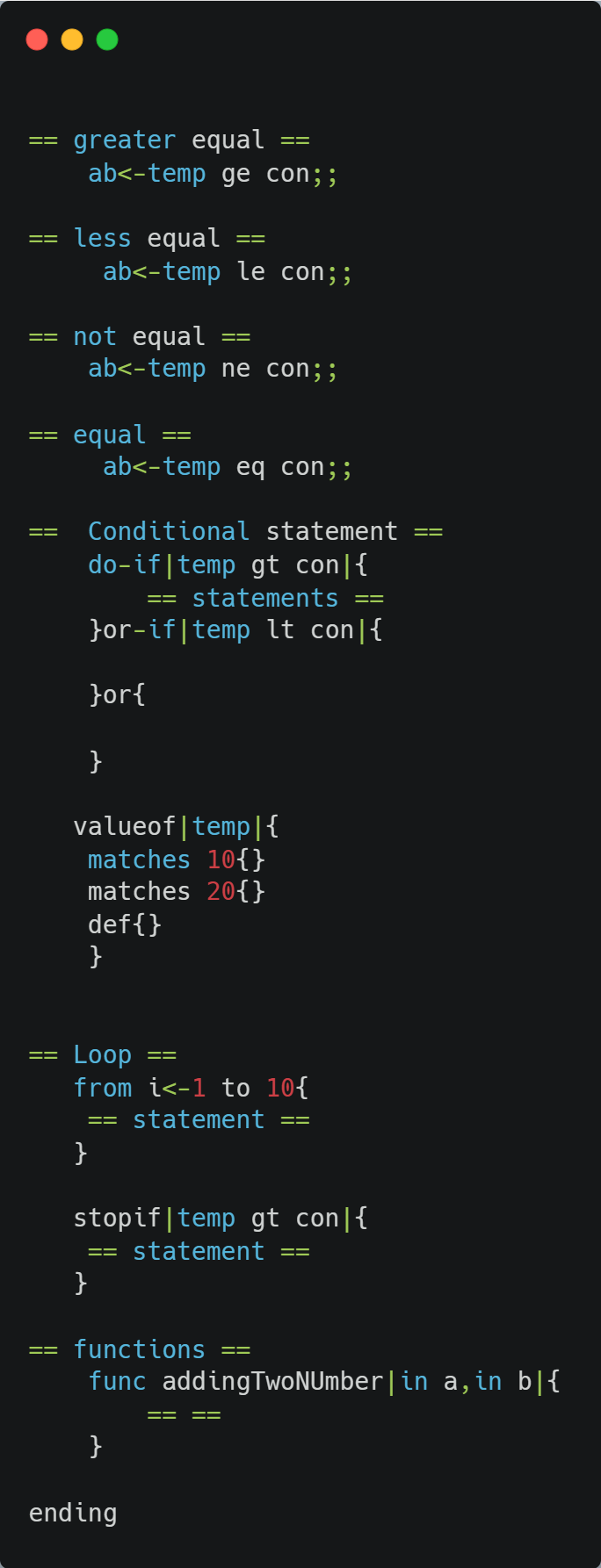
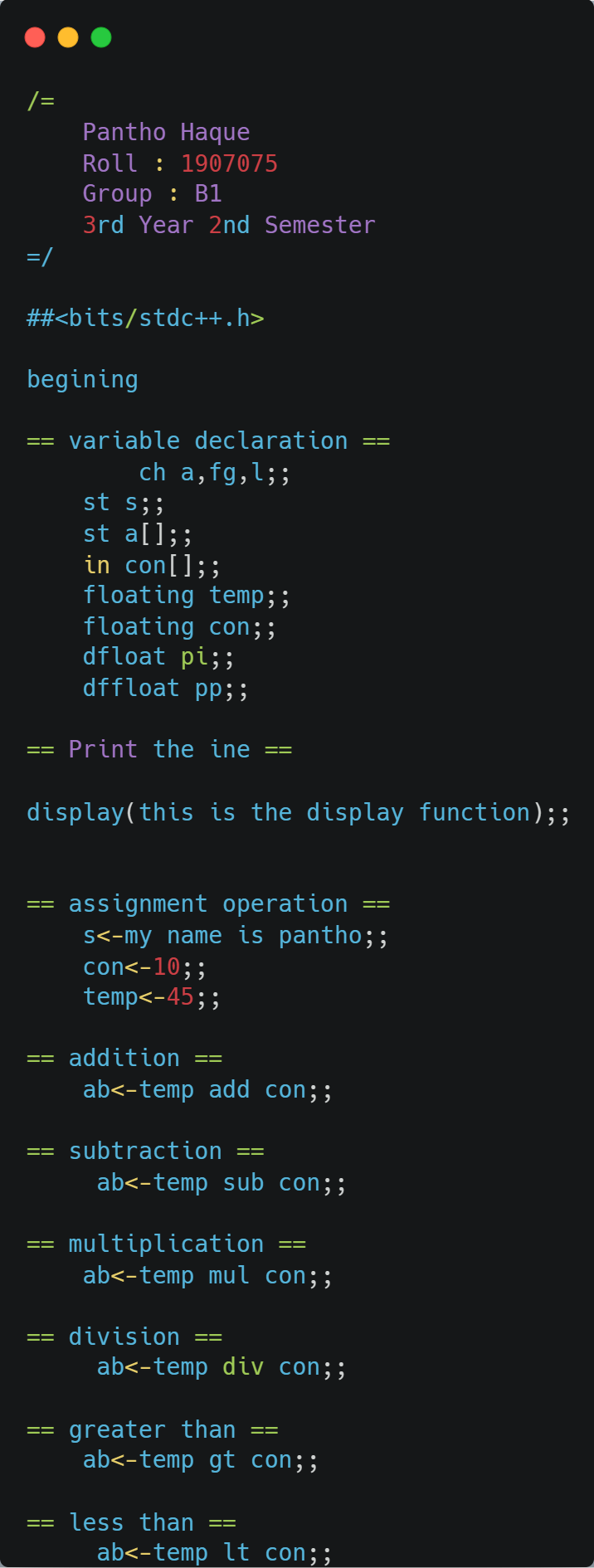
**Directives:**

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

**End of a statement:**

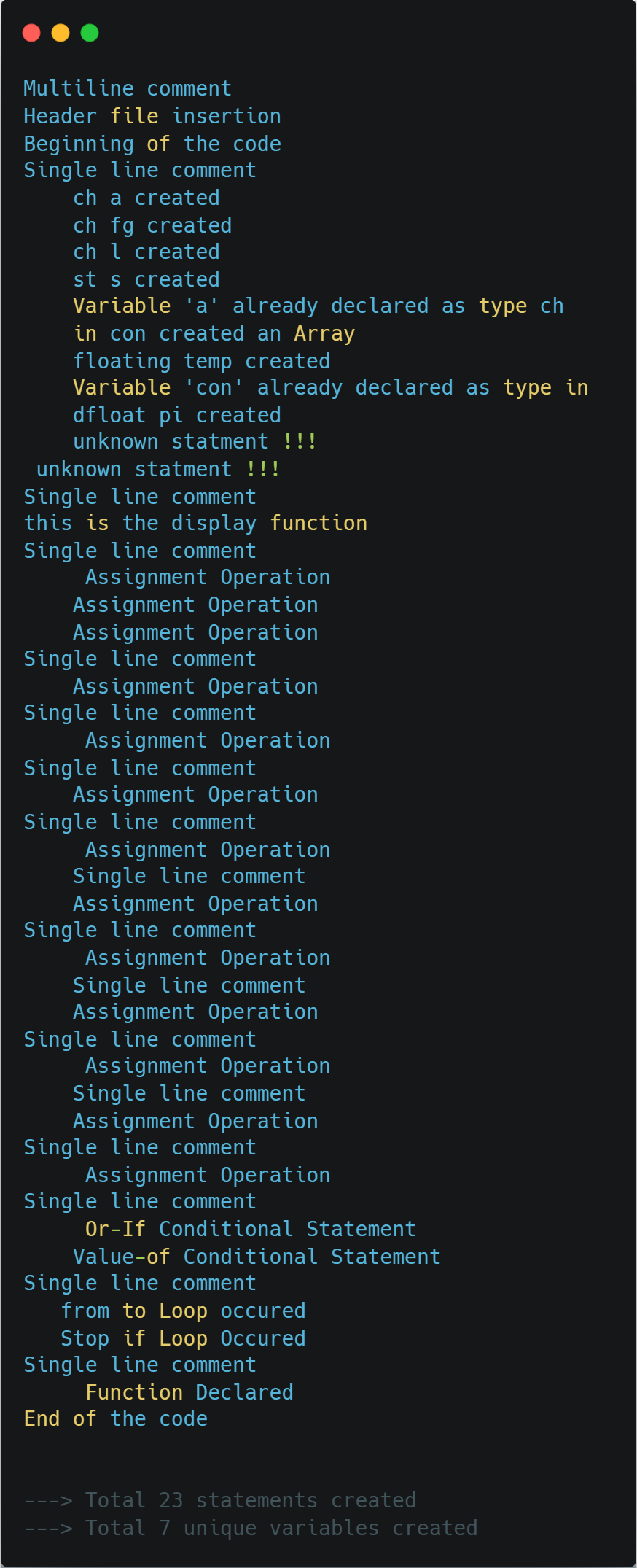
**;;** will indicate the end of each statement



**Source code:**

**Flex(.l) file:**

****

**Output File:**

**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.