

**PROJECT REPORT**

**Compiler Construction Lab**

**CSL 323**

Student Compiler

Group Members:

Ali Gauhar (02-134202-006)

BSCS 5A

Contents

[Introduction 3](#_Toc75385514)

[Objectives 3](#_Toc75385515)

[System Analysis 4](#_Toc75385519)

[Language Description 7](#_Toc75385520)

[Context Free Grammar 11](#_Toc75385521)

[Output](#_Toc75385523)

[Work Flow Diagram 15](#_Toc75385524)

[References 17](#_Toc75385526)

# Introduction

Student compiler is a user friendly language compiler in which our describe language will be use through which, we will first implement lexical analyzer, in which we will tokenize the input and after that syntax analyzer will be implement in which syntax of the input will be check through which we will know that the input is according to our describe language then semantic analyzer will ensure that the declarations and statements of a program are semantically correct.

# Objectives

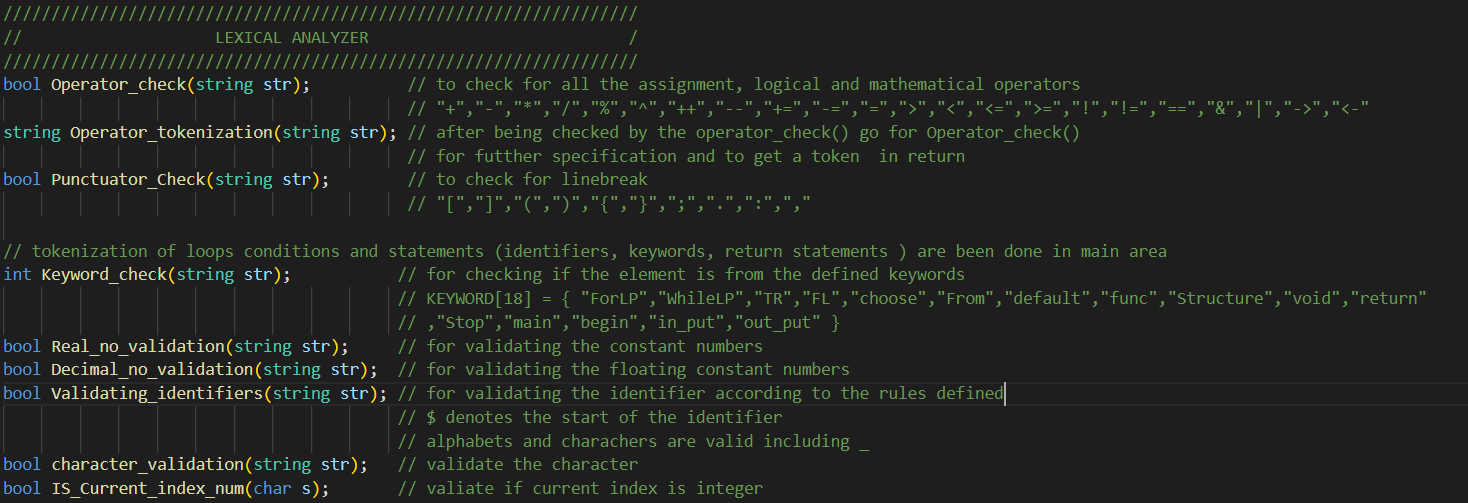
To build a compiler for primary students where very small number of keywords will be present in the programing language and the children would be able to play with them without being lost into logics, such idea can help in building the basic concept of children that how the language works and how code because whenever a person starts learning coding his primary question is where to start and how much time I need to learn the language and which language should I learn so we gathered the basic concept from the languages and made a language that will help the person in quick starting of any language

# System Analysis

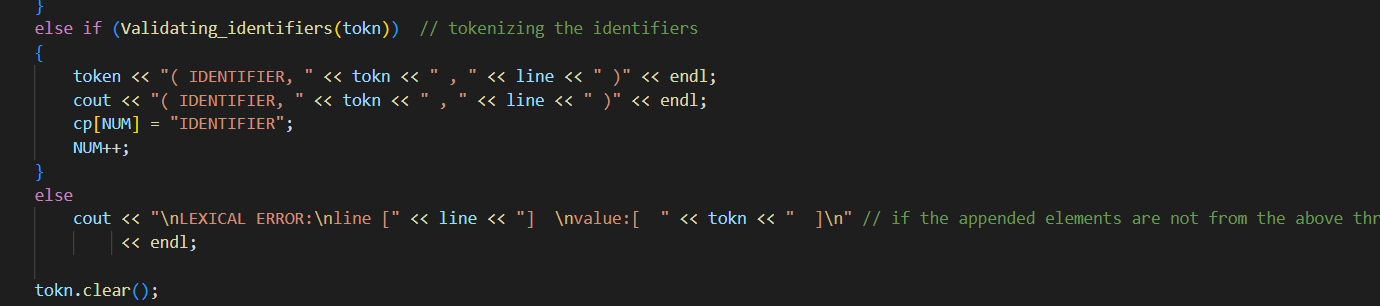
* Lexer Overview

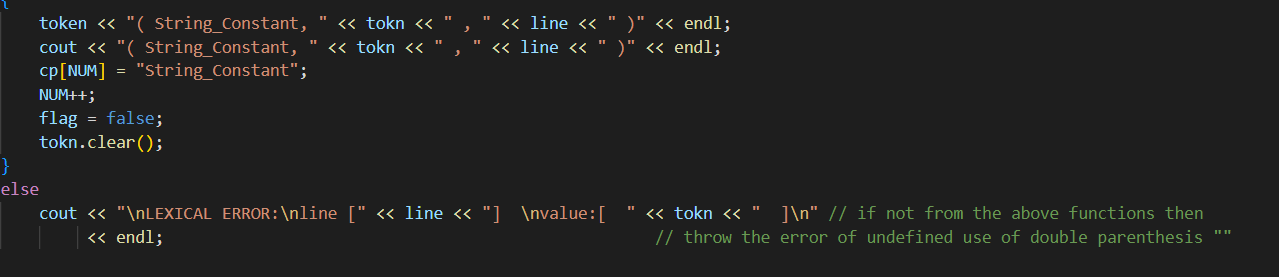
The first module of our compiler is called the *lexer*. Given a string of code, it will iterate character by character to do two things: decide where each token starts/stops and what type of token it is. If the lexer is unable to do this, then it will report an error for an invalid token.

**Some Lexical Analyzer Functions:**



**Errors:**



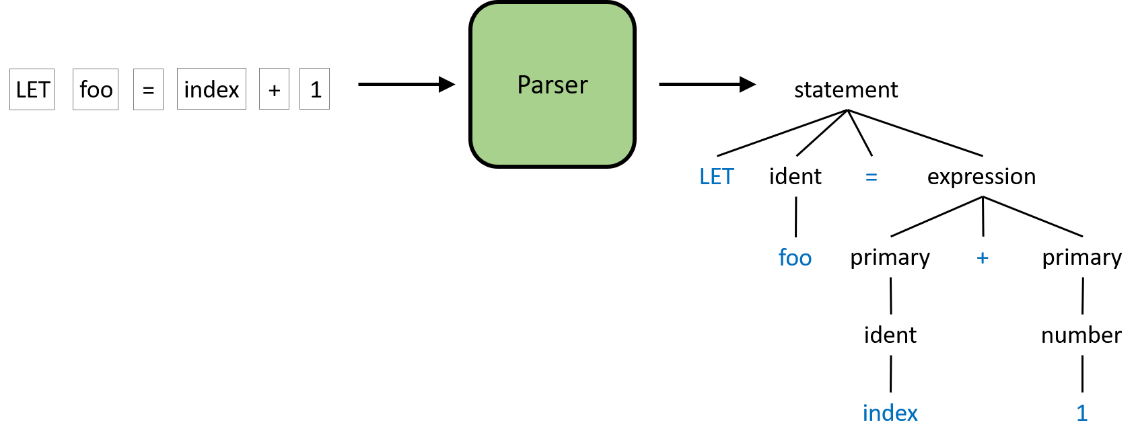




* Syntax overview:

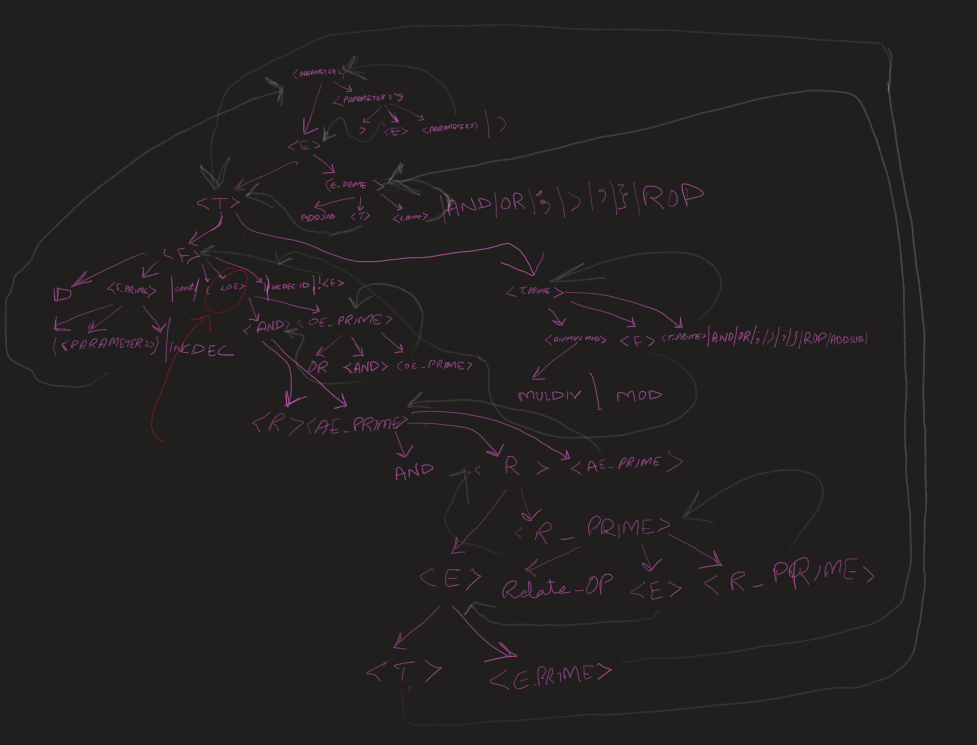
Parser:

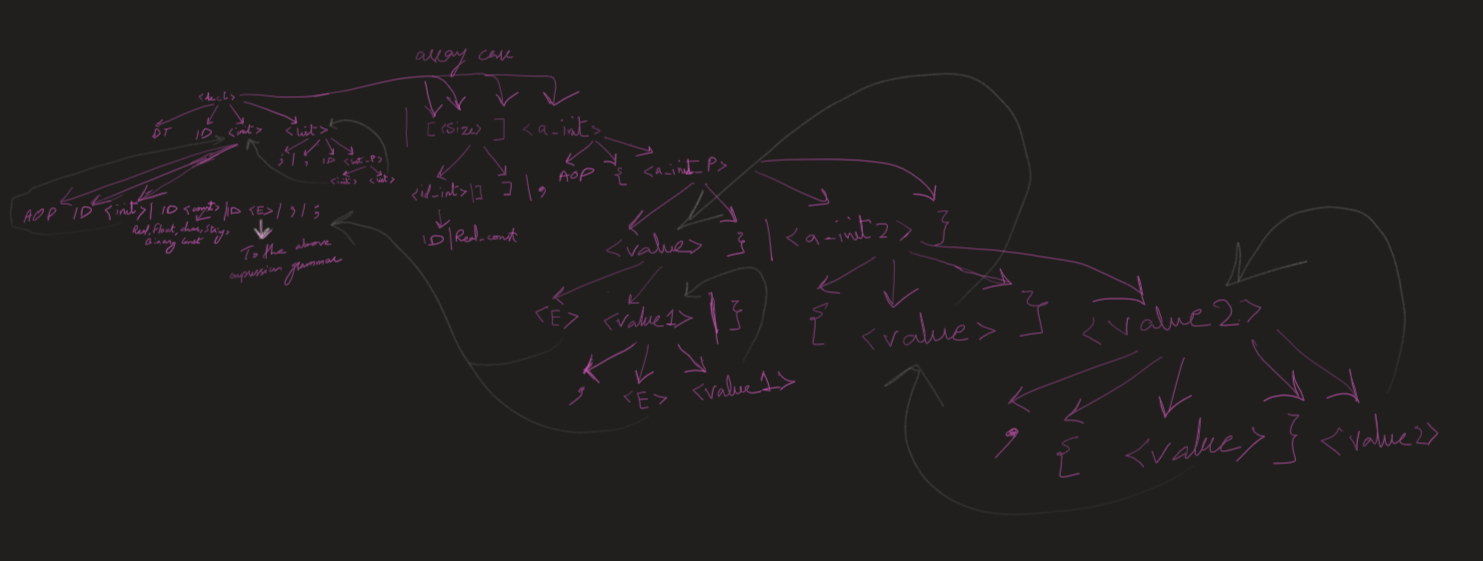
The parser is the component that will make sure the code follows the correct syntax. It does this by looking at the tokens, one at a time, and deciding if the ordering is legal as defined by our language.

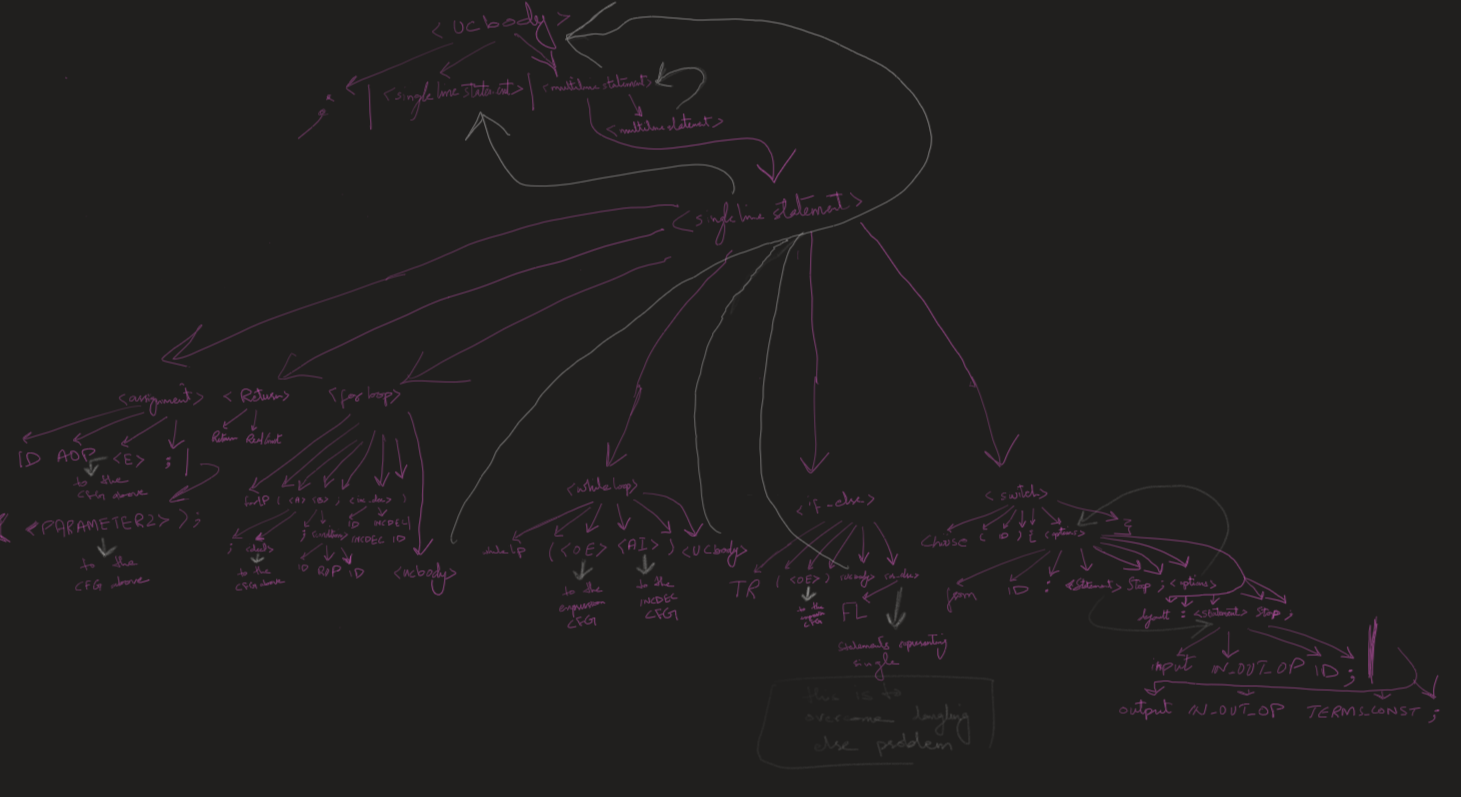


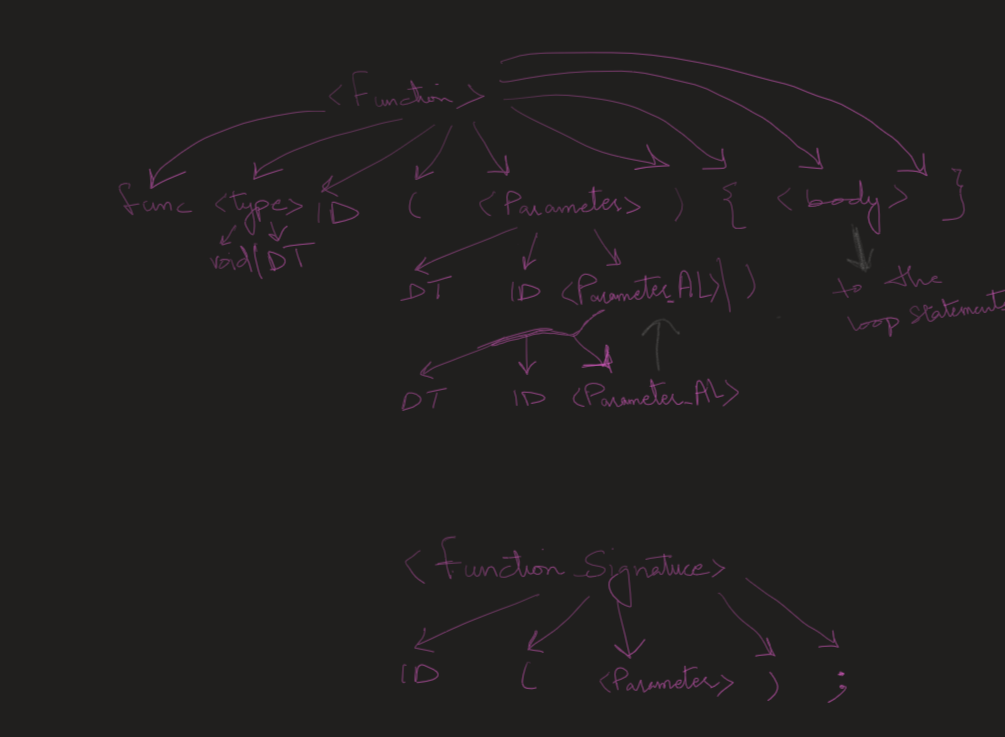
The figure above is a slightly simplified example of what the parser will do. The input to the parser is the sequence of tokens and the output is the parse tree. A parse tree is a more structured representation of the code than just a text string or a sequence of tokens. The process to create this tree will be discussed in the remainder of this post. I know trees can sometimes be scary, but we won't be building any complicated data structure for this. Rather, we will utilize the *call stack* of our parser to implicitly build the parse tree as we go.

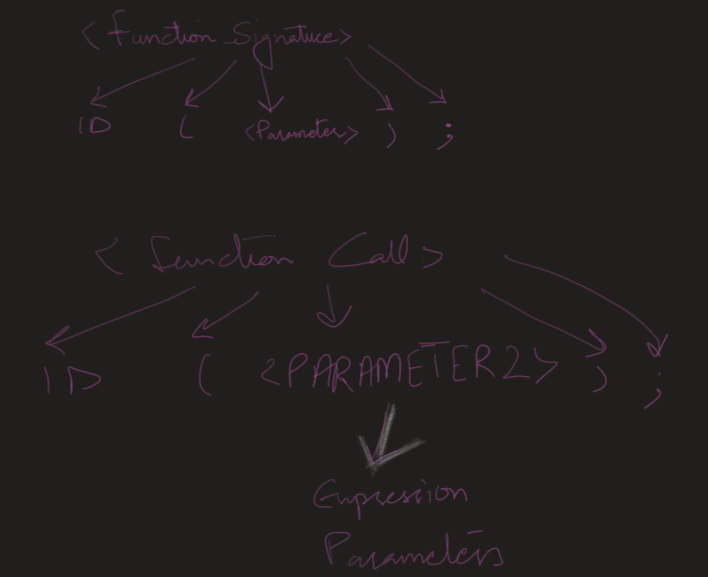
**Some Parser Tree:**





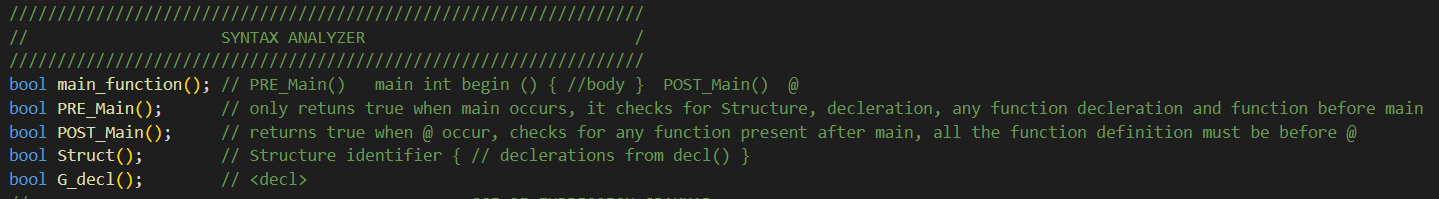




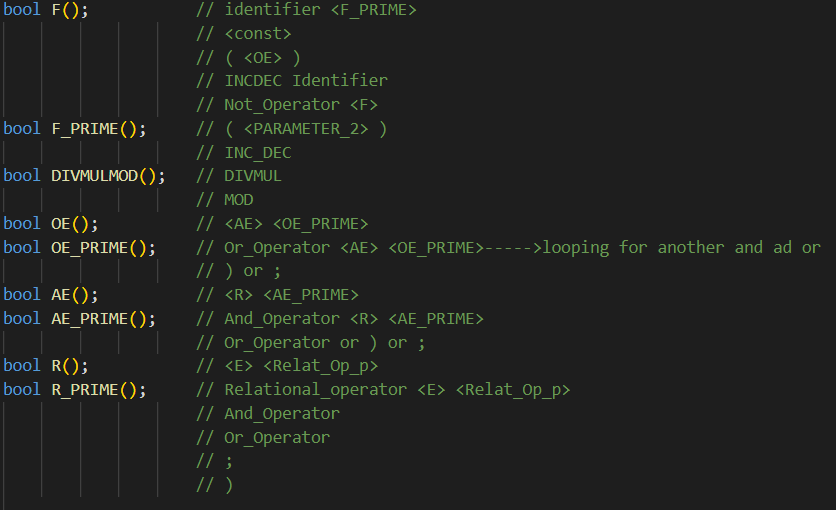
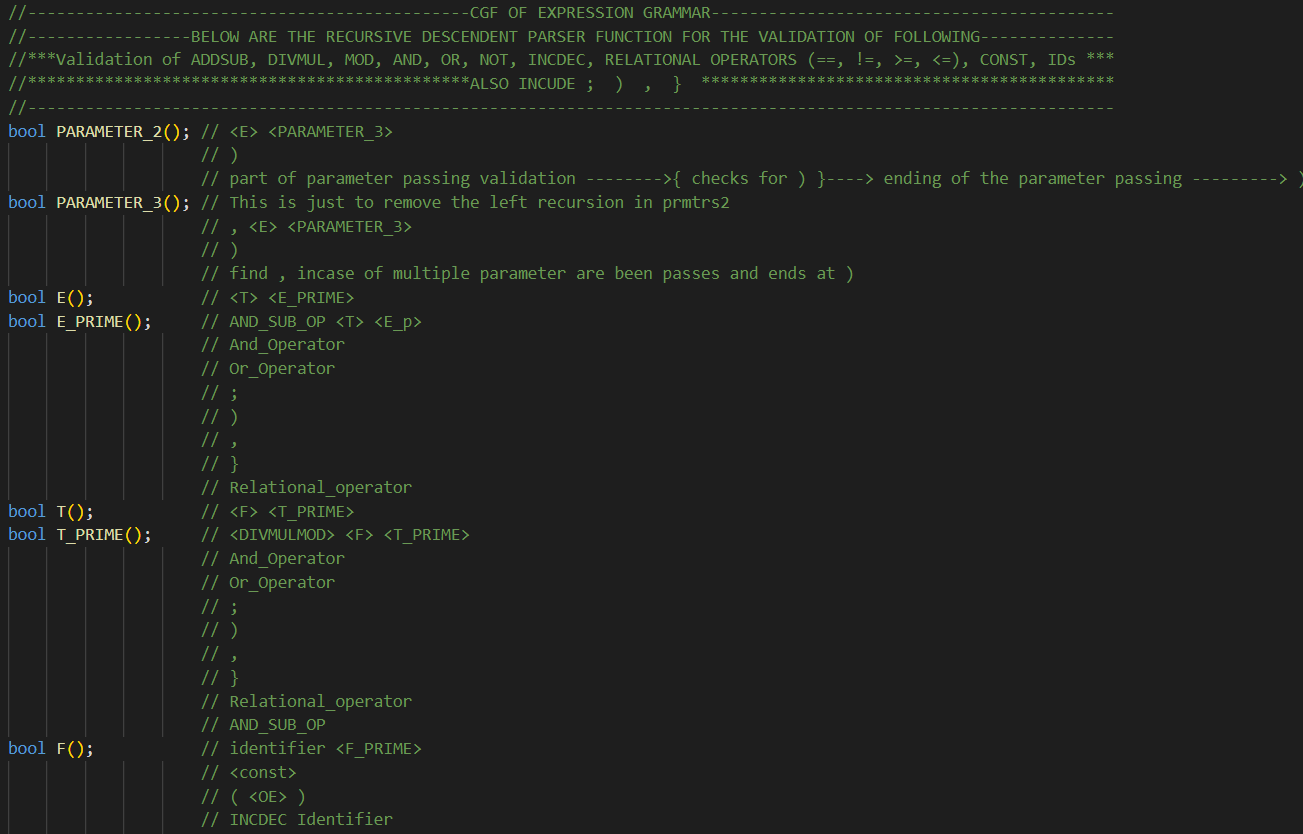


**Some Syntax analyzer Functions:**

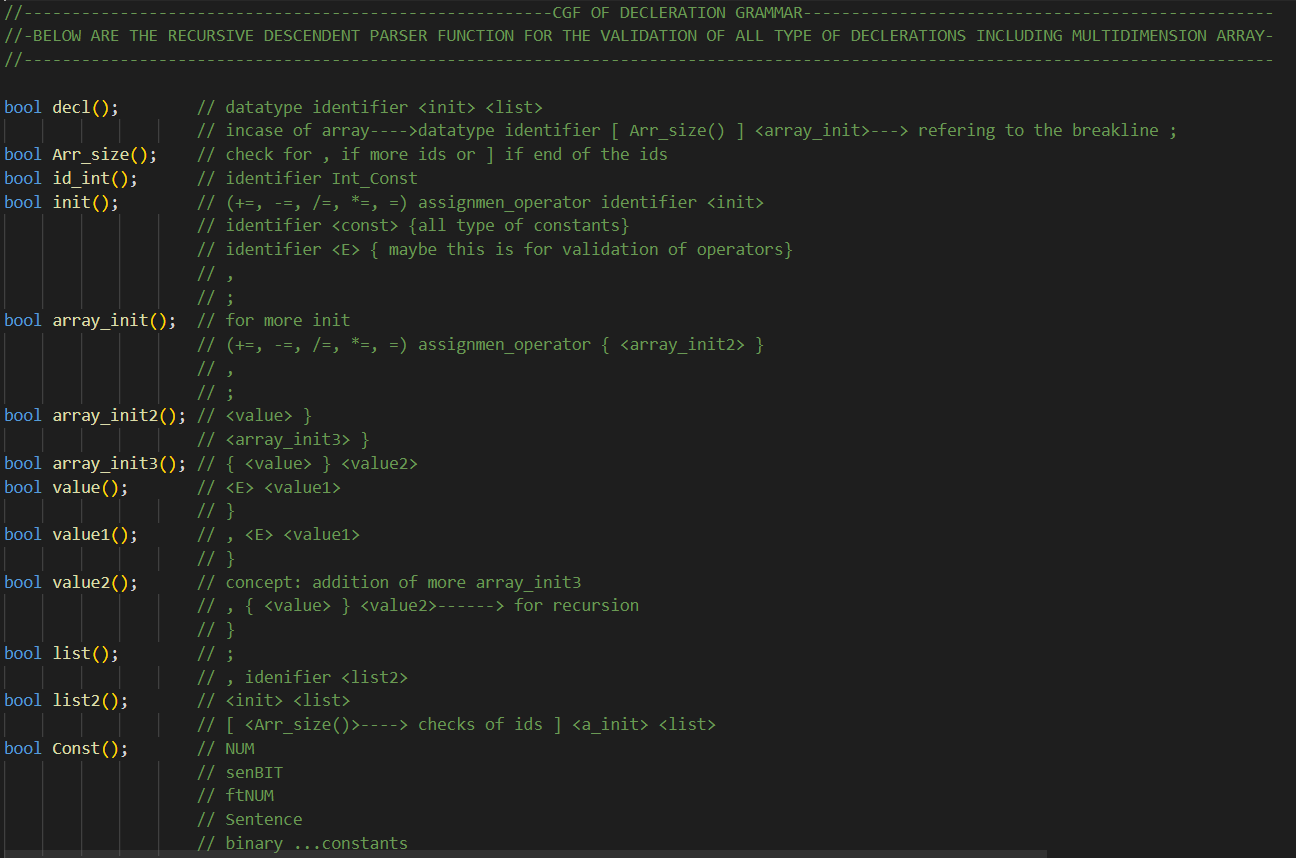
**Main functions is the main source of returning the valid response:**



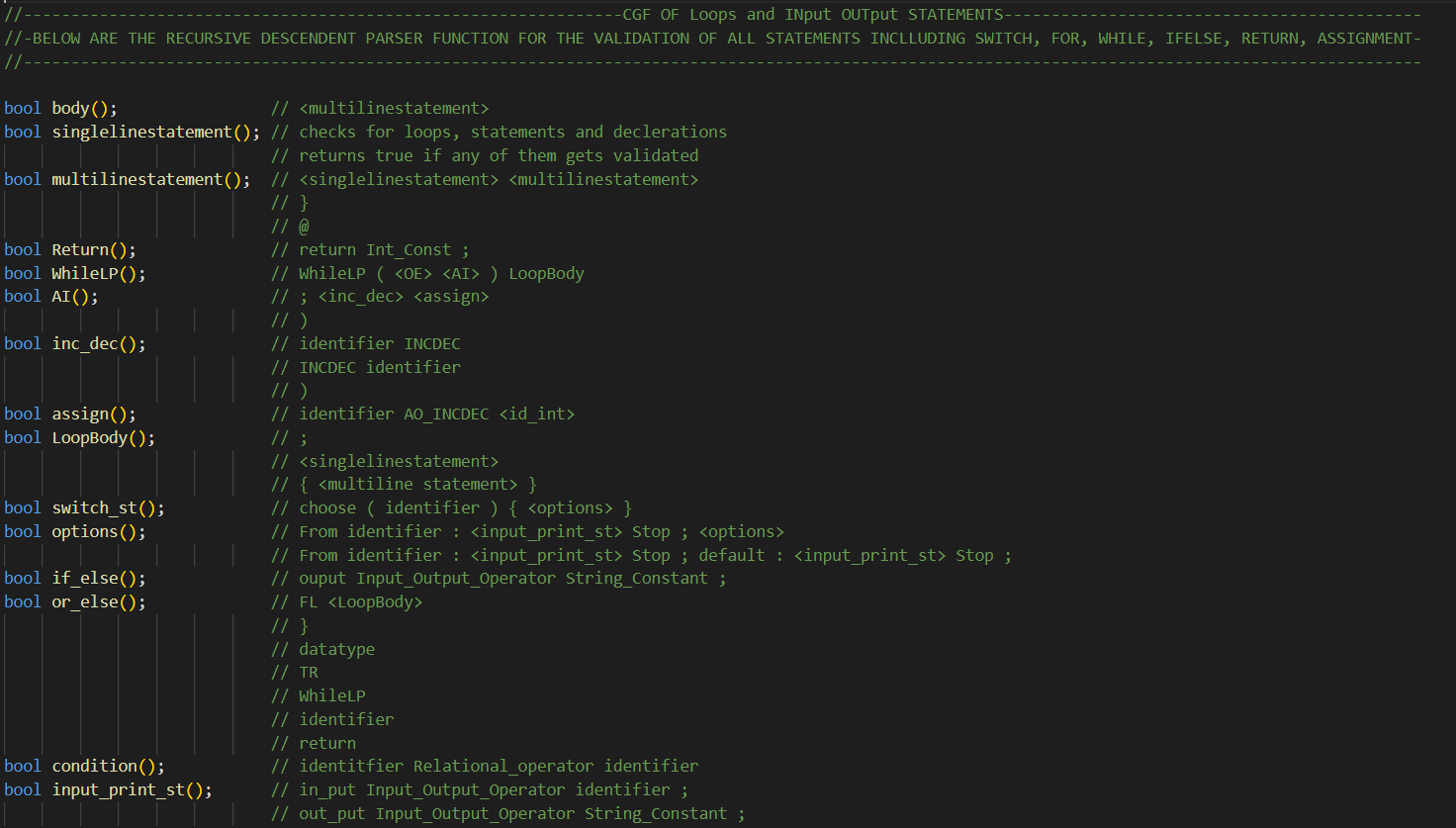
**Expression Grammar validation:**

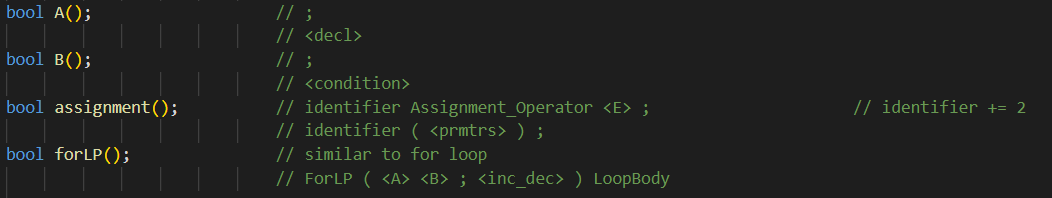


**Function for Declarative Statement Cfgs:**

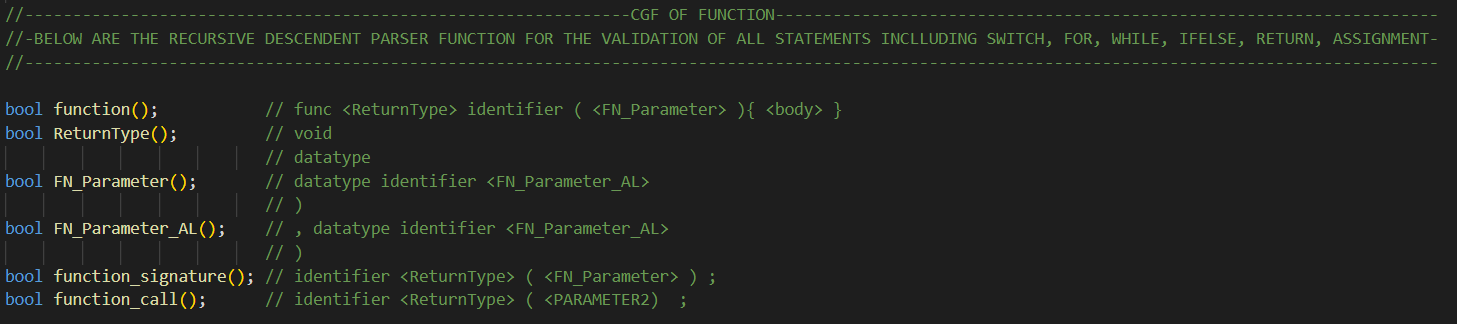


**Function for Loops and input output statement:**

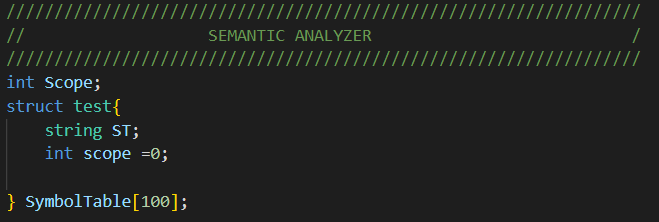


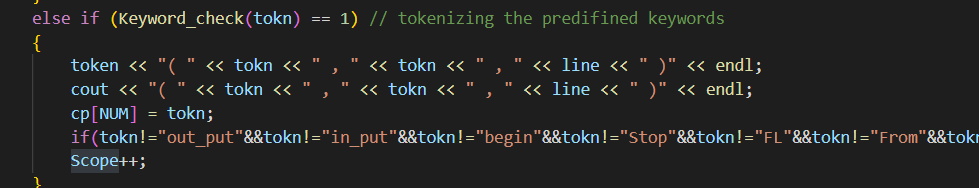


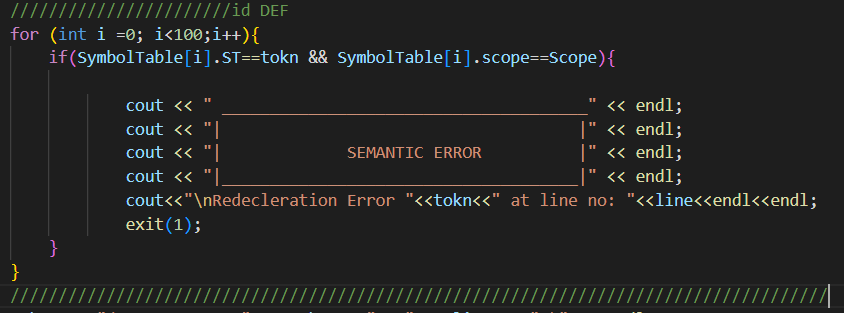
**Function for Function Deceleration CFG:**

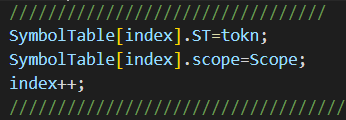


* Semantic Analyzer:









# Language Description

* **Data types**
* **ftNUM** (float)
* **NUM** (integer)
* **senBIT** (char)
* **Sentence** (string)
* **binary**  (bool)
* **Identifier**
* Starts with **~**
* **Comment**
* Single Comment (**->**)
* Multi line comment (**||< any comment >||**)
* **Loops**
* **for loop** = forLP (int a=1;a<=10;a++)

{ statements

};

* **while loop** = whileLP (true)

{ statements

};

* **Conditional Statement**
  + TR(true): 🡪if

statement

* + Fl:

statement 🡪else

* + choose (true) 🡪switch

{

from 1{…..};

from 2{…..} …

from n{…..}

}

* **Comments (Multi line + Single line):**

Multi line **|||…|||**

Single line **-->**

* **Function**

**func ~abc ()**

**{ statement }**

* **Structure**
* **Structure name { datatypes name}**
* **Array**
* **Datatype ~a [ ]={ }**
* **Input and output statement**
* Input <- identifier ;
* Output -> string\_constant;
* **Operators:**

|  |  |
| --- | --- |
| **Arithmetic Operator** | |
| + | addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulus |
| **Assignment Operator** | |
| = | Equal to |
| += | Add + assign |
| -= | Subtract + assign |
| \*= | Multiply + assign |
| /= | Divide + assign |
| %= | Modulus + assign |
| **Logical Operator** | |
| & | and |
| | | or |
| ! | not |
| **Comparison Operator** | |
| == | Is Equal |
| != | Is not equal |
| < | Is less than |
| > | Is greater than |
| <= | Is less than equal to |
| >= | Is greater than equal to |

* **Punctuators:**

|  |
| --- |
| ( ) |
| { } |
| [ ] |
| |||…||| or -->(for comment) |
| \newline (for line terminator) |
| # |
| “ “ (use in string) |
| ‘ ‘ (use in char) |
| . (use in float) |
| , (use in loops) |

* **Keywords:**

|  |  |  |
| --- | --- | --- |
| Keyword | Keyword | Keyword |
| forLP(for) | Tr(if) | Stop(stoping the loop) |
| Num(intigers) | Fl(else) | From(switch case options) |
| ftNUM(float) | Chose(switch case) | in\_put (for taking input) |
| Sentence(String) | whileLP(while) | Structure(for constructors creation) |
| senBIT(char) | Return (return func) | out\_put(for printing output) |

# Context Free Grammar

* **Main:**

---- PRE\_Main()   main int begin () { //body }  POST\_Main()  @---------

* **Deceleration:**

 Structure identifier { // declerations from decl() }

* **Return:**

--return Int\_Const--

* **DECLARATION:**

//<DECL>◊ID <INT> <LIST> DT!

//<INT>◊ID <INT> | = <const> | epsilon

//<LIST>◊;| , ID <INT> <LIST> | epsilon

//<const>◊ chr const | int const | flt const | str const

* **BODY:**

//<body>◊<mst>

//<mst>◊<sst><mst> |epsilon

//<sst>◊<DECL> | <if> | <while> |<dw>| <call\_func> | <for>

* **IF - ELSE:**

// <if> ◊ TR (<OE>) <l\_body> <elif>

// <elif> ◊ <elif> <body> | epsilon

* **WHILE:**

---------WhileLP( condition ){ body }-------------

//<AI><inc> |<Assign>|epsilon

//<ID\_int> ID | int const

//<inc> ID incdec | incdec ID

* **FOR LOOP:**

// --------ForLP ( A B ; INCDEC) { body }

* **Function**

//------------func Identifier ( parameter ) { body }----------------

* **CONDITIONAL STATEMENT:**

// <OE> ◊ <AND> <OE’>

// <OE’>◊|| <AND> <OE’> | epsilon

// <AND>◊ <RE> <AND’>

// <AND’>◊ &&  <RE> <AND’> | epsilon

// <RE>◊ <E> <RE’>

// <RE’>◊ Relop <E> <RE’> |epsilon

// <E> ◊ <T> <E’>

// <E’>◊ PM <T> <E’>| epsilon

// <T> ◊ <F> <T’>

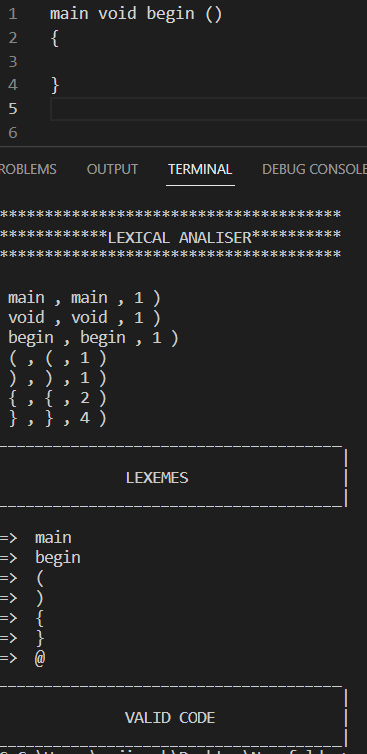
// <T’> ◊ divMul <F><T’> |epsilon

// <F> ◊ ID <F’> | <const> | <OE> | incdec ID | !<F>

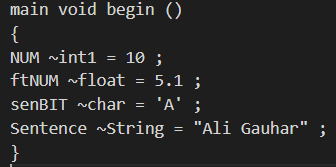
// <F’>◊ (<P>) | incdec ID | epsilon

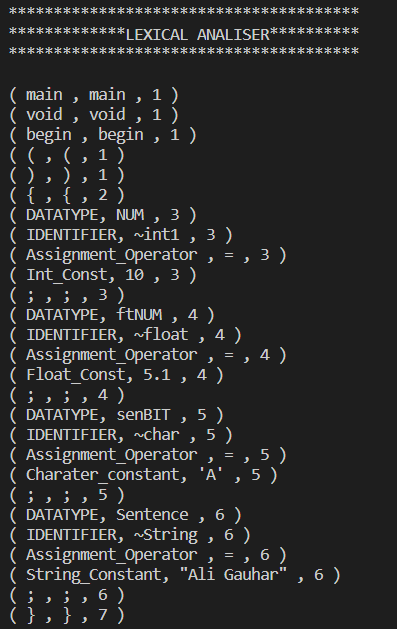
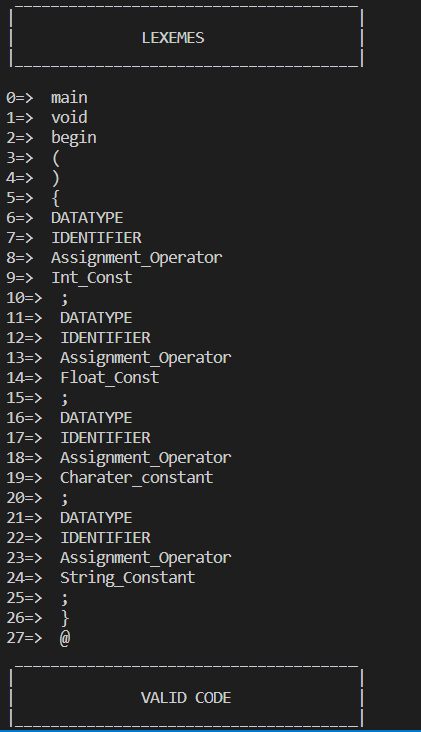
# Outputs:

**Main function:**

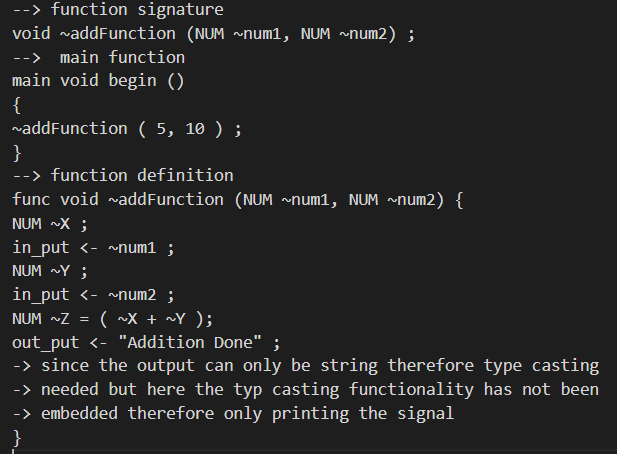


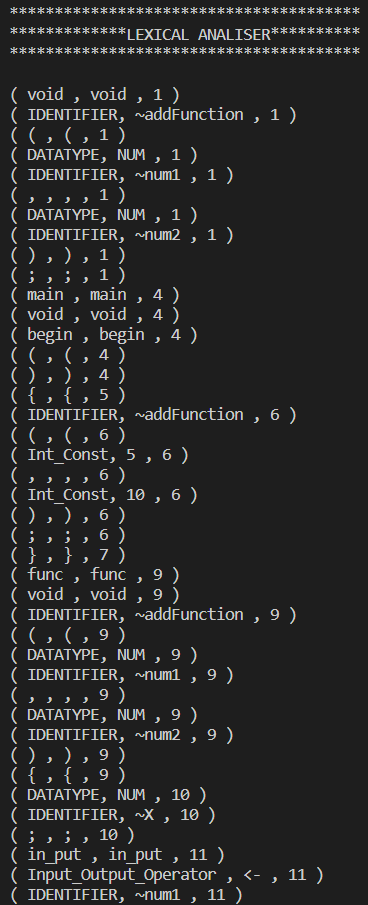
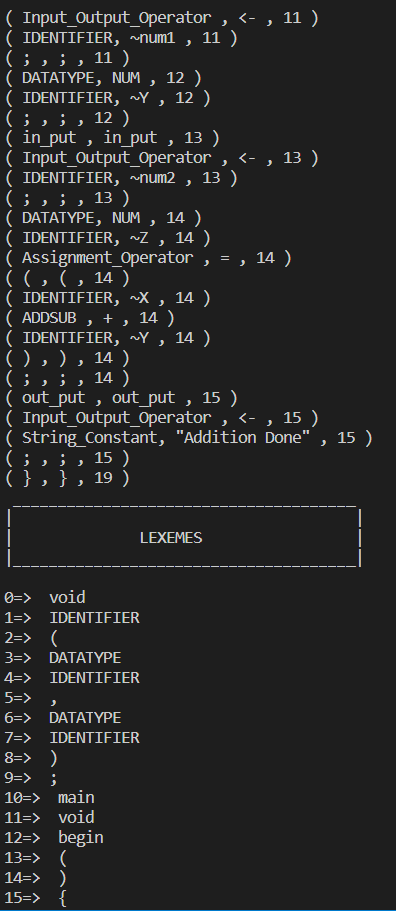
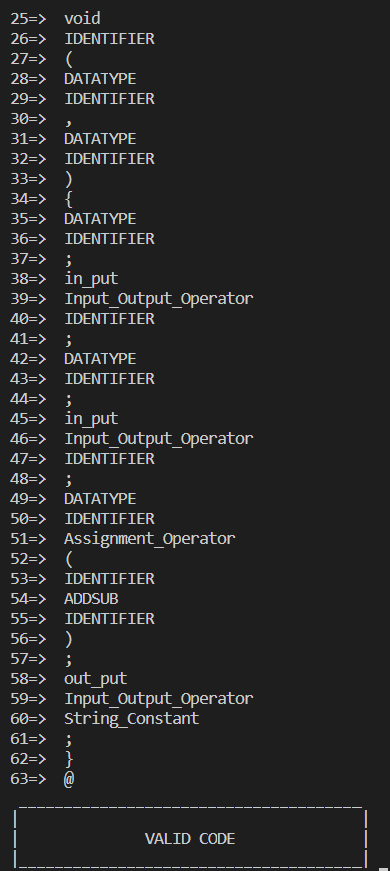
**Decelerations;**



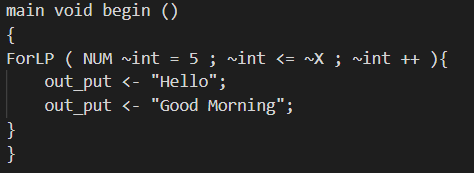
 

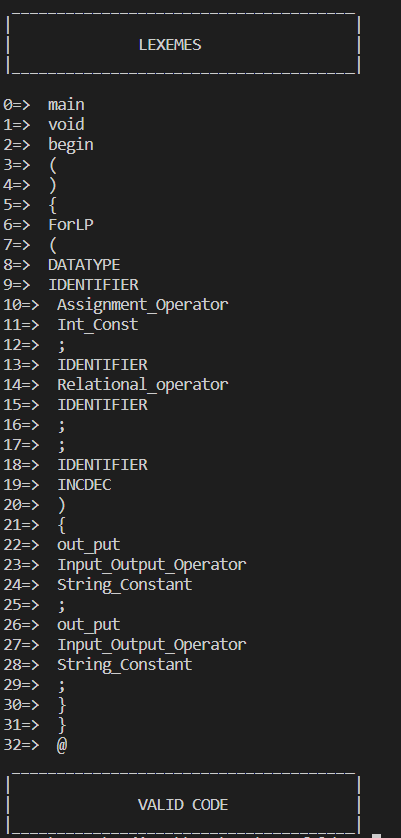
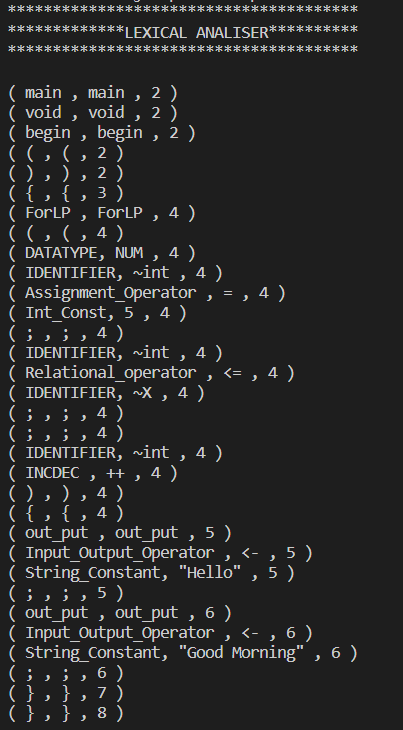
**Single line comment, input output statement and functions:**



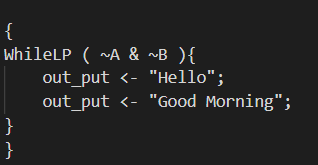
  

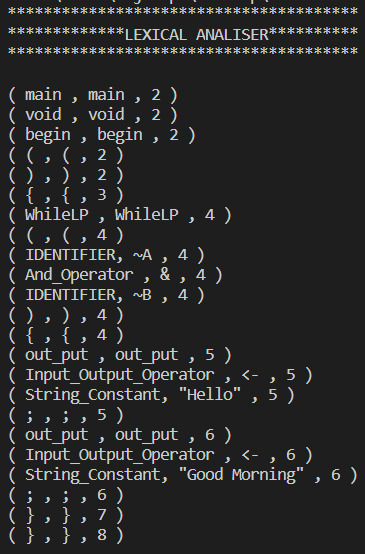
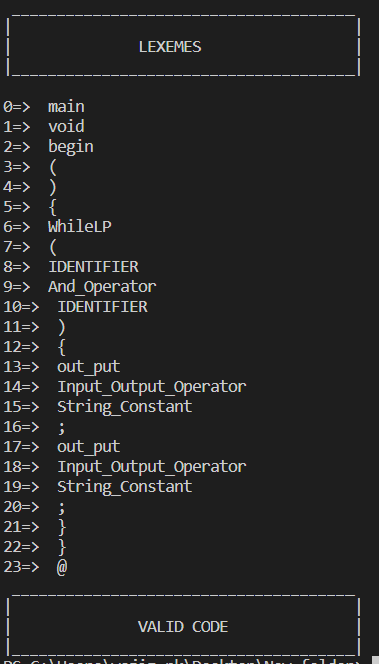
**For loop:**



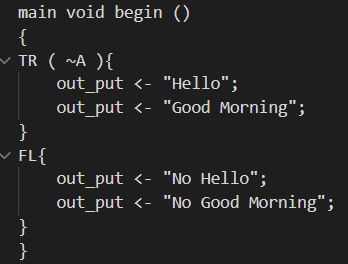


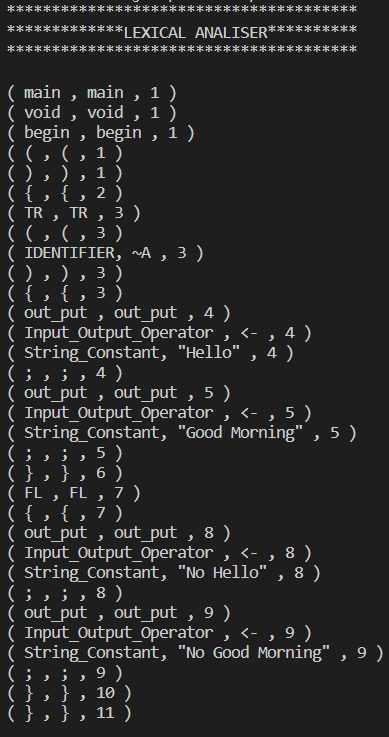
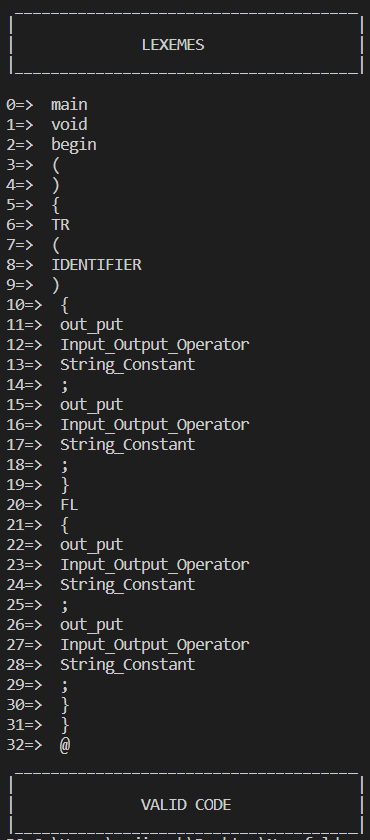
**While Loop:**



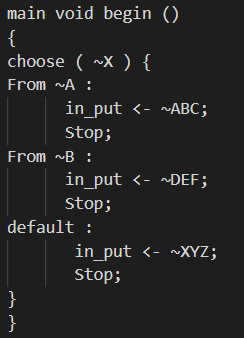
 

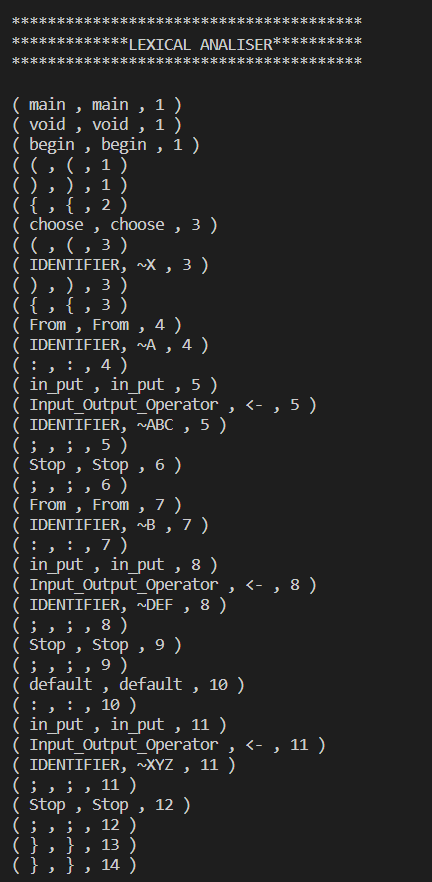
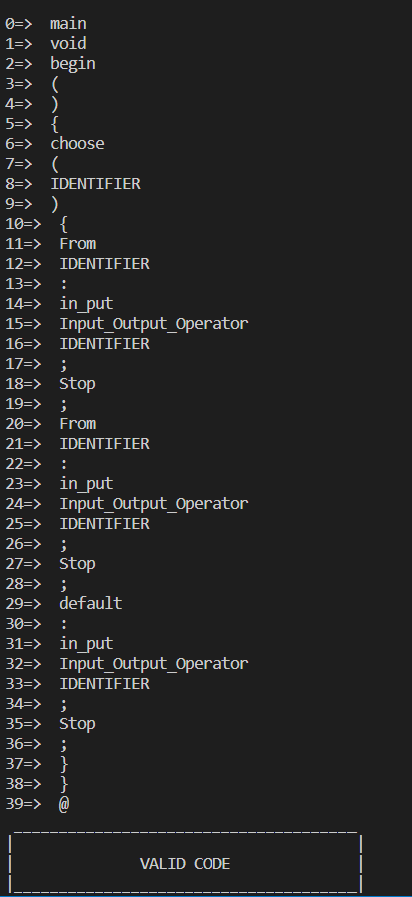
**IF else statement:**



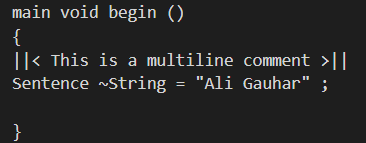
 

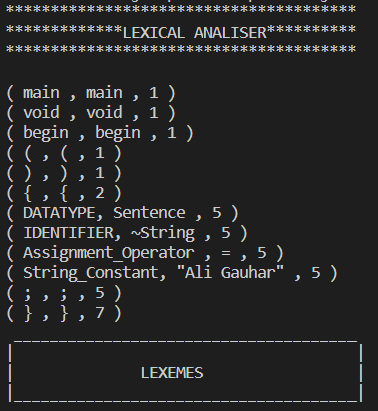
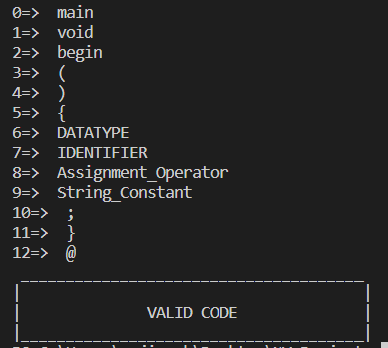
**Switch cases:**



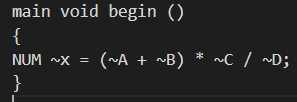
 

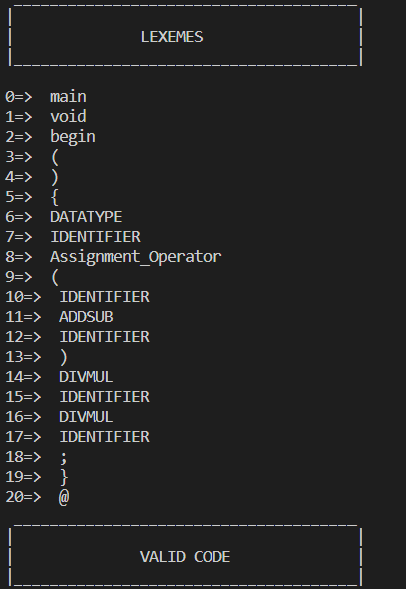
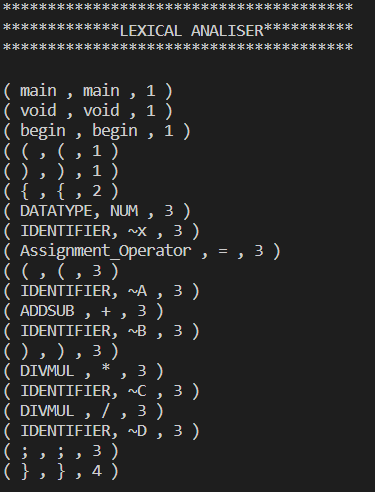
**Multiline comment;**



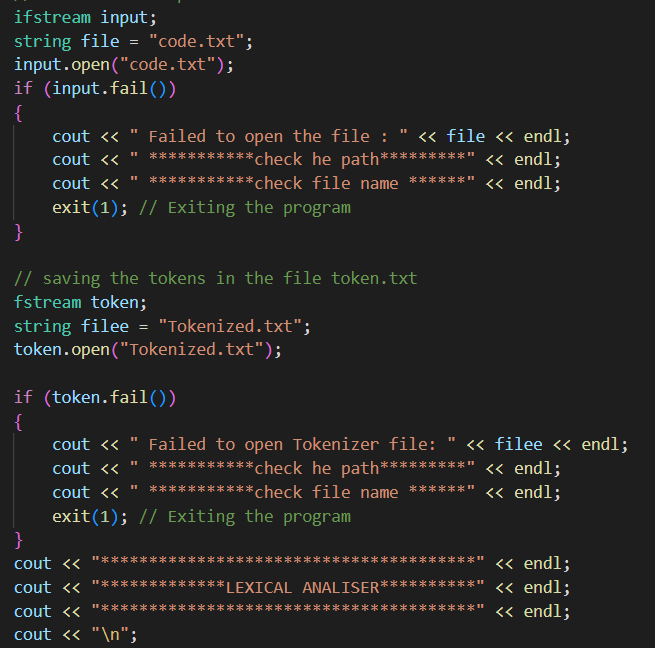
 

**Expression:**

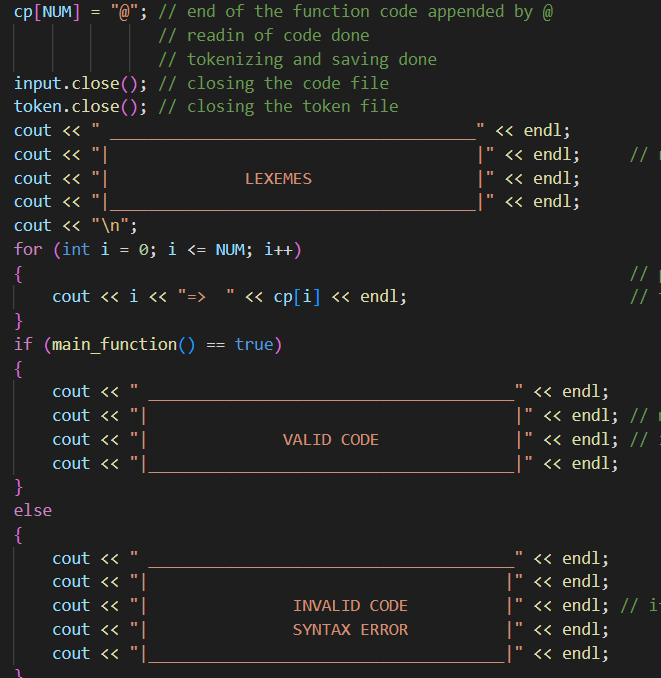




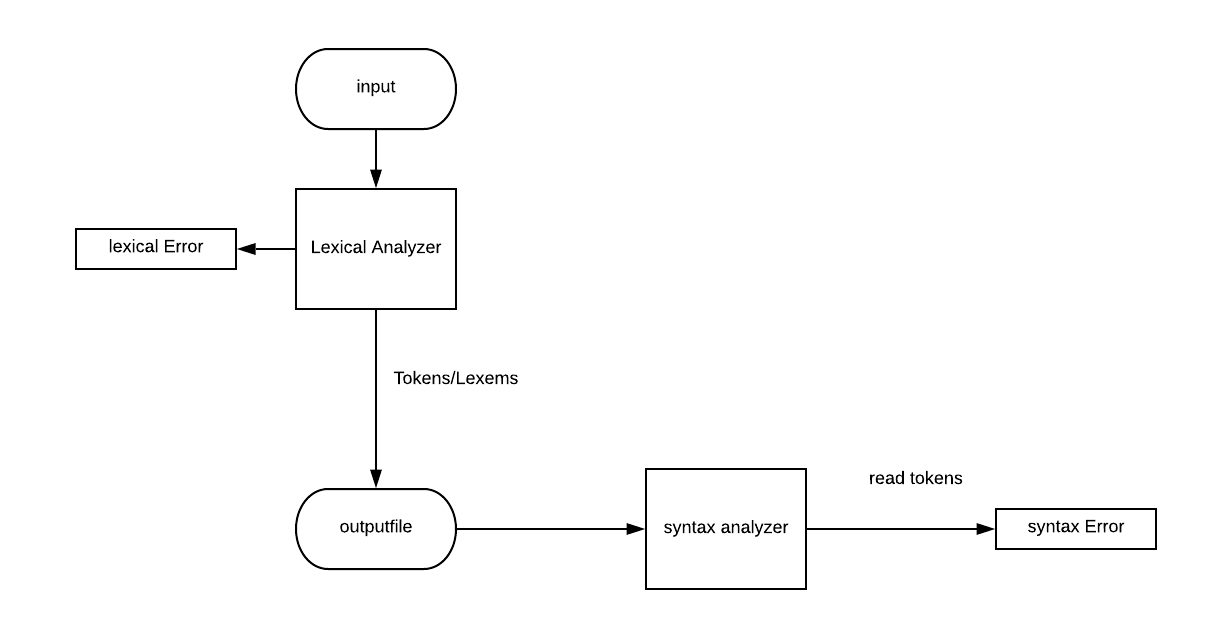
**Code reading and tokenization:**



**Validation Output:**



# Work Flow Diagram



# References

1. <https://www.thecrazyprogrammer.com/2017/02/lexical-analyzer-in-c.html>
2. <https://www.youtube.com/watch?v=edZfw9Yp7h4>
3. [**https://www.youtube.com/watch?v=Eythq9848Fg&list=PLZQftyCk7\_SdoVexSmwy\_tBgs7P0b97yD&ab\_channel=CodePulse**](https://www.youtube.com/watch?v=Eythq9848Fg&list=PLZQftyCk7_SdoVexSmwy_tBgs7P0b97yD&ab_channel=CodePulse)
4. [**https://www.youtube.com/watch?v=f1u3me4GYmw&ab\_channel=ProgrammingHero**](https://www.youtube.com/watch?v=f1u3me4GYmw&ab_channel=ProgrammingHero)