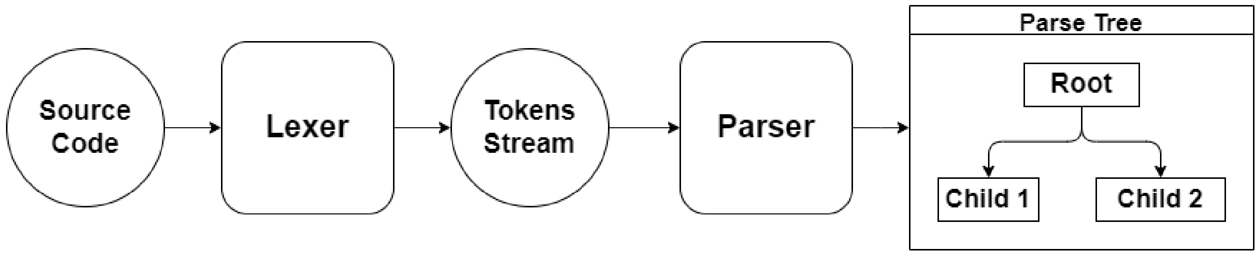
Project Architecture



1- Lexer (Scanner): Responsible for scanning the source code character by character and detecting a reserved patterns for every token type, then converting this stream of characters into stream of tokens

2- Parser: Responsible for converting the tokens into meaningful syntax nodes with children

Our Parser
This parser is implemented in typescript + node, it understands a custom syntax that will be described in the CFG later.
This parser can compile about 7 Statements:

1. If Else Statement
2. While Statement
3. Do While Statement
4. Switch Case Statement
5. Assignment Statement
6. Break Statement
7. Expression Statement

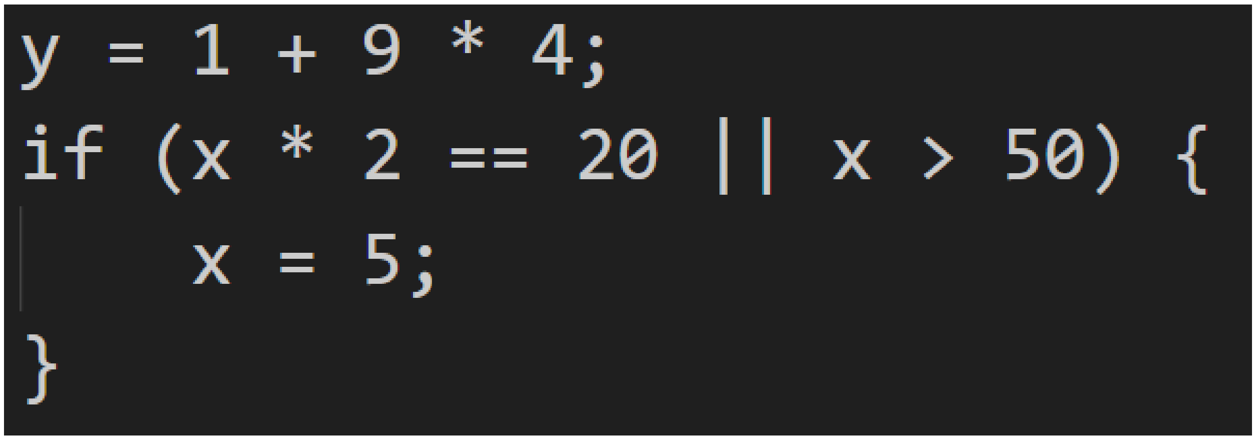
It has 22 type of nodes which is:

1. Program
2. Statements
3. Block Statement
4. If Statement
5. Else Statement
6. While Statement
7. Do_While Statement
8. Switch Statement
9. Cases
10. Case Clause
11. Default
12. Expression Statment
13. Expression
14. Assignement Statement
15. Logical Expression
16. Conditional Expression
17. Expr (Math Expression)
18. Term
19. Factor
20. Unary Expression
21. Break Statement
22. **Number**
23. **Identifier**
24. **Syntax Node**

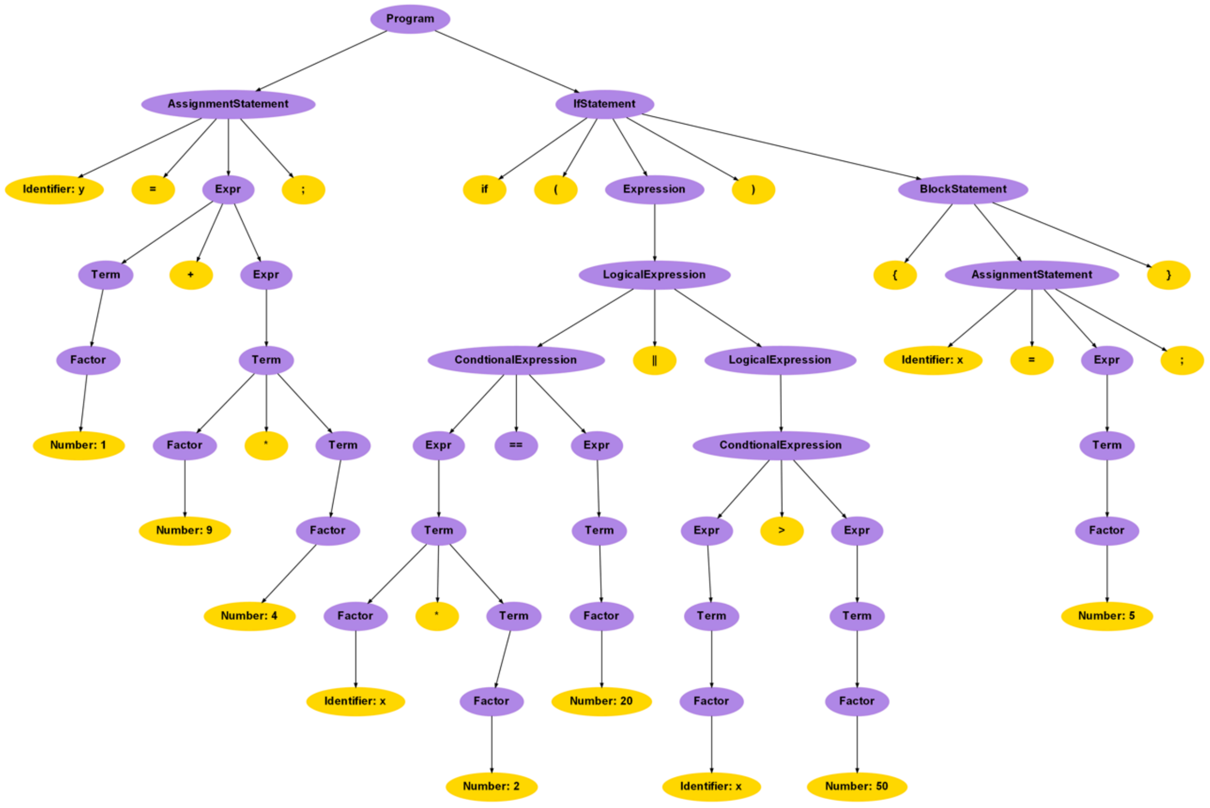
Context Free Grammer:

Program ⇒ Statments
Statments ⇒ Statment Statments | Empty
Statment ⇒ BlockStatment | IfStatment | WhileStatment | SwitchStatment | Do_WhileStatment | ExpressionStatment | BreakStatment | AssignmentStatment
IfStatement ⇒ **if (** Expression **)** Statement ElseStatement
ElseStatement ⇒ **else** Statement | Empty
BlockStatment ⇒ **{** Statements **}**
WhileStatement ⇒ **while (** Expression **)** Statement
DoWhileStatement ⇒ **do** Statement **while (** Expression **)**
SwitchStatement ⇒ **switch (** Identifier **) {** Cases Default **}**
Cases ⇒ CaseClause Cases | Empty
CaseClause ⇒ **case** factor **:** Statements
Default ⇒ **default :** Statements | Empty
BreakStatement ⇒ **break ;**
AssignementStatements ⇒ **Identifier =** Expression **;**
ExpressionStatement ⇒ Expression **;**
Expression ⇒ Expr | LogicalExpression | **Identifier**
LogicalExpression ⇒ LogicalExpression LogOp Conditional | Conditional
Conditional ⇒ Expr ConOp Expr | Expr
logOp ⇒ **&&** | **||**
ConOp ⇒ **>** | **<** | **==** | **!=** | **===** | **<=** | **>=**
Expr ⇒ Term **+** Expr | Term **-** Expr | Term
Term ⇒ Factor ***** Term | Factor **/** Term | Factor
Factor ⇒ **Identifier** | **Number** | **(** Expr **)** | UnaryExpression
UnaryExpression ⇒ **+** Factor | **-** Factor

Example:



```
y = 1 + 9 * 4;
if (x * 2 == 20 || x > 50) {
    x = 5;
}
```



Program
AssignmentStatement
IfStatement
Identifier: y
=
Expr
;
if
(
Expression
)
BlockStatement
Term
+
Expr
LogicalExpression
{
AssignmentStatement
}
Factor
Term
CondtionalExpression
||
LogicalExpression
Identifier: x
=
Expr
;
Number: 1
Factor
*
Term
Expr
==
Expr
CondtionalExpression
Term
Number: 9
Factor
Term
Term
Expr
>
Expr
Factor
Number: 4
Factor
*
Term
Factor
Term
Term
Number: 5
Identifier: x
Factor
Number: 20
Factor
Factor
Number: 2
Identifier: x
Number: 50