Table of Contents

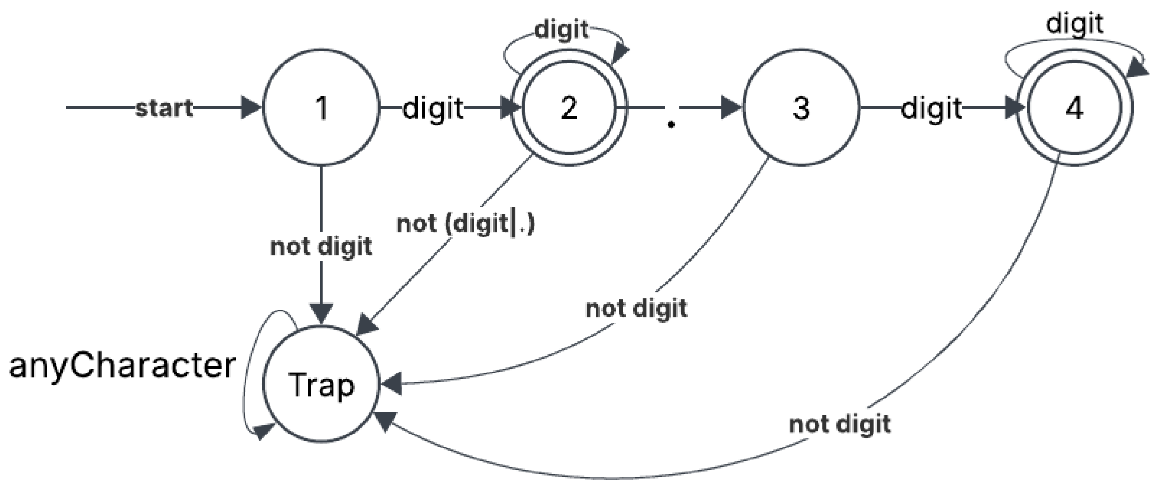
1.0 Regular Expressions and their DFA’s

1.1 Number

digit =0|1|2|3|4......|9

Number = digit+(\.digit+)?

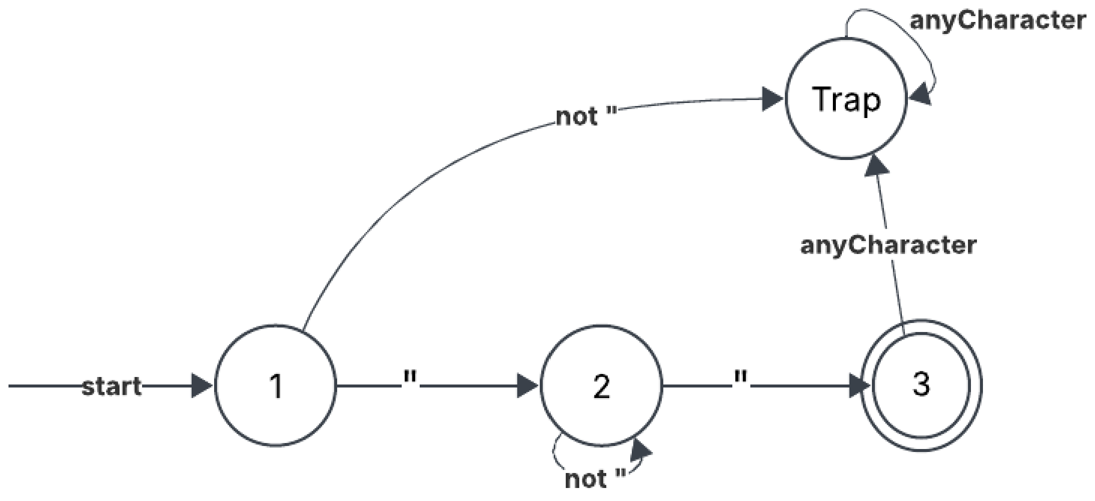
anyCharacter = .



1.2 String

String = \"[^"]*\"

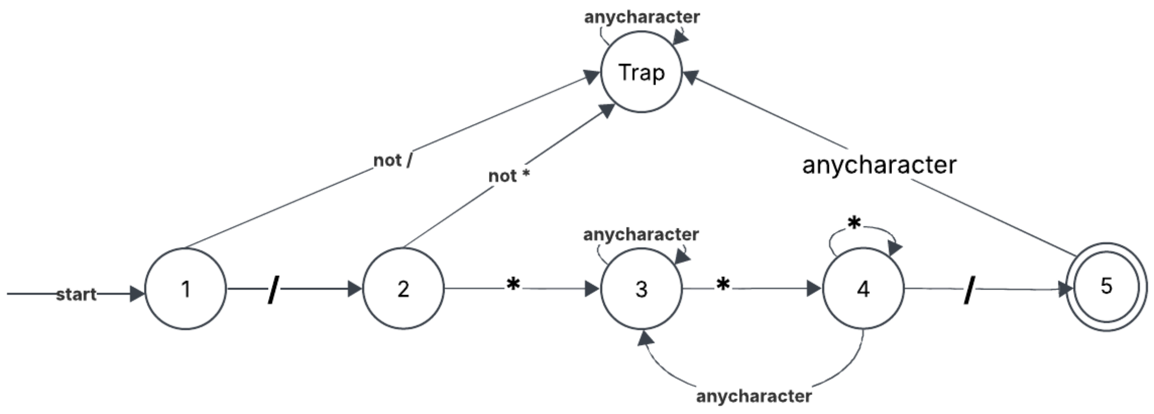
anyCharacter = .



1.3 Comment

Comment=/*(anycharacter)**/

anycharacter = /s/S



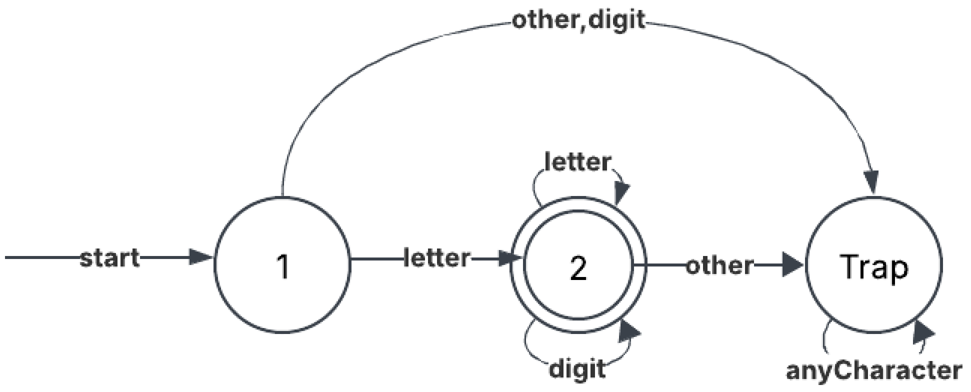
1.4 Identifier

letter = [a-zA-Z]

Identifier = letter (letter | digit)*

other = not (letter | digit)

anyCharacter = .

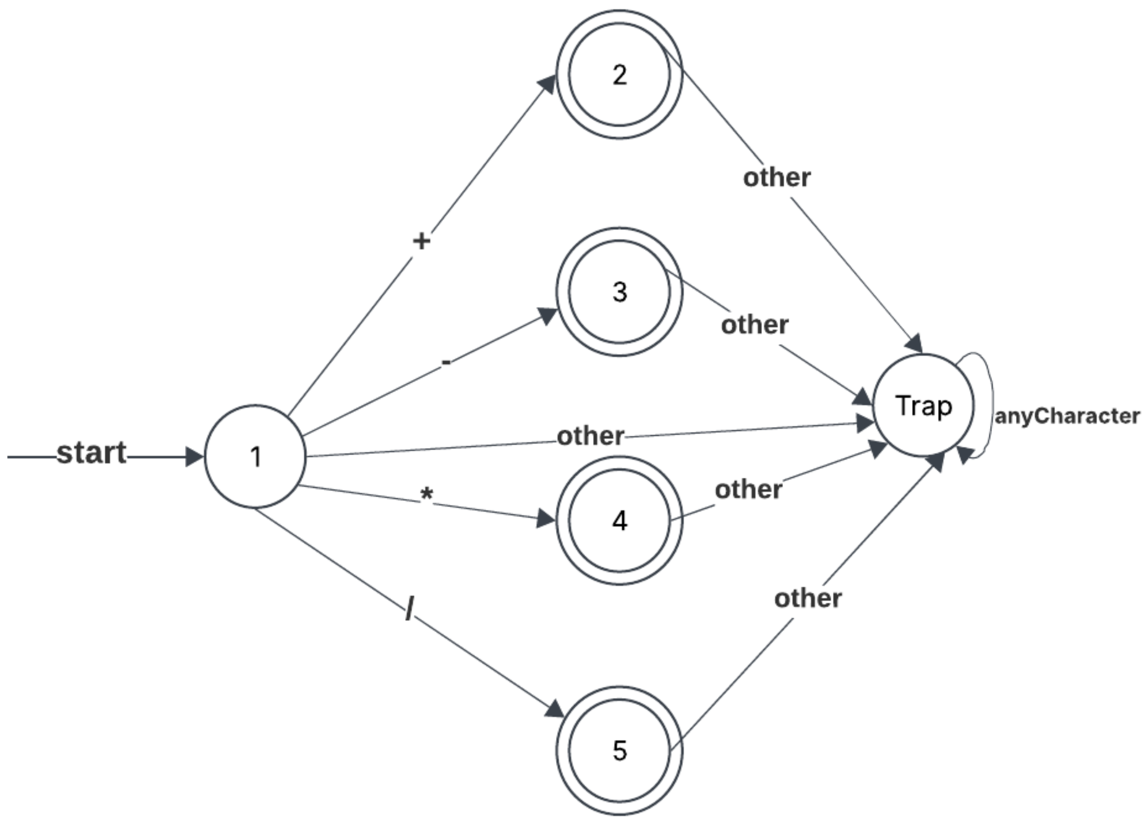


1.5 Arithmetic_Operator

Arithmetic_Operator = + | **-** | * | /

other = not Arithmetic_Operator

anyCharacter = .

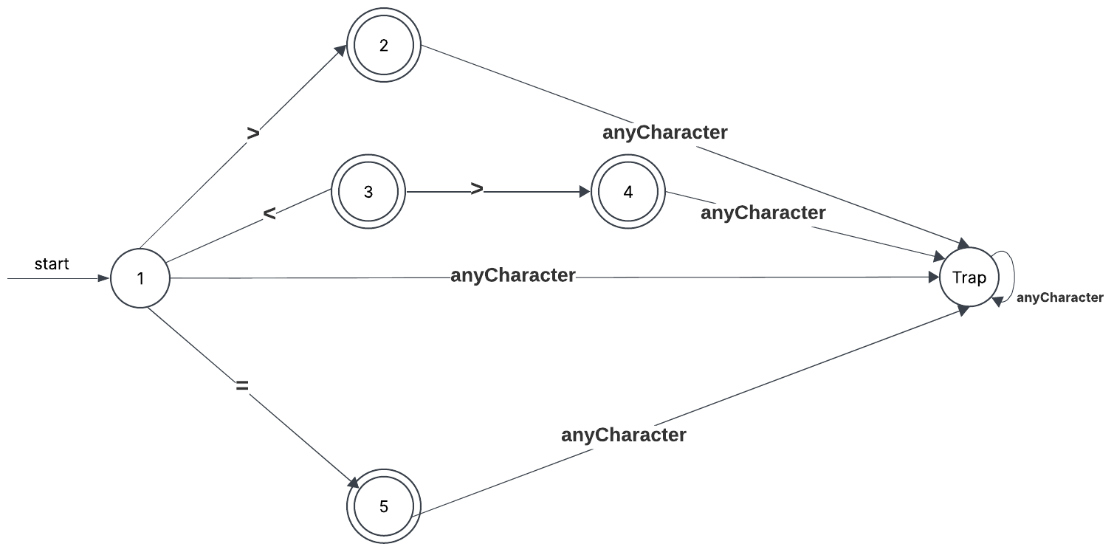


1.6 Condition_Operator

Condition_Operator= > | < | <> | =

other = not Condition_Operator

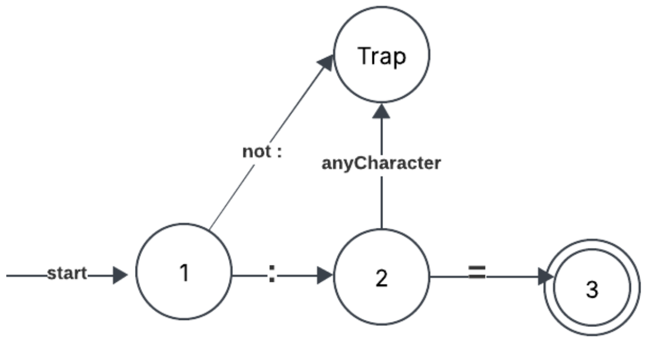
anyCharacter = .



1.7 Assignment_Operator

Assignment_Operator = :=

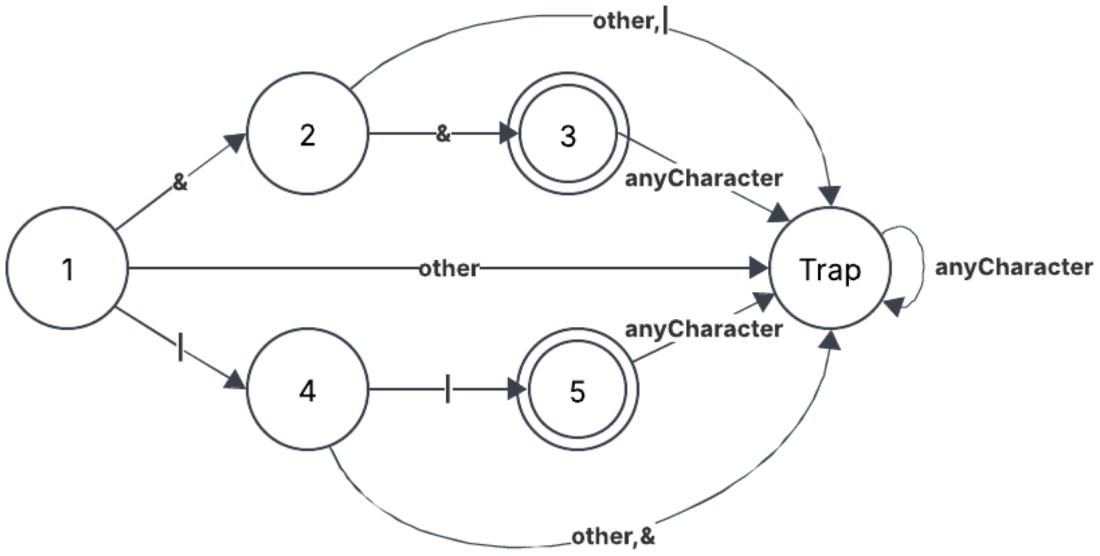
anyCharacter = .



1.8 Boolean_Operator

Boolean_Operator= && | ||
other = not Boolean_Operator

anyCharacter = .

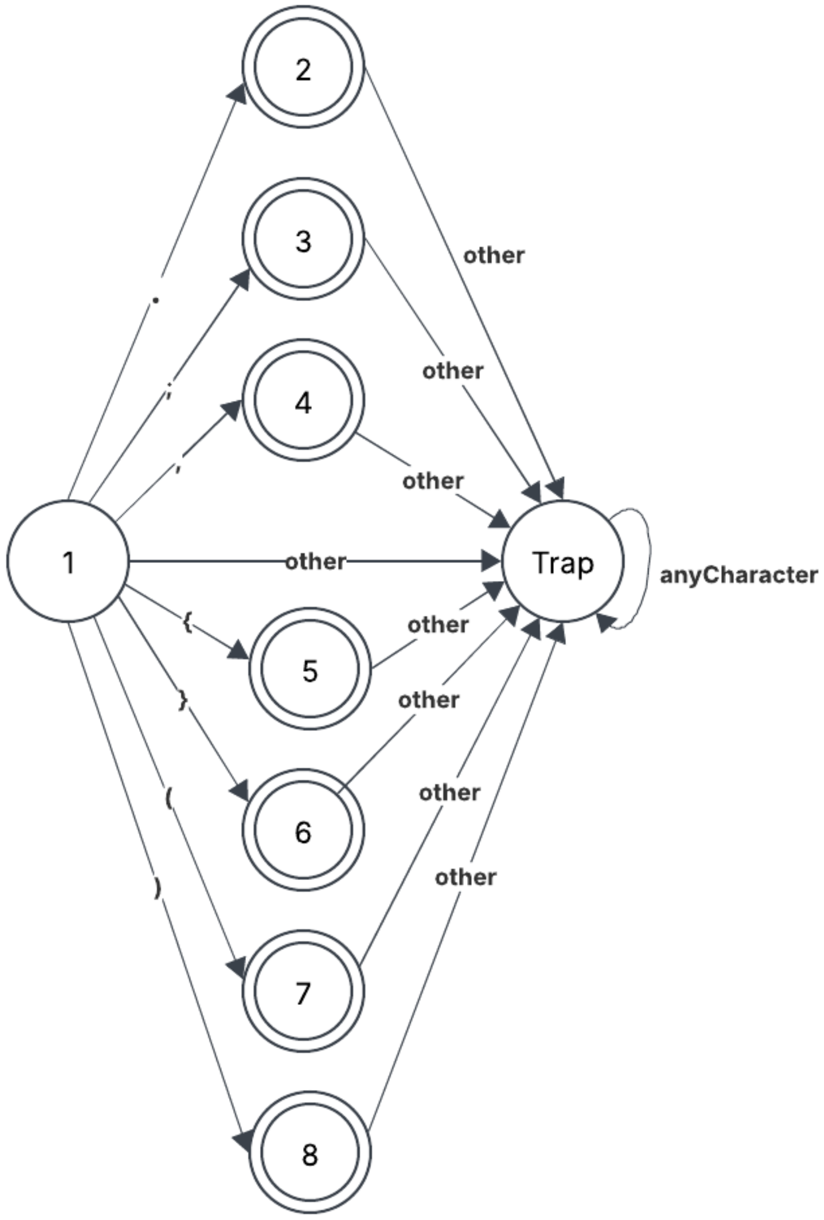


1.8 Symbols

Symbols= .|;|,|(|)|{|}

other = not Symbols

anyCharacter = .

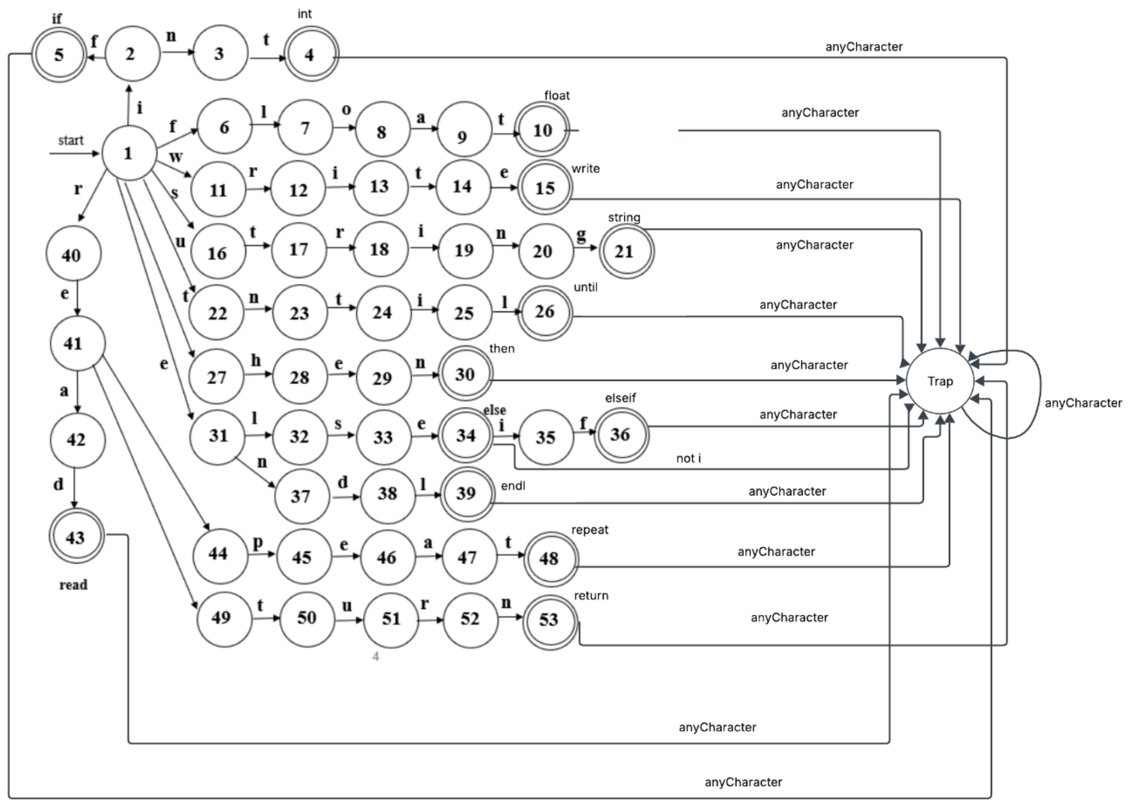


1.10 Reserved Keywords

Reserved_Keywords = int | float | string | read | write | repeat | until | if | elseif | else | then | return | endl

other= not Reserved_Keywords

anyCharacter = .

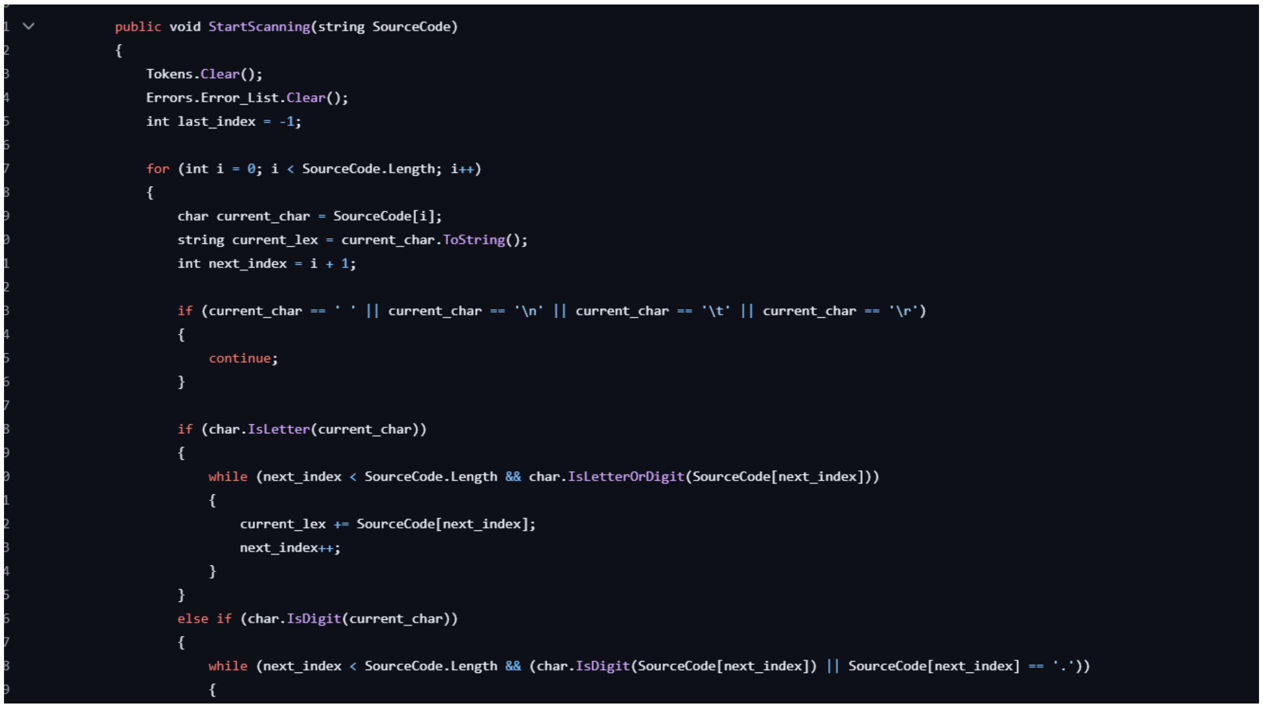


2.0 Scanner

2.1 GitHub Link

https://github.com/mo7amedmengasu/Tiny_Comp_phase1

2.2 Snippets from code

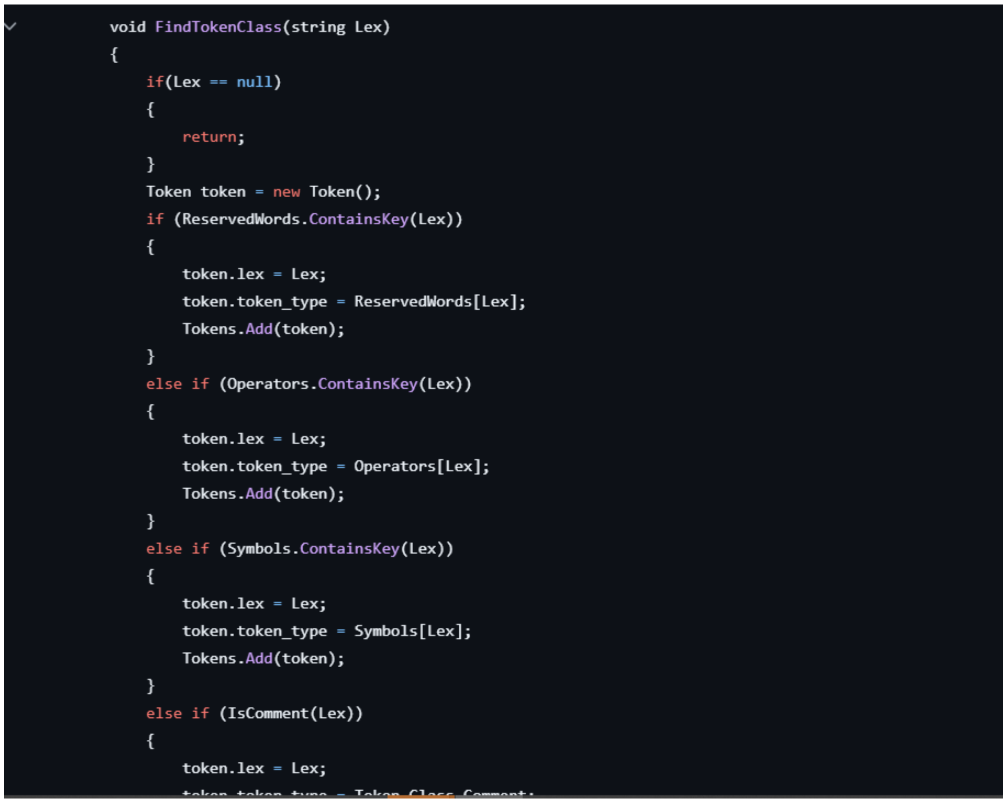


```
public void StartScanning(string SourceCode)
{
    Tokens.Clear();
    Errors.Error_List.Clear();
    int last_index = -1;

    for (int i = 0; i < SourceCode.Length; i++)
    {
        char current_char = SourceCode[i];
        string current_lex = current_char.ToString();
        int next_index = i + 1;

        if (current_char == ' ' || current_char == '\n' || current_char == '\t' || current_char == '\r')
        {
            continue;
        }

        if (char.IsLetter(current_char))
        {
            while (next_index < SourceCode.Length && char.IsLetterOrDigit(SourceCode[next_index]))
            {
                current_lex += SourceCode[next_index];
                next_index++;
            }
        }
        else if (char.IsDigit(current_char))
        {
            while (next_index < SourceCode.Length && (char.IsDigit(SourceCode[next_index]) || SourceCode[next_index] == '.'))
            {
```



```
        void FindTokenClass(string Lex)
        {
            if(Lex == null)
            {
                return;
            }
            Token token = new Token();
            if (ReservedWords.ContainsKey(Lex))
            {
                token.lex = Lex;
                token.token_type = ReservedWords[Lex];
                Tokens.Add(token);
            }
            else if (Operators.ContainsKey(Lex))
            {
                token.lex = Lex;
                token.token_type = Operators[Lex];
                Tokens.Add(token);
            }
            else if (Symbols.ContainsKey(Lex))
            {
                token.lex = Lex;
                token.token_type = Symbols[Lex];
                Tokens.Add(token);
            }
            else if (IsComment(Lex))
            {
                token.lex = Lex;
                token.token_type = Token_Class.Comment;
```