Lexer

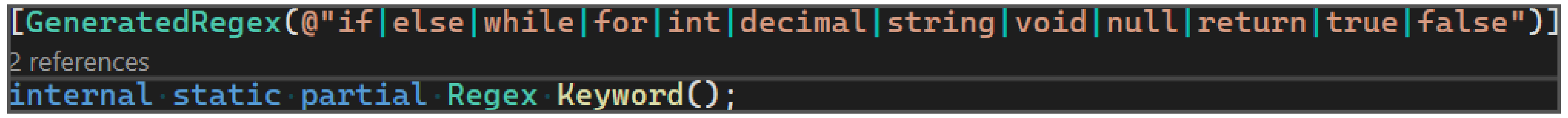
2025/2 - Compiladores

Matheus N. Toso

Tokens e Padrões

Palavras-chave

Padrão:

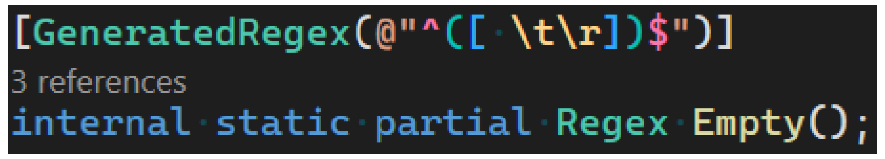


```
[GeneratedRegex(@"if|else|while|for|int|decimal|string|void|null|return|true|false")]
2 references
internal static partial Regex Keyword();
```

Nome: KEYWORD

Vazio

Padrão:

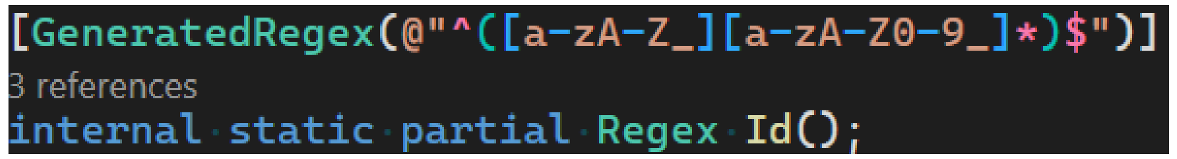


```
[GeneratedRegex(@"^([ \t\r])$")]
3 references
internal static partial Regex Empty();
```

Nome: EMPTY

Identificador

Padrão:

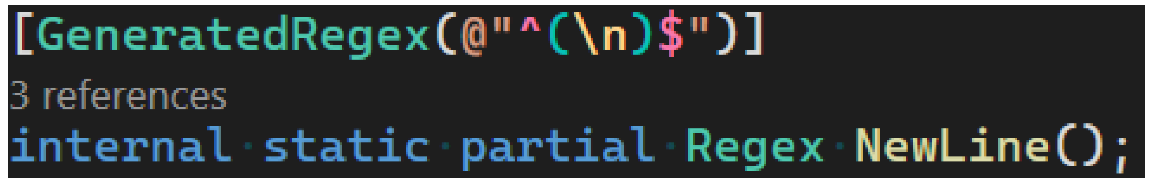


```
[GeneratedRegex(@"^([a-zA-Z_][a-zA-Z0-9_]*)$")]
3 references
internal static partial Regex Id();
```

Nome: ID

Nova Linha

Padrão:



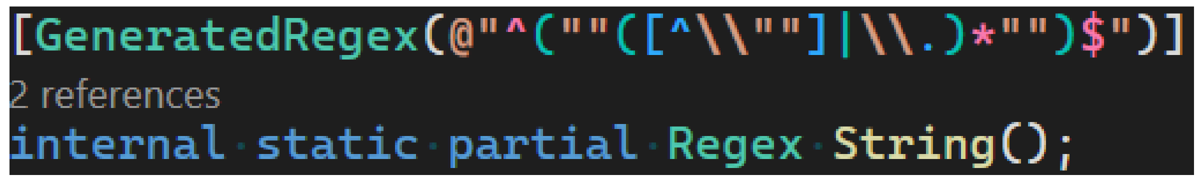
```
[GeneratedRegex(@"^(\n)$")]
3 references
internal static partial Regex NewLine();
```

Nome: NEW_LINE

CONSTANTES

String

Padrão:

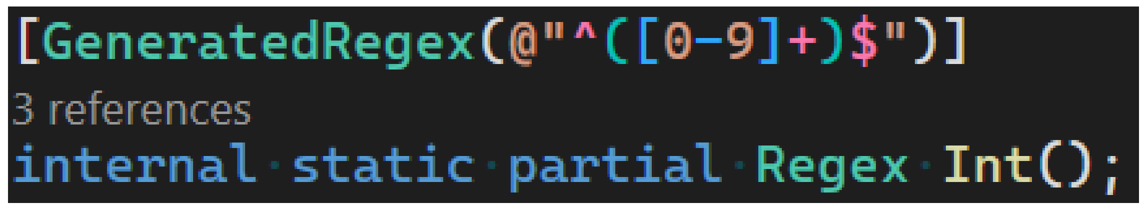


```
[GeneratedRegex(@"^(""([^\\""]|\\.)*"")$")]
2 references
internal static partial Regex String();
```

Nome: STRING

Inteiro

Padrão:

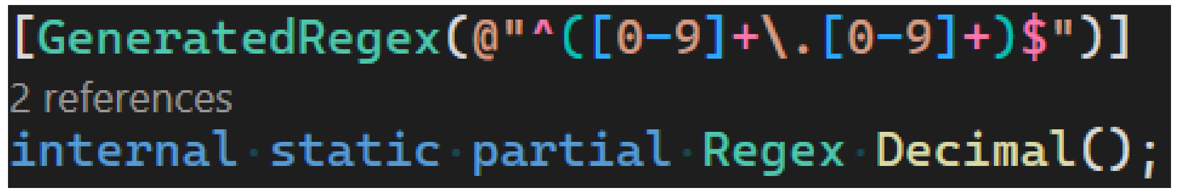


```
[GeneratedRegex(@"^([0-9]+)$")]
3 references
internal static partial Regex Int();
```

Nome: INT

Decimal

Padrão:



```
[GeneratedRegex(@"^([0-9]+\.[0-9]+)$")]
2 references
internal static partial Regex Decimal();
```

Nome: DECIMAL

OPERADORES

Atribuição

Padrão: =

Nome: ASSIGN

Igual

Padrão: ==

Nome: EQUAL

Diferente

Padrão: !=

Nome: NOT_EQUAL

Maior

Padrão: >

Nome: LARGER

Menor

Padrão: <

Nome: SMALLER

Maior ou Igual

Padrão: >=

Nome: LARGER_EQUAL

Menor ou Igual

Padrão: <=

Nome: SMALLER_EQUAL

OPERADORES

E

Padrão: &

Nome: AND

Ou

Padrão: |

Nome: OR

Negação

Padrão: !

Nome: NOT

Adição

Padrão: +

Nome: ADD

Subtração

Padrão: -

Nome: SUBTRACT

Multiplicação

Padrão: *

Nome: MULTIPLY

Divisão

Padrão: /

Nome: DIVIDE

DELIMITADORES

Parêntese Esquerdo

Padrão: (

Nome: LEFT_PARENTHESIS

Parêntese Direito

Padrão:)

Nome: RIGHT_PARENTHESIS

Chave Esquerda

Padrão: {

Nome: LEFT_BRACE

Chave Direita

Padrão: }

Nome: RIGHT_BRACE

Ponto e Vírgula

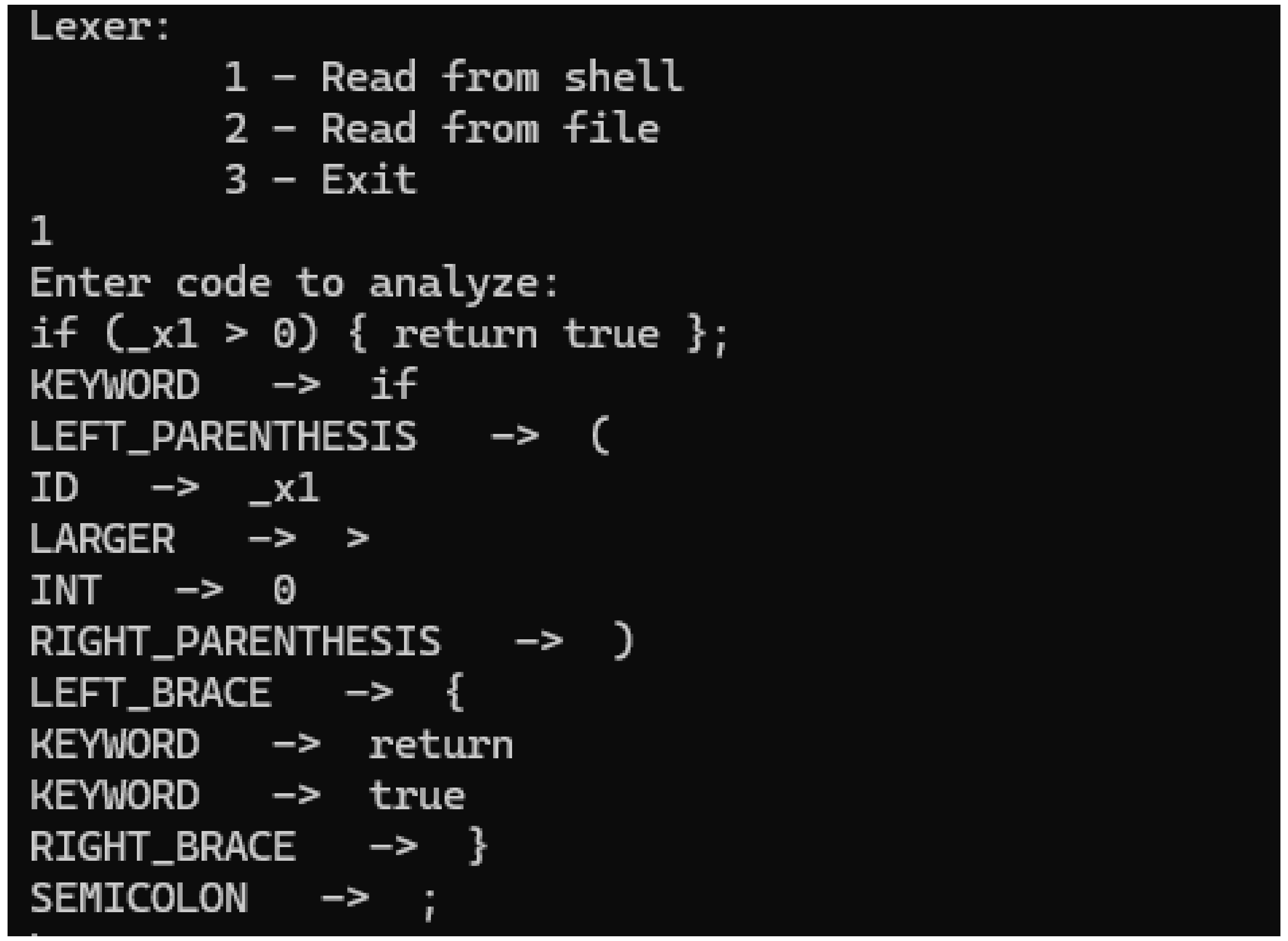
Padrão: ;

Nome: SEMICOLON

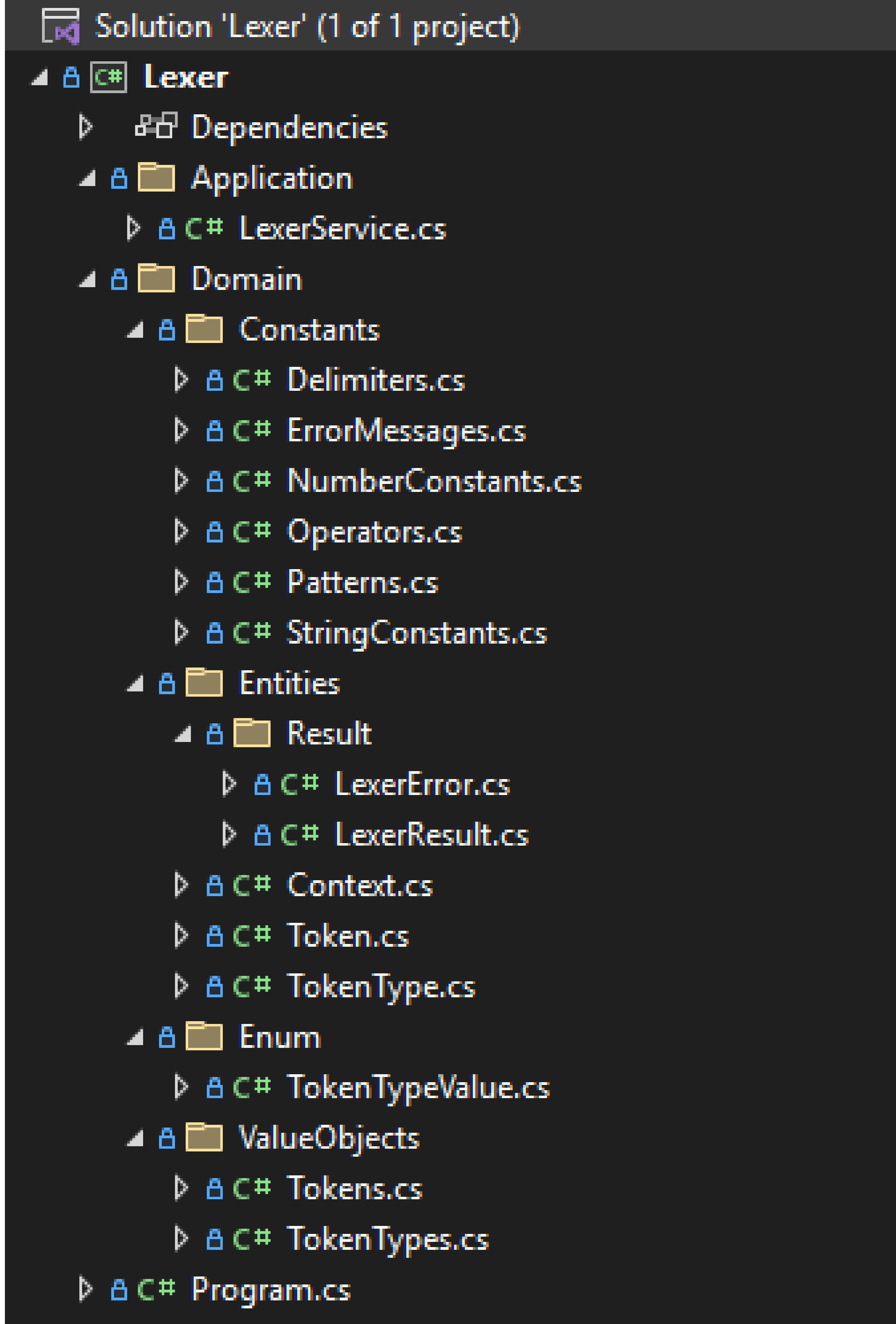
Código

Código

- .NET 7.0
- Aplicação de console
- Separada em Aplicação e Domínio
- Input por console ou arquivo
- Disponível em: https://github.com/matheustoso/COMPILERS.2025.2



```
Lexer:
        1 - Read from shell
        2 - Read from file
        3 - Exit
1
Enter code to analyze:
if (_x1 > 0) { return true };
KEYWORD   -> if
LEFT_PARENTHESIS   -> (
ID   -> _x1
LARGER   -> >
INT   -> 0
RIGHT_PARENTHESIS   -> )
LEFT_BRACE   -> {
KEYWORD   -> return
KEYWORD   -> true
RIGHT_BRACE   -> }
SEMICOLON   -> ;
```

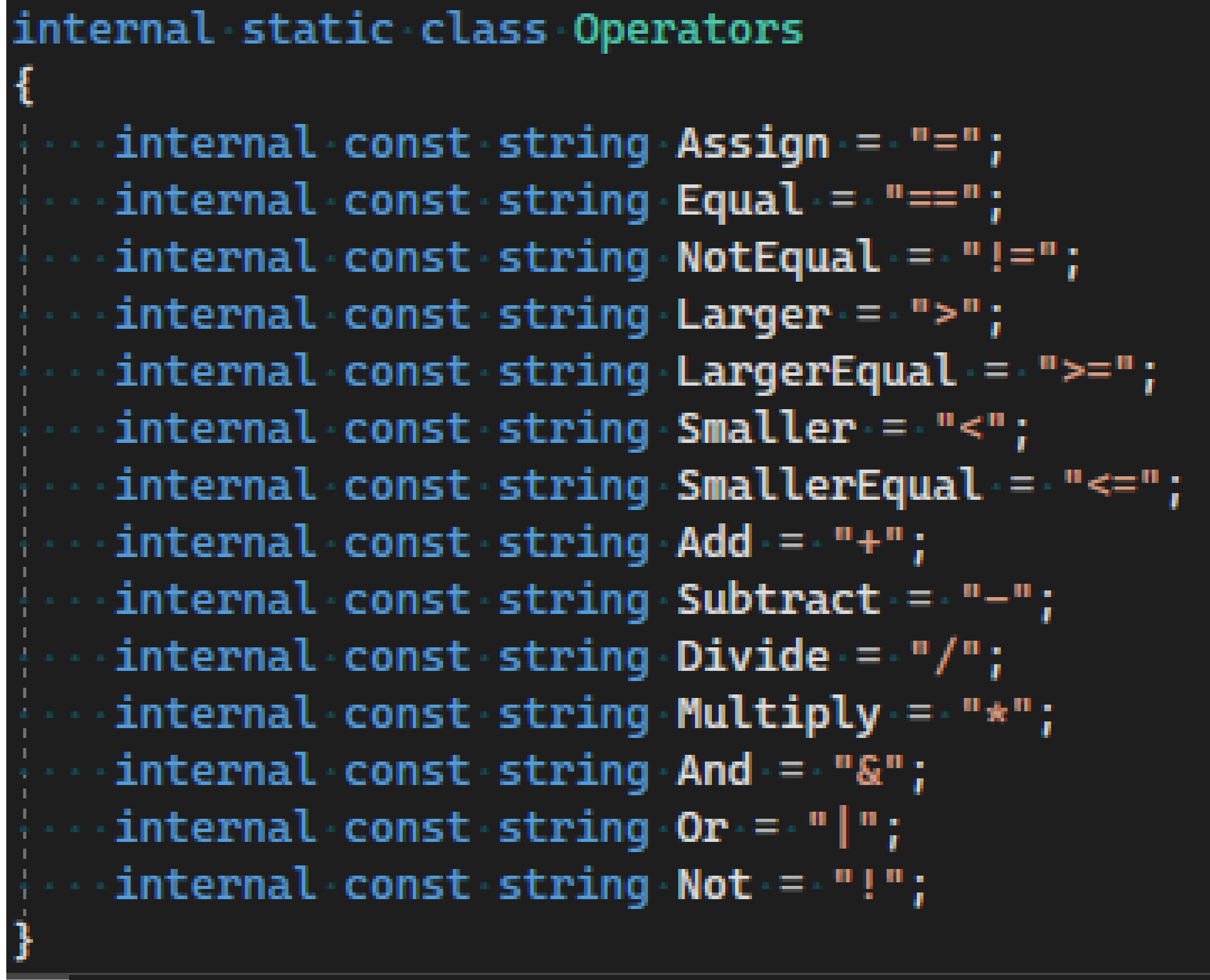


```
Solution 'Lexer' (1 of 1 project)
  Lexer
    Dependencies
    Application
      LexerService.cs
    Domain
      Constants
        Delimiters.cs
        ErrorMessages.cs
        NumberConstants.cs
        Operators.cs
        Patterns.cs
        StringConstants.cs
      Entities
        Result
          LexerError.cs
          LexerResult.cs
        Context.cs
        Token.cs
        TokenType.cs
      Enum
        TokenTypeValue.cs
      ValueObjects
        Tokens.cs
        TokenTypes.cs
    Program.cs
```

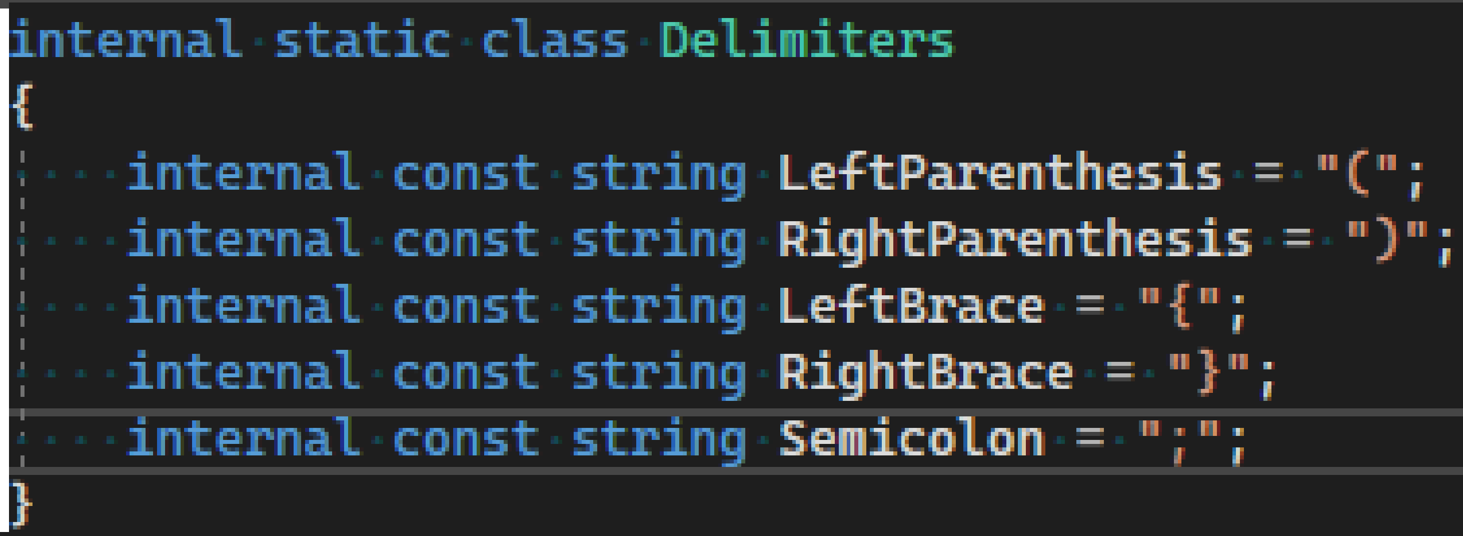
Domínio

Domínio - Constantes

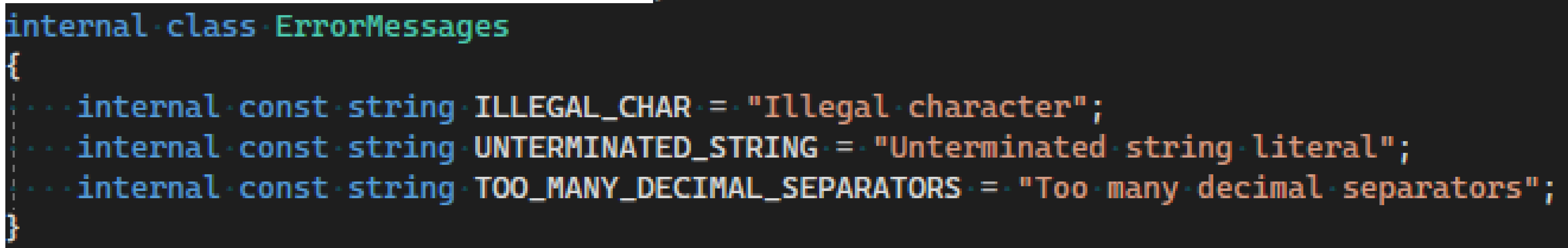
- Operadores e Delimitadores
  - Representações literais
- Padrões
  - Todas Regex, incluindo operadores e delimitadores
- Componentes de string e número
  - Aspas, backlash, separador decimal
- Mensagens de Erro



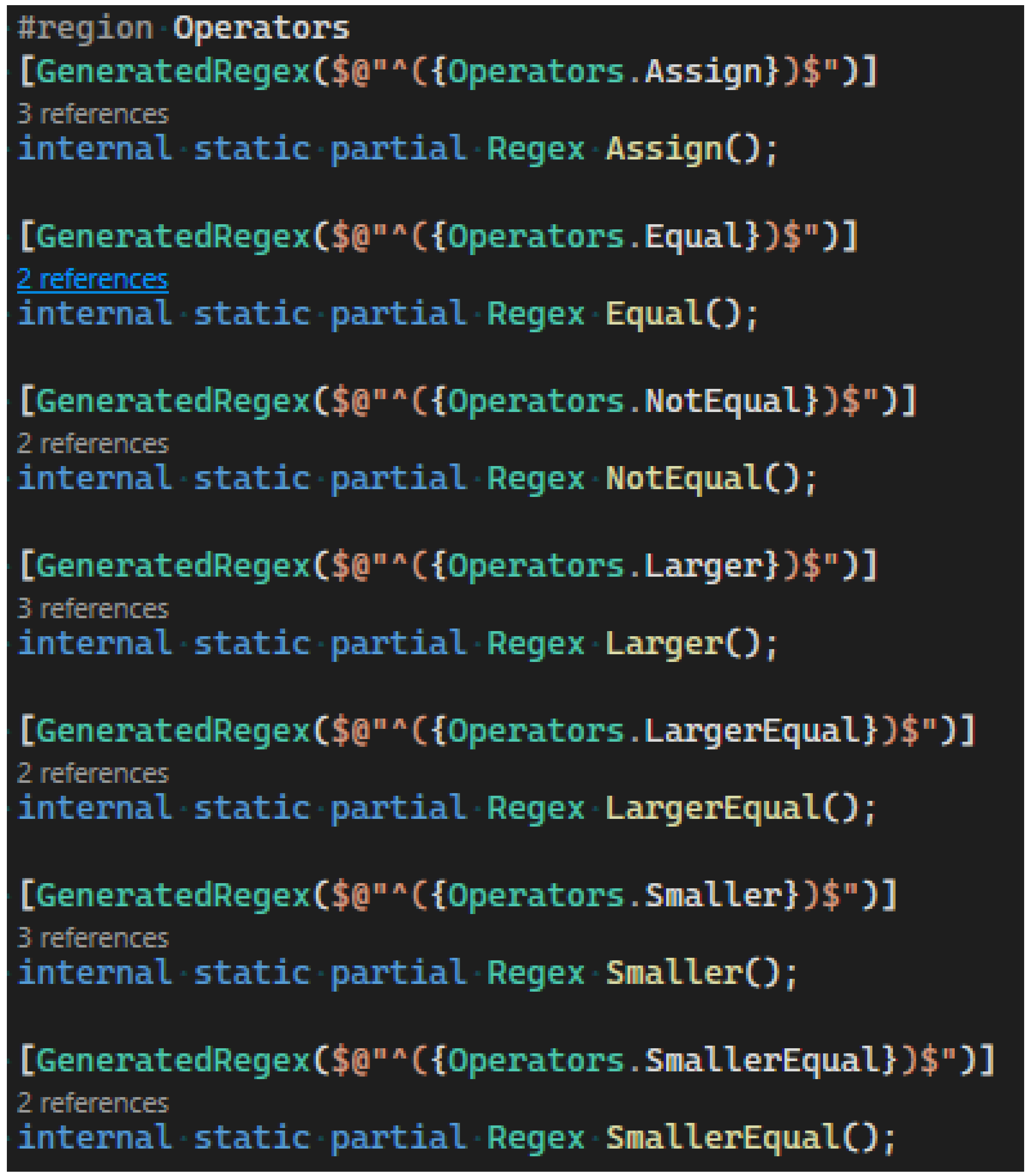
```
internal static class Operators
{
    internal const string Assign = "=";
    internal const string Equal = "==";
    internal const string NotEqual = "!=";
    internal const string Larger = ">";
    internal const string LargerEqual = ">=";
    internal const string Smaller = "<";
    internal const string SmallerEqual = "<=";
    internal const string Add = "+";
    internal const string Subtract = "-";
    internal const string Divide = "/";
    internal const string Multiply = "*";
    internal const string And = "&";
    internal const string Or = "|";
    internal const string Not = "!";
}
```



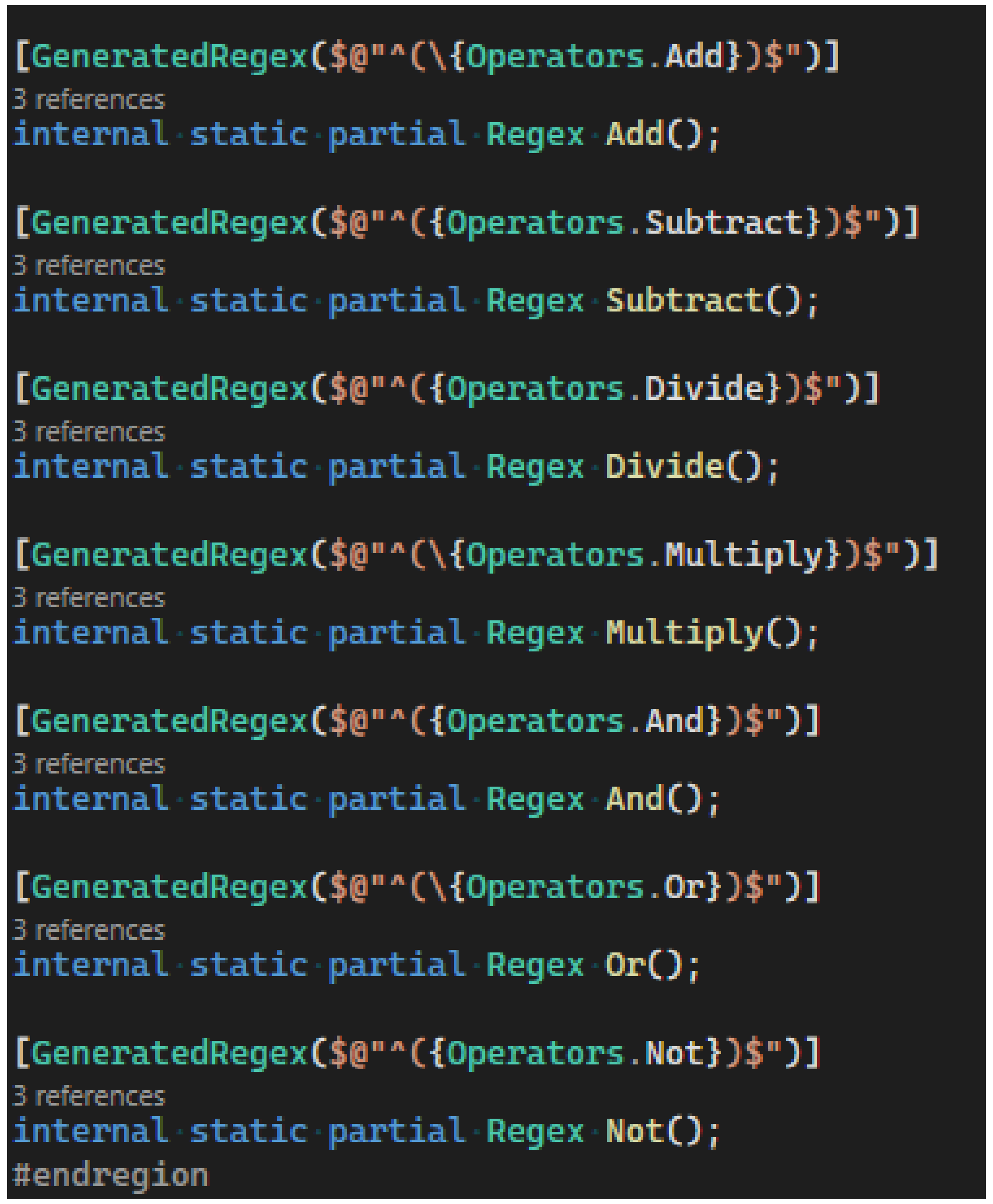
```
internal static class Delimiters
{
    internal const string LeftParenthesis = "(";
    internal const string RightParenthesis = ")";
    internal const string LeftBrace = "{";
    internal const string RightBrace = "}";
    internal const string Semicolon = ";";
}
```



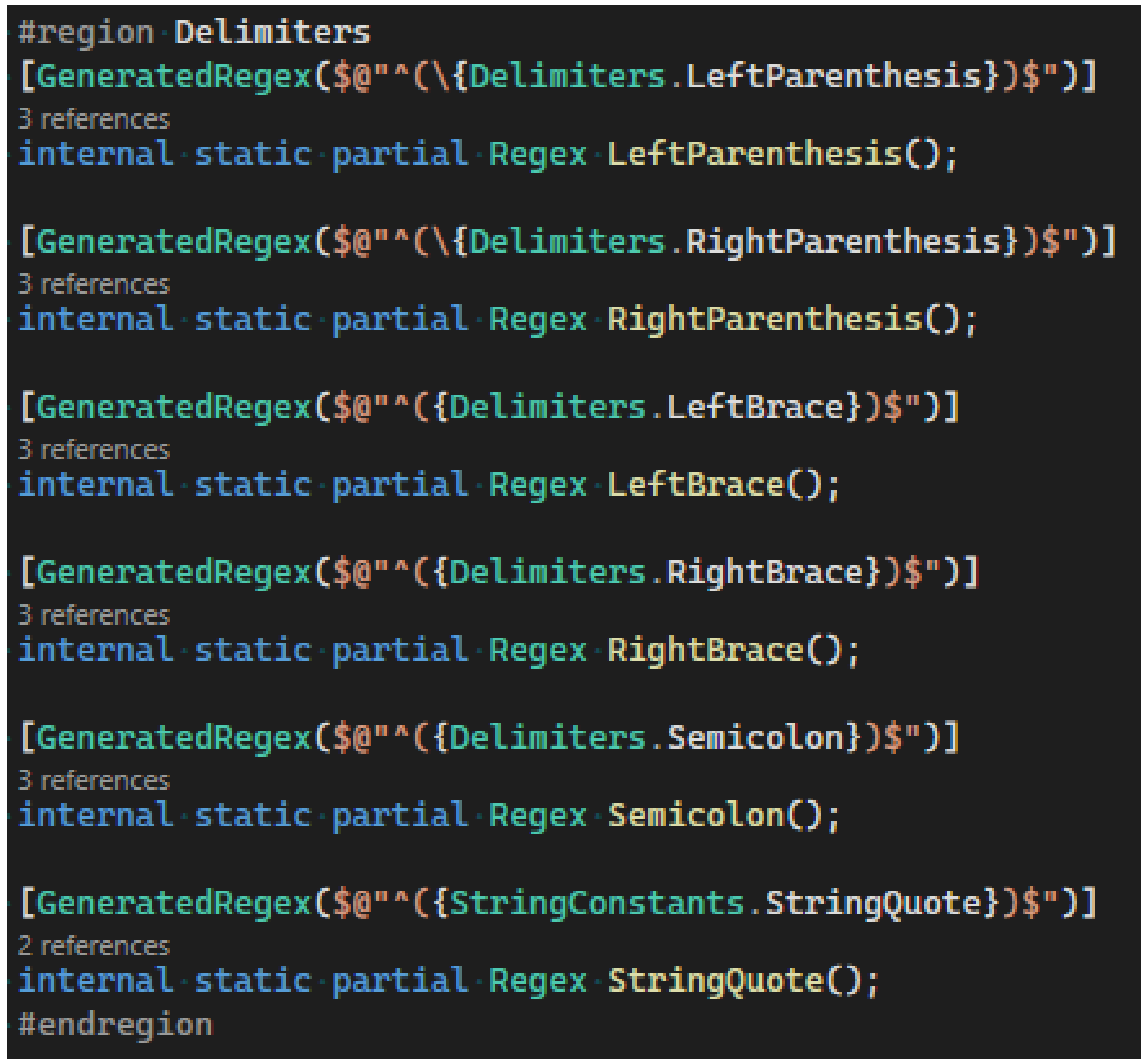
```
internal class ErrorMessages
{
    internal const string ILLEGAL_CHAR = "Illegal character";
    internal const string UNTERMINATED_STRING = "Unterminated string literal";
    internal const string TOO_MANY_DECIMAL_SEPARATORS = "Too many decimal separators";
}
```



```
#region Operators
[GeneratedRegex($@"^({Operators.Assign})$")]
3 references
internal static partial Regex Assign();

[GeneratedRegex($@"^({Operators.Equal})$")]
2 references
internal static partial Regex Equal();

[GeneratedRegex($@"^({Operators.NotEqual})$")]
2 references
internal static partial Regex NotEqual();

[GeneratedRegex($@"^({Operators.Larger})$")]
3 references
internal static partial Regex Larger();

[GeneratedRegex($@"^({Operators.LargerEqual})$")]
2 references
internal static partial Regex LargerEqual();

[GeneratedRegex($@"^({Operators.Smaller})$")]
3 references
internal static partial Regex Smaller();

[GeneratedRegex($@"^({Operators.SmallerEqual})$")]
2 references
internal static partial Regex SmallerEqual();
```



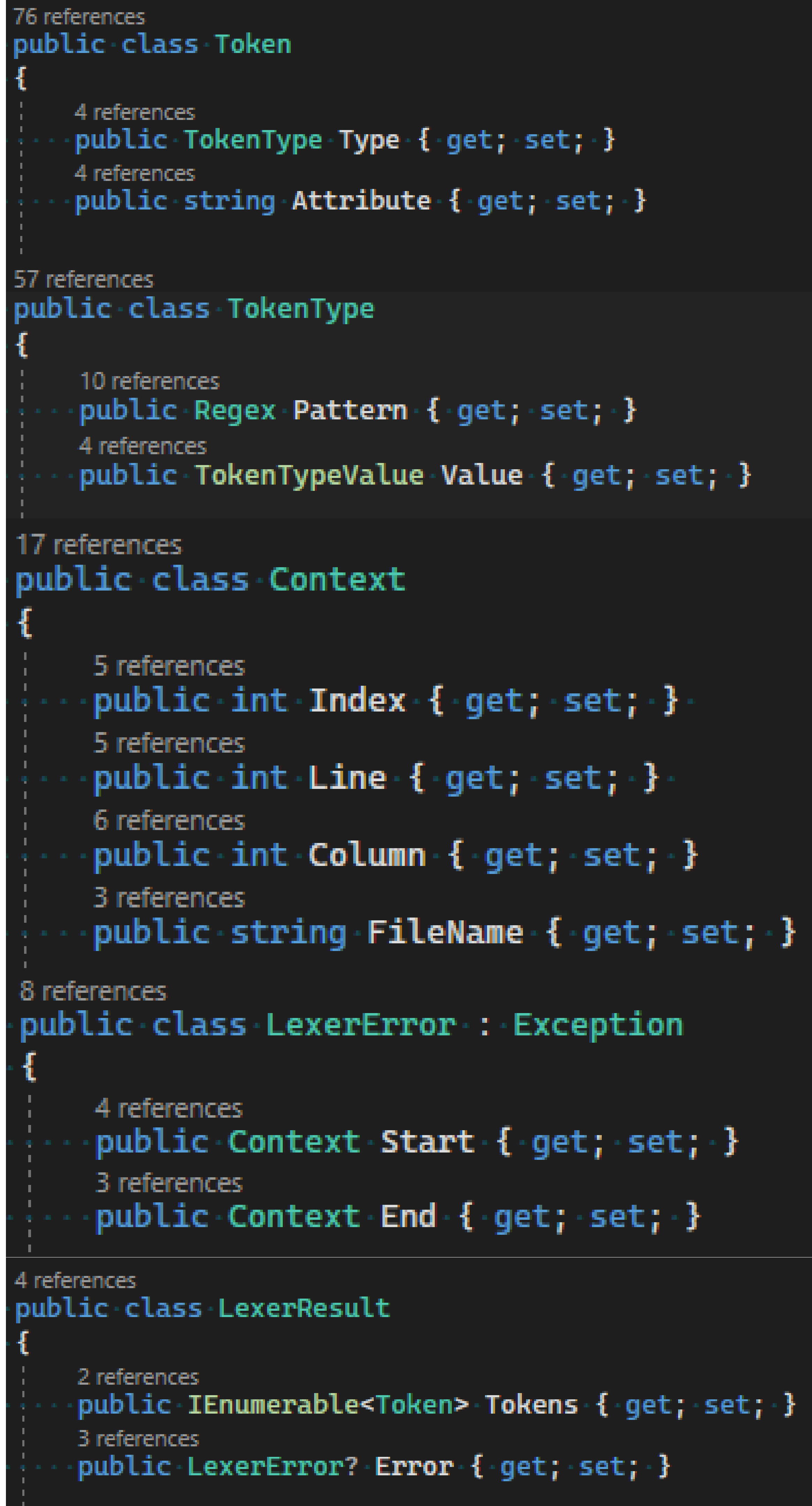
```
[GeneratedRegex($@"^(\{Operators.Add})$")]
3 references
internal static partial Regex Add();

[GeneratedRegex($@"^({Operators.Subtract})$")]
3 references
internal static partial Regex Subtract();

[GeneratedRegex($@"^({Operators.Divide})$")]
3 references
internal static partial Regex Divide();

[GeneratedRegex($@"^(\{Operators.Multiply})$")]
3 references
internal static partial Regex Multiply();

[GeneratedRegex($@"^({Operators.And})$")]
3 references
internal static partial Regex And();

[GeneratedRegex($@"^(\{Operators.Or})$")]
3 references
internal static partial Regex Or();

[GeneratedRegex($@"^({Operators.Not})$")]
3 references
internal static partial Regex Not();
#endregion
```



```
#region Delimiters
[GeneratedRegex($@"^(\{Delimiters.LeftParenthesis})$")]
3 references
internal static partial Regex LeftParenthesis();

[GeneratedRegex($@"^(\{Delimiters.RightParenthesis})$")]
3 references
internal static partial Regex RightParenthesis();

[GeneratedRegex($@"^({Delimiters.LeftBrace})$")]
3 references
internal static partial Regex LeftBrace();

[GeneratedRegex($@"^({Delimiters.RightBrace})$")]
3 references
internal static partial Regex RightBrace();

[GeneratedRegex($@"^({Delimiters.Semicolon})$")]
3 references
internal static partial Regex Semicolon();

[GeneratedRegex($@"^({StringConstants.StringQuote})$")]
2 references
internal static partial Regex StringQuote();
#endregion
```

Domínio - Entidades

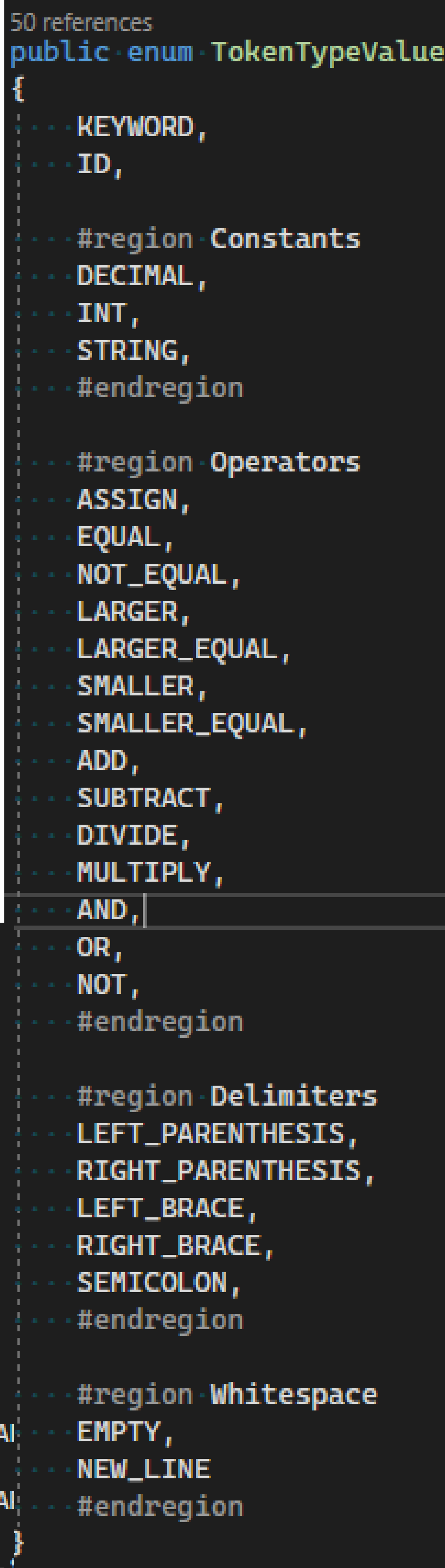
- Erro de Lexer
  - Contexto inicial, final, mensagem
- Resultado
  - Lista de tokens, Erro
- Contexto
  - Índice atual, linha, coluna
  - Nome do arquivo
- Tipo de Token
  - Padrão Regex
  - Nome
- Token
  - Tipo de token
  - Atributo



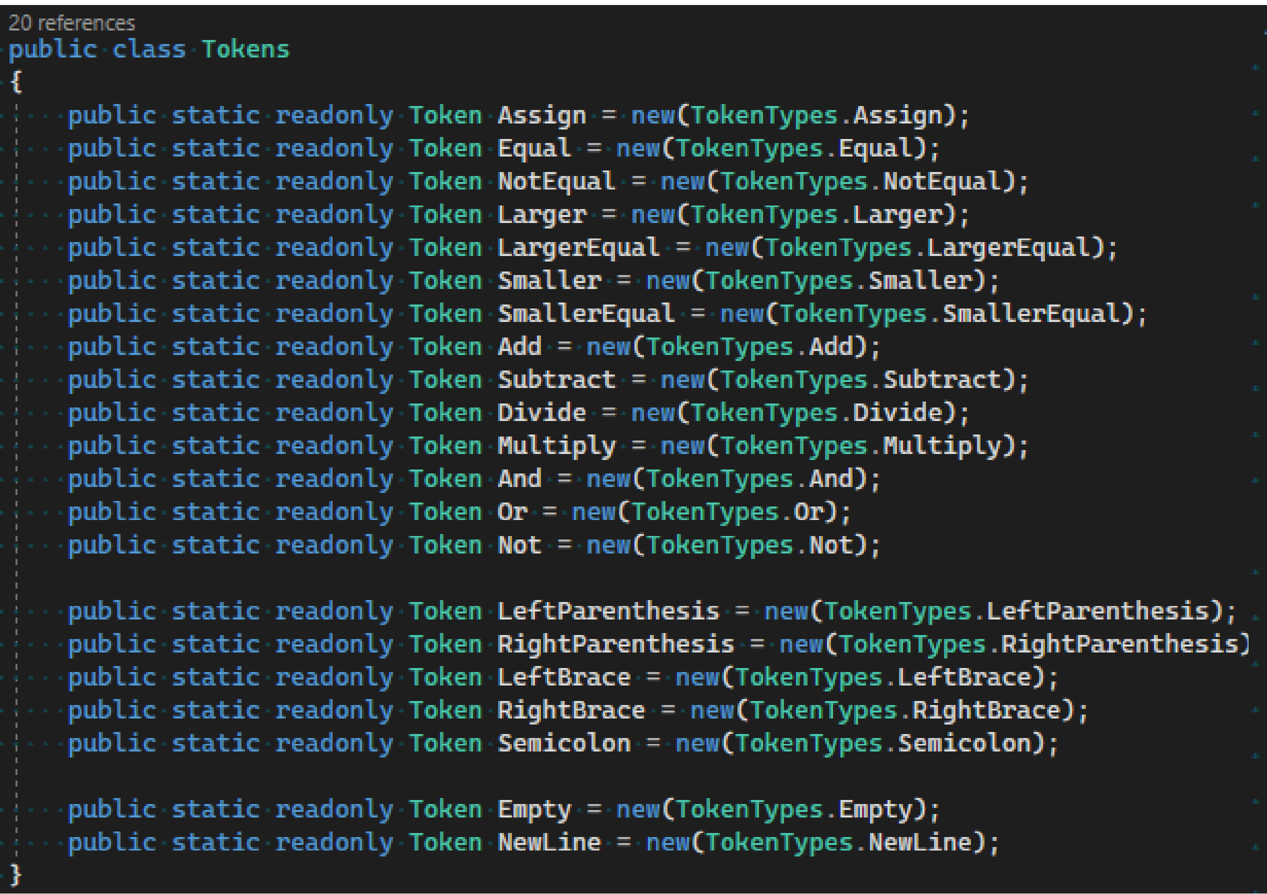
```
76 references
public class Token
{
    4 references
    public TokenType Type { get; set; }
    4 references
    public string Attribute { get; set; }

57 references
public class TokenType
{
    10 references
    public Regex Pattern { get; set; }
    4 references
    public TokenTypeValue Value { get; set; }

17 references
public class Context
{
    5 references
    public int Index { get; set; }
    5 references
    public int Line { get; set; }
    6 references
    public int Column { get; set; }
    3 references
    public string FileName { get; set; }

8 references
public class LexerError : Exception
{
    4 references
    public Context Start { get; set; }
    3 references
    public Context End { get; set; }

4 references
public class LexerResult
{
    2 references
    public IEnumerable<Token> Tokens { get; set; }
    3 references
    public LexerError? Error { get; set; }
```

Domínio - Enum e ValueObjects

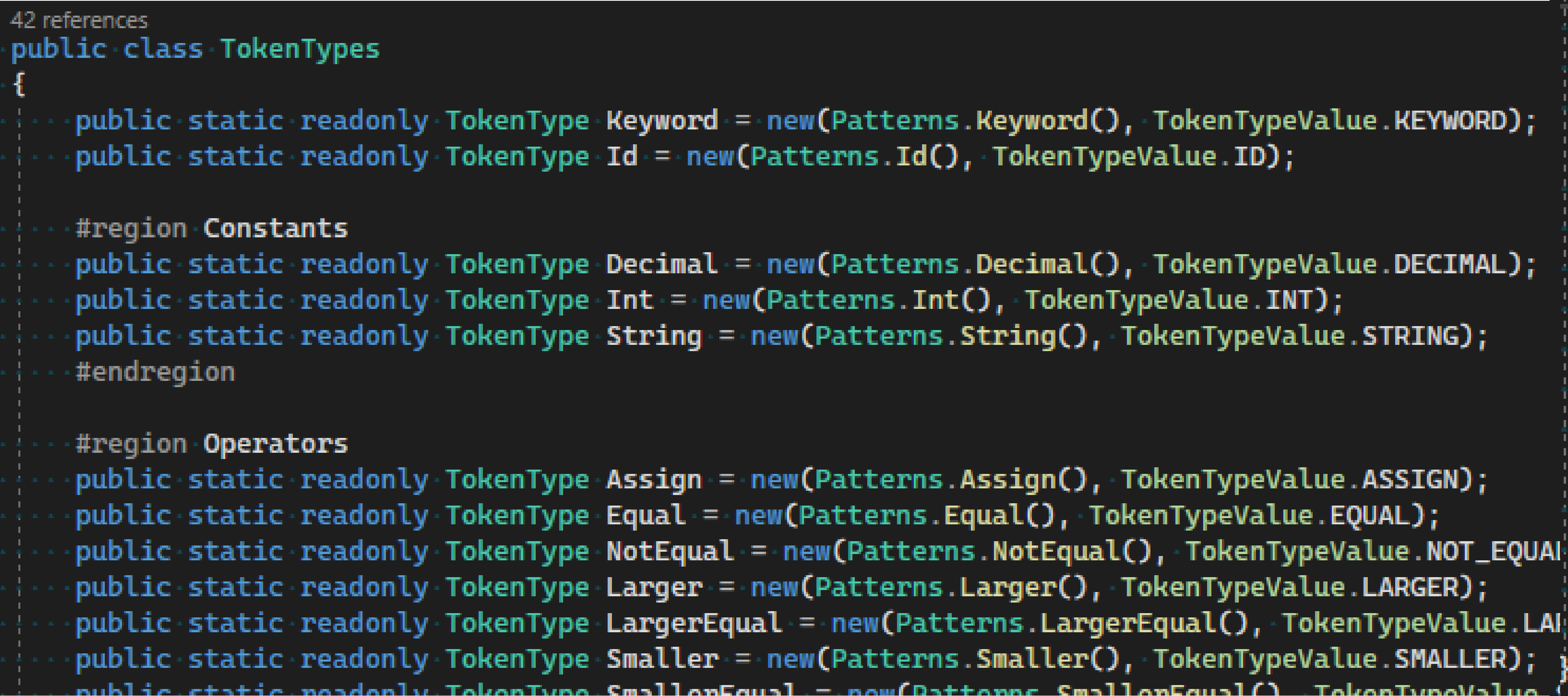
- Nomes de Token
- Tokens sem Atributo
  - Operadores, Delimitadores, Espaços em branco e nova linha
- Tipos de Token



```
50 references
public enum TokenTypeValue
{
    KEYWORD,
    ID,

    #region Constants
    DECIMAL,
    INT,
    STRING,
    #endregion

    #region Operators
    ASSIGN,
    EQUAL,
    NOT_EQUAL,
    LARGER,
    LARGER_EQUAL,
    SMALLER,
    SMALLER_EQUAL,
    ADD,
    SUBTRACT,
    DIVIDE,
    MULTIPLY,
    AND,
    OR,
    NOT,
    #endregion

    #region Delimiters
    LEFT_PARENTHESIS,
    RIGHT_PARENTHESIS,
    LEFT_BRACE,
    RIGHT_BRACE,
    SEMICOLON,
    #endregion

    #region Whitespace
    EMPTY,
    NEW_LINE
    #endregion
}
```



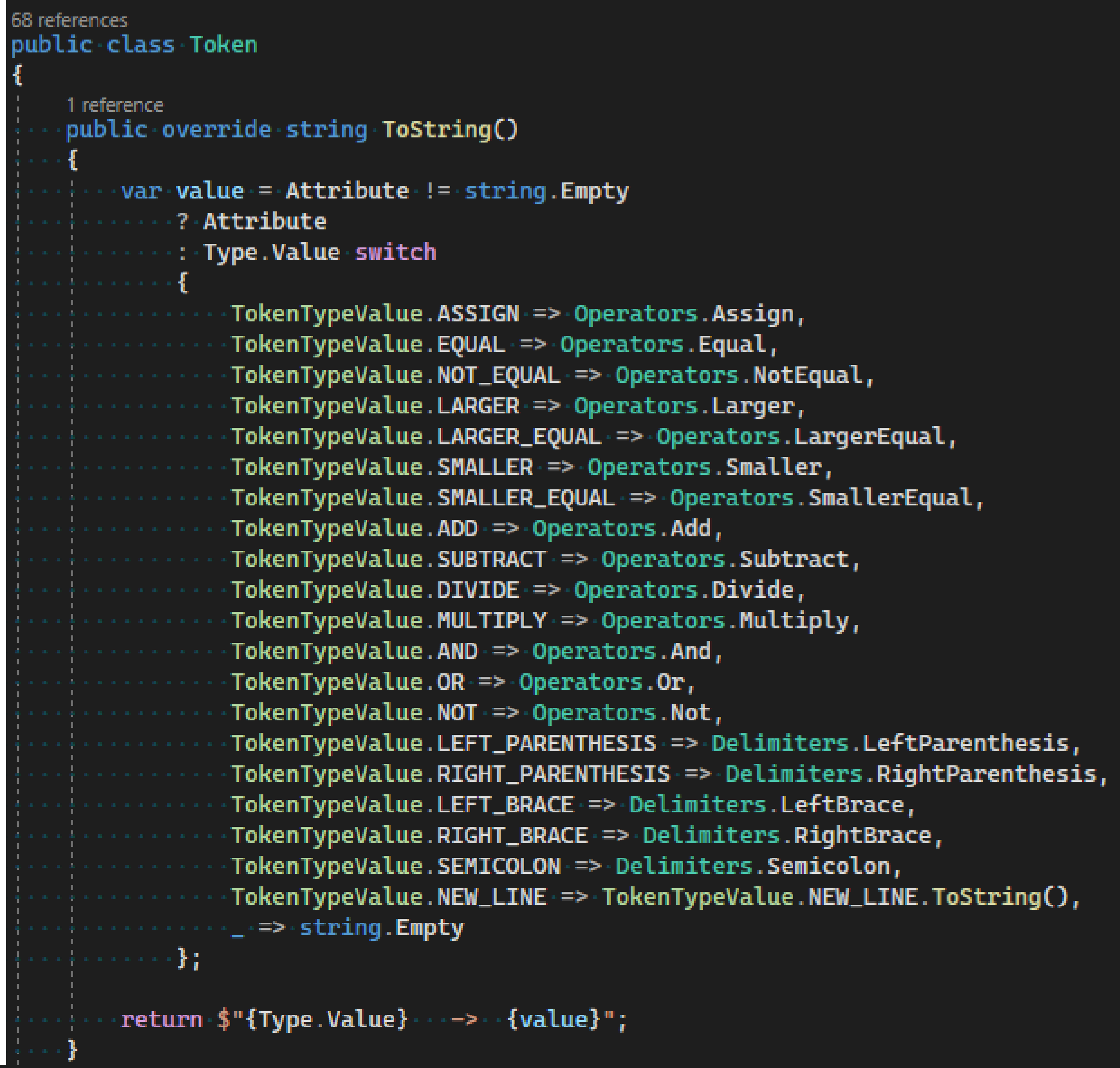
```
20 references
public class Tokens
{
    public static readonly Token Assign = new(TokenTypes.Assign);
    public static readonly Token Equal = new(TokenTypes.Equal);
    public static readonly Token NotEqual = new(TokenTypes.NotEqual);
    public static readonly Token Larger = new(TokenTypes.Larger);
    public static readonly Token LargerEqual = new(TokenTypes.LargerEqual);
    public static readonly Token Smaller = new(TokenTypes.Smaller);
    public static readonly Token SmallerEqual = new(TokenTypes.SmallerEqual);
    public static readonly Token Add = new(TokenTypes.Add);
    public static readonly Token Subtract = new(TokenTypes.Subtract);
    public static readonly Token Divide = new(TokenTypes.Divide);
    public static readonly Token Multiply = new(TokenTypes.Multiply);
    public static readonly Token And = new(TokenTypes.And);
    public static readonly Token Or = new(TokenTypes.Or);
    public static readonly Token Not = new(TokenTypes.Not);

    public static readonly Token LeftParenthesis = new(TokenTypes.LeftParenthesis);
    public static readonly Token RightParenthesis = new(TokenTypes.RightParenthesis)
    public static readonly Token LeftBrace = new(TokenTypes.LeftBrace);
    public static readonly Token RightBrace = new(TokenTypes.RightBrace);
    public static readonly Token Semicolon = new(TokenTypes.Semicolon);

    public static readonly Token Empty = new(TokenTypes.Empty);
    public static readonly Token NewLine = new(TokenTypes.NewLine);
}
```



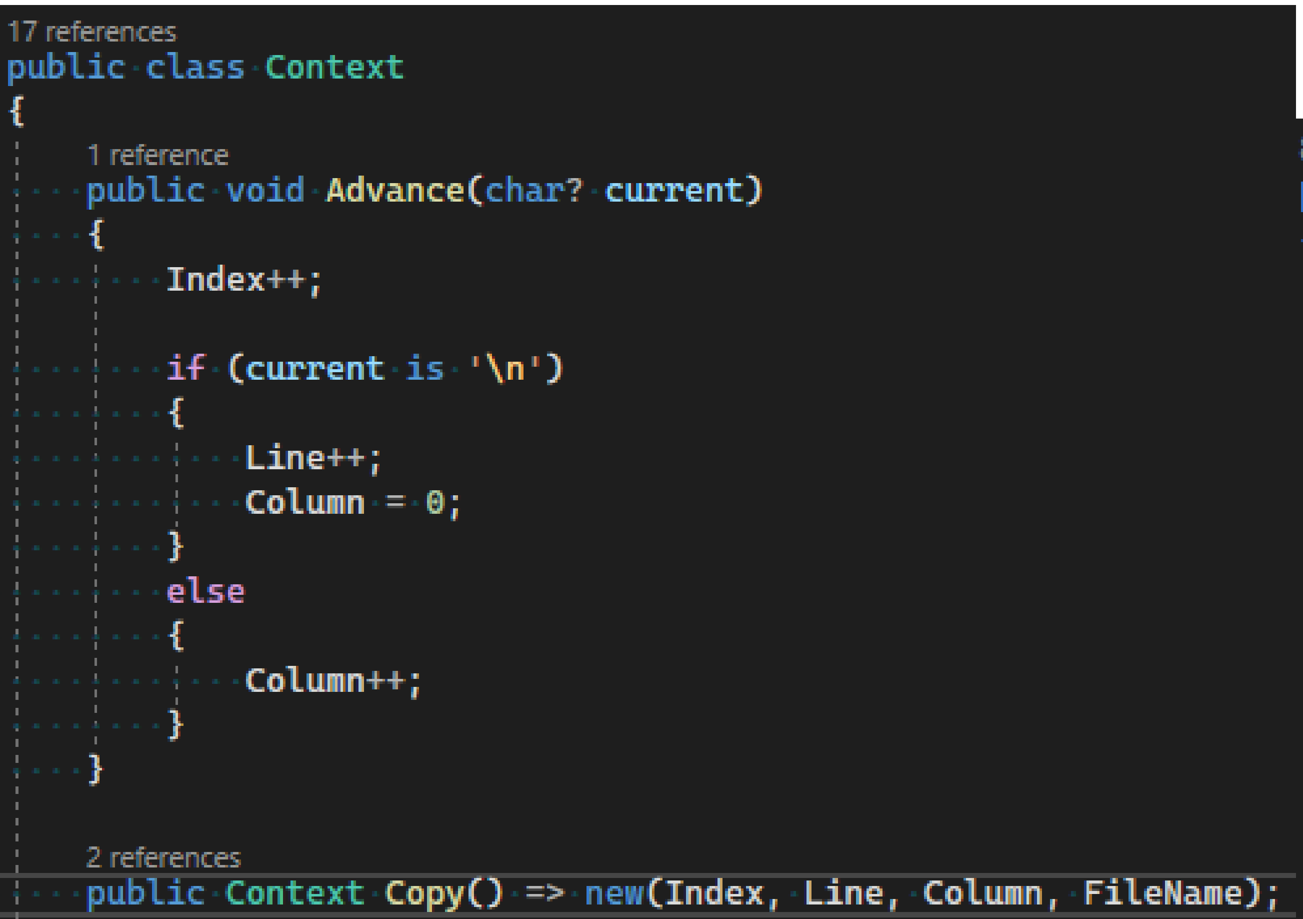
```
42 references
public class TokenTypes
{
    public static readonly TokenType Keyword = new(Patterns.Keyword(), TokenTypeValue.KEYWORD);
    public static readonly TokenType Id = new(Patterns.Id(), TokenTypeValue.ID);

    #region Constants
    public static readonly TokenType Decimal = new(Patterns.Decimal(), TokenTypeValue.DECIMAL);
    public static readonly TokenType Int = new(Patterns.Int(), TokenTypeValue.INT);
    public static readonly TokenType String = new(Patterns.String(), TokenTypeValue.STRING);
    #endregion

    #region Operators
    public static readonly TokenType Assign = new(Patterns.Assign(), TokenTypeValue.ASSIGN);
    public static readonly TokenType Equal = new(Patterns.Equal(), TokenTypeValue.EQUAL);
    public static readonly TokenType NotEqual = new(Patterns.NotEqual(), TokenTypeValue.NOT_EQUA
    public static readonly TokenType Larger = new(Patterns.Larger(), TokenTypeValue.LARGER);
    public static readonly TokenType LargerEqual = new(Patterns.LargerEqual(), TokenTypeValue.LA
    public static readonly TokenType Smaller = new(Patterns.Smaller(), TokenTypeValue.SMALLER);
    public static readonly TokenType SmallerEqual = new(Patterns.SmallerEqual(), TokenTypeValue
```

Domínio - Funções

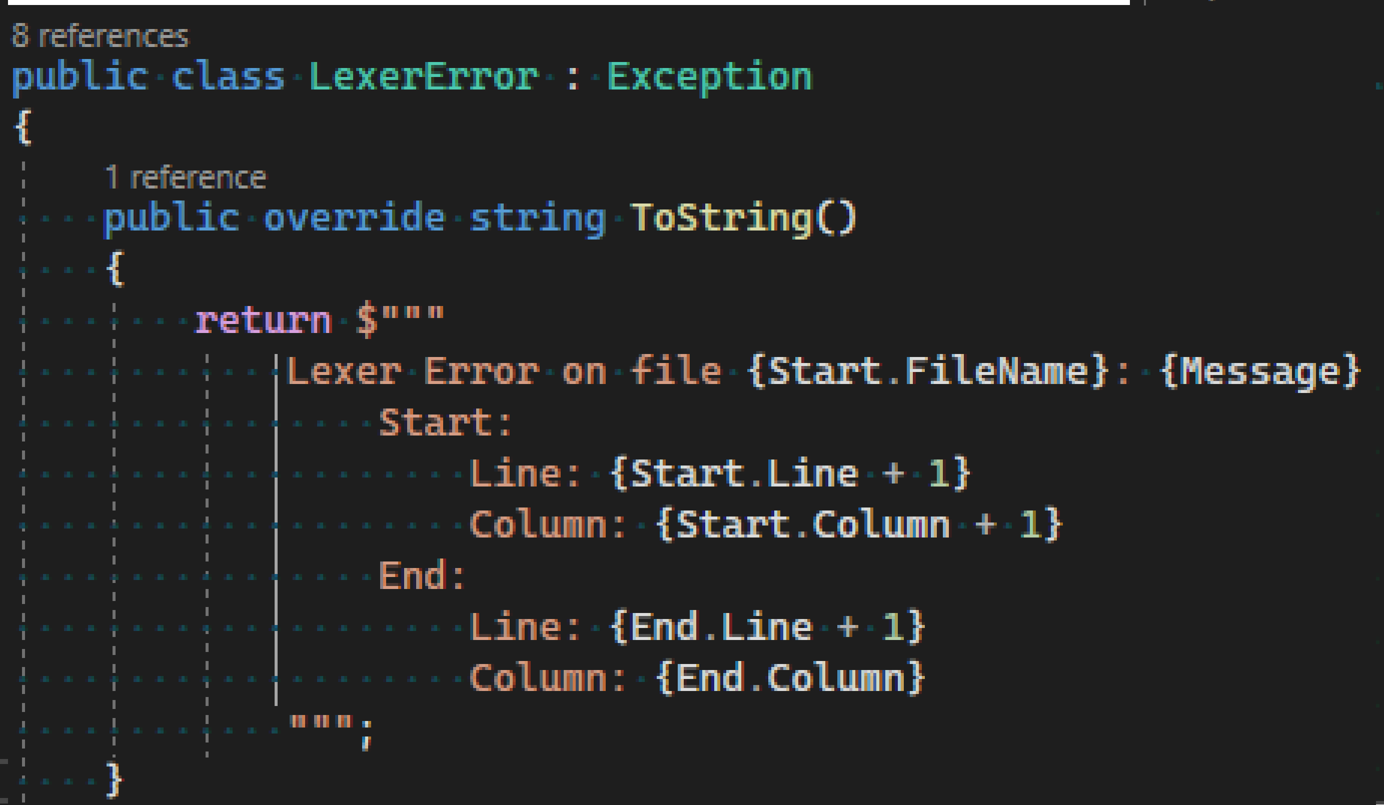
- Contexto
  - Avançar, copiar
- Token, Resultado de Lexer, Erro de Lexer
  - ToString
- Resultado de Lexer
  - Validar



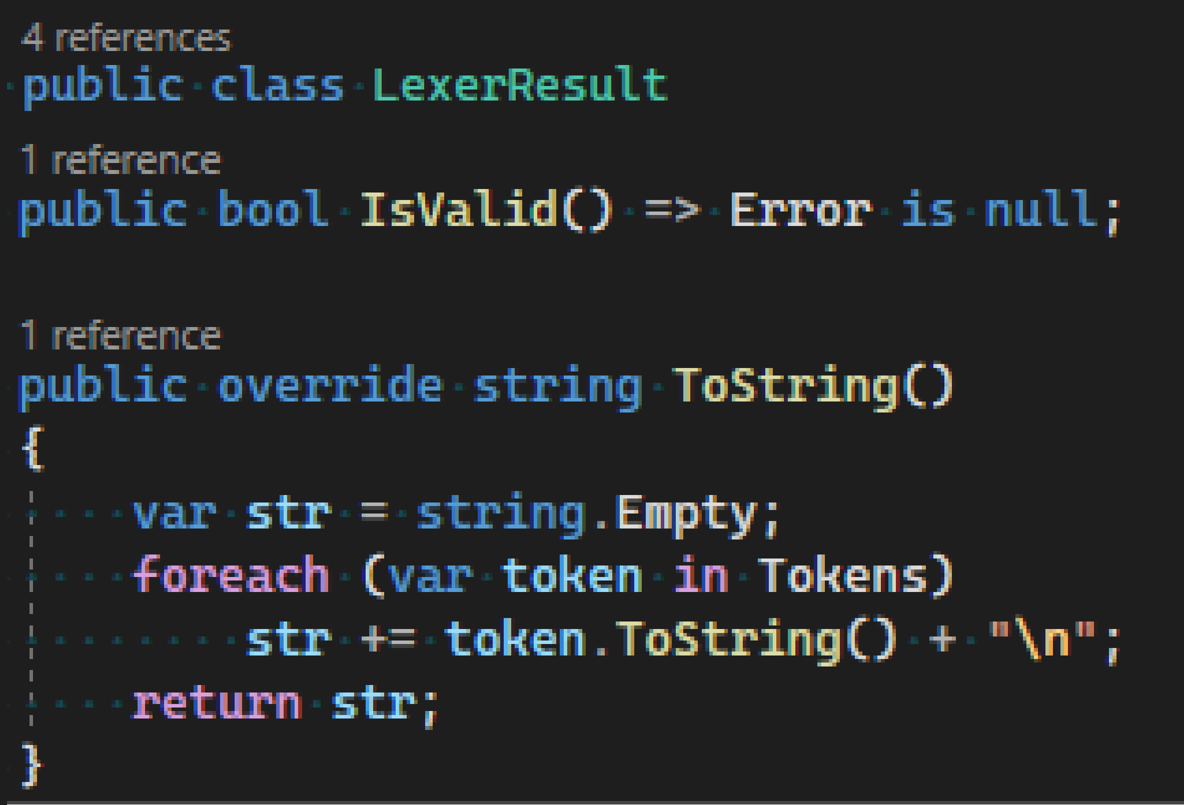
```csharp
68 references
public class Token
{
    1 reference
    public override string ToString()
    {
        var value = Attribute != string.Empty
            ? Attribute
            : Type.Value switch
            {
                TokenTypeValue.ASSIGN => Operators.Assign,
                TokenTypeValue.EQUAL => Operators.Equal,
                TokenTypeValue.NOT_EQUAL => Operators.NotEqual,
                TokenTypeValue.LARGER => Operators.Larger,
                TokenTypeValue.LARGER_EQUAL => Operators.LargerEqual,
                TokenTypeValue.SMALLER => Operators.Smaller,
                TokenTypeValue.SMALLER_EQUAL => Operators.SmallerEqual,
                TokenTypeValue.ADD => Operators.Add,
                TokenTypeValue.SUBTRACT => Operators.Subtract,
                TokenTypeValue.DIVIDE => Operators.Divide,
                TokenTypeValue.MULTIPLY => Operators.Multiply,
                TokenTypeValue.AND => Operators.And,
                TokenTypeValue.OR => Operators.Or,
                TokenTypeValue.NOT => Operators.Not,
                TokenTypeValue.LEFT_PARENTHESIS => Delimiters.LeftParenthesis,
                TokenTypeValue.RIGHT_PARENTHESIS => Delimiters.RightParenthesis,
                TokenTypeValue.LEFT_BRACE => Delimiters.LeftBrace,
                TokenTypeValue.RIGHT_BRACE => Delimiters.RightBrace,
                TokenTypeValue.SEMICOLON => Delimiters.Semicolon,
                TokenTypeValue.NEW_LINE => TokenTypeValue.NEW_LINE.ToString(),
                _ => string.Empty
            };

        return $"{Type.Value}   ->  {value}";
    }
```



```csharp
17 references
public class Context
{
    1 reference
    public void Advance(char? current)
    {
        Index++;

        if (current is '\n')
        {
            Line++;
            Column = 0;
        }
        else
        {
            Column++;
        }
    }

    2 references
    public Context Copy() => new(Index, Line, Column, FileName);
```



```csharp
8 references
public class LexerError : Exception
{
    1 reference
    public override string ToString()
    {
        return $"""
            Lexer Error on file {Start.FileName}: {Message}
                Start:
                    Line: {Start.Line + 1}
                    Column: {Start.Column + 1}
                End:
                    Line: {End.Line + 1}
                    Column: {End.Column}
            """;
    }
```

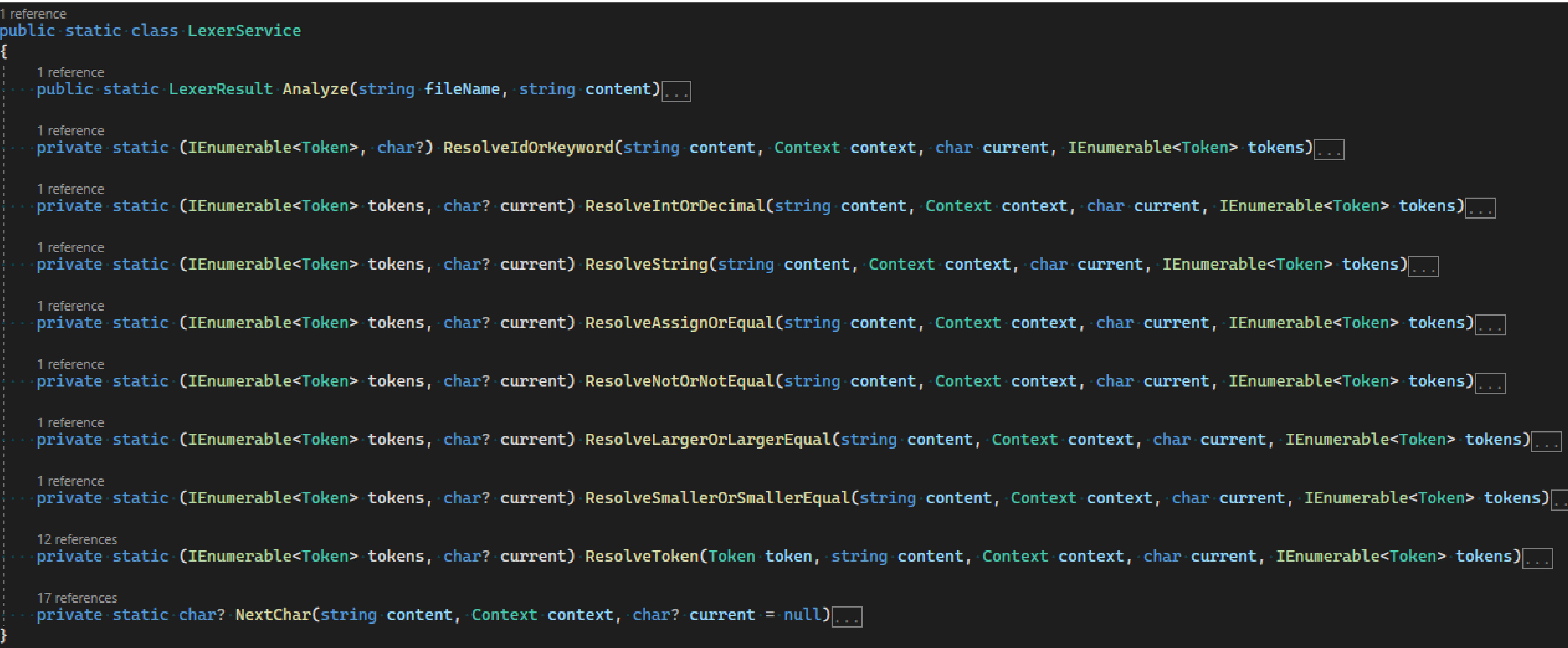


```csharp
4 references
public class LexerResult

1 reference
public bool IsValid() => Error is null;

1 reference
public override string ToString()
{
    var str = string.Empty;
    foreach (var token in Tokens)
        str += token.ToString() + "\n";
    return str;
}
```

Aplicação

Aplicação

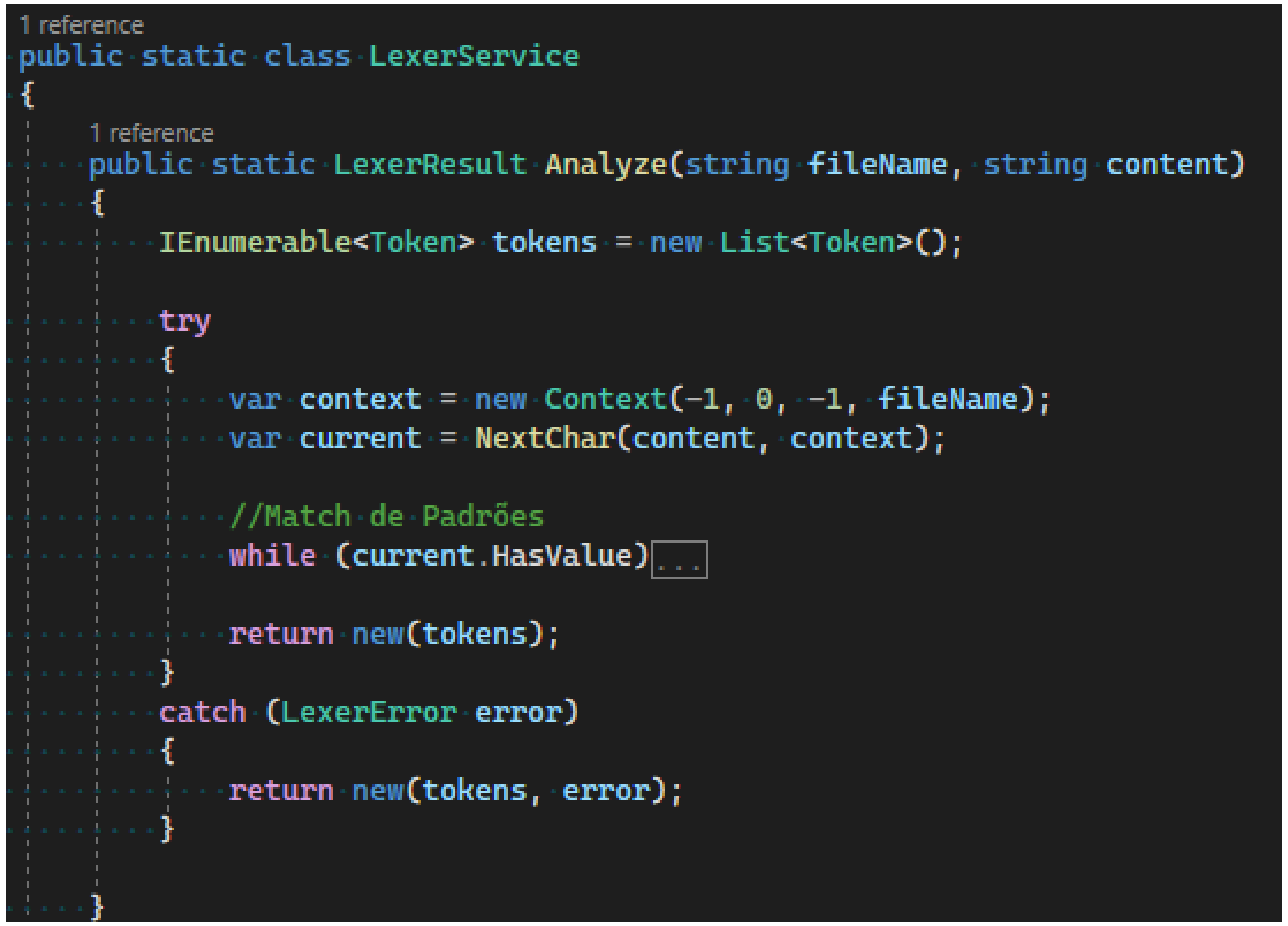
- Somente uma classe: Serviço de Lexer



```
1 reference
public static class LexerService
{
    1 reference
    public static LexerResult Analyze(string fileName, string content)...

    1 reference
    private static (IEnumerable<Token>, char?) ResolveIdOrKeyword(string content, Context context, char current, IEnumerable<Token> tokens)...

    1 reference
    private static (IEnumerable<Token> tokens, char? current) ResolveIntOrDecimal(string content, Context context, char current, IEnumerable<Token> tokens)...

    1 reference
    private static (IEnumerable<Token> tokens, char? current) ResolveString(string content, Context context, char current, IEnumerable<Token> tokens)...

    1 reference
    private static (IEnumerable<Token> tokens, char? current) ResolveAssignOrEqual(string content, Context context, char current, IEnumerable<Token> tokens)...

    1 reference
    private static (IEnumerable<Token> tokens, char? current) ResolveNotOrNotEqual(string content, Context context, char current, IEnumerable<Token> tokens)...

    1 reference
    private static (IEnumerable<Token> tokens, char? current) ResolveLargerOrLargerEqual(string content, Context context, char current, IEnumerable<Token> tokens)...

    1 reference
    private static (IEnumerable<Token> tokens, char? current) ResolveSmallerOrSmallerEqual(string content, Context context, char current, IEnumerable<Token> tokens)

    12 references
    private static (IEnumerable<Token> tokens, char? current) ResolveToken(Token token, string content, Context context, char current, IEnumerable<Token> tokens)...

    17 references
    private static char? NextChar(string content, Context context, char? current = null)...
}
```

Serviço de Lexer - Análise

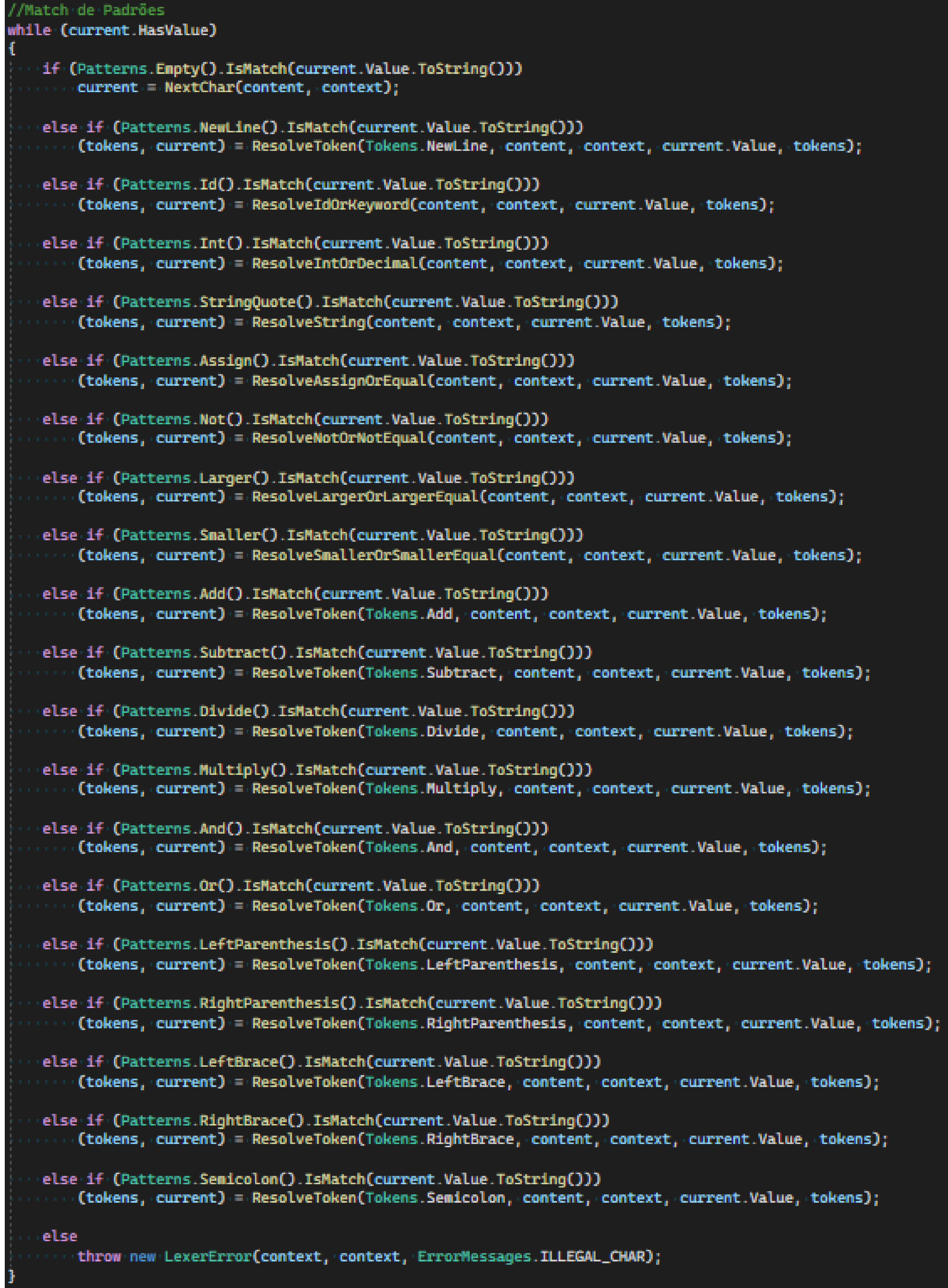
- Inicializa contexto, lista de tokens
- Percorre cada carácter do texto de entrada
- Casa carácter com início de padrões
- Aplica processo para cada carácter dento do padrão até obter lexema completo
- Retorna resultado com lista de tokens, e erro caso presente



```
1 reference
public static class LexerService
{
    1 reference
    public static LexerResult Analyze(string fileName, string content)
    {
        IEnumerable<Token> tokens = new List<Token>();

        try
        {
            var context = new Context(-1, 0, -1, fileName);
            var current = NextChar(content, context);

            //Match de Padrões
            while (current.HasValue)...

            return new(tokens);
        }
        catch (LexerError error)
        {
            return new(tokens, error);
        }

    }
```

Análise - Matching

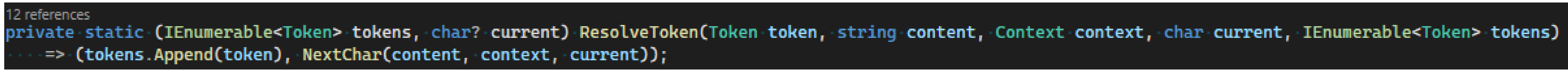
- Não gera token
  - Vazio
- Match direto:
  - Nova linha, adição, subtração, divisão, multiplicação, e, ou, parênteses, chaves, ponto e vírgula
- Match composto simples:
  - Atribuição ou igualdade: = | ==
  - Negação ou diferença: ! | !=
  - Maior ou Maior Igual: > | >=
  - Menor ou Menor Igual: < | <=
- Match composto complexo:
  - Identificador ou palavra-chave
  - Inteiro ou decimal
  - String



```
//Match de Padrões
while (current.HasValue)
{
    if (Patterns.Empty().IsMatch(current.Value.ToString()))
        current = NextChar(content, context);

    else if (Patterns.NewLine().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveToken(Tokens.NewLine, content, context, current.Value, tokens);

    else if (Patterns.Id().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveIdOrKeyword(content, context, current.Value, tokens);

    else if (Patterns.Int().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveIntOrDecimal(content, context, current.Value, tokens);

    else if (Patterns.StringQuote().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveString(content, context, current.Value, tokens);

    else if (Patterns.Assign().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveAssignOrEqual(content, context, current.Value, tokens);

    else if (Patterns.Not().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveNotOrNotEqual(content, context, current.Value, tokens);

    else if (Patterns.Larger().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveLargerOrLargerEqual(content, context, current.Value, tokens);

    else if (Patterns.Smaller().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveSmallerOrSmallerEqual(content, context, current.Value, tokens);

    else if (Patterns.Add().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveToken(Tokens.Add, content, context, current.Value, tokens);

    else if (Patterns.Subtract().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveToken(Tokens.Subtract, content, context, current.Value, tokens);

    else if (Patterns.Divide().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveToken(Tokens.Divide, content, context, current.Value, tokens);

    else if (Patterns.Multiply().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveToken(Tokens.Multiply, content, context, current.Value, tokens);

    else if (Patterns.And().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveToken(Tokens.And, content, context, current.Value, tokens);

    else if (Patterns.Or().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveToken(Tokens.Or, content, context, current.Value, tokens);

    else if (Patterns.LeftParenthesis().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveToken(Tokens.LeftParenthesis, content, context, current.Value, tokens);

    else if (Patterns.RightParenthesis().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveToken(Tokens.RightParenthesis, content, context, current.Value, tokens);

    else if (Patterns.LeftBrace().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveToken(Tokens.LeftBrace, content, context, current.Value, tokens);

    else if (Patterns.RightBrace().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveToken(Tokens.RightBrace, content, context, current.Value, tokens);

    else if (Patterns.Semicolon().IsMatch(current.Value.ToString()))
        (tokens, current) = ResolveToken(Tokens.Semicolon, content, context, current.Value, tokens);

    else
        throw new LexerError(context, context, ErrorMessages.ILLEGAL_CHAR);
}
```

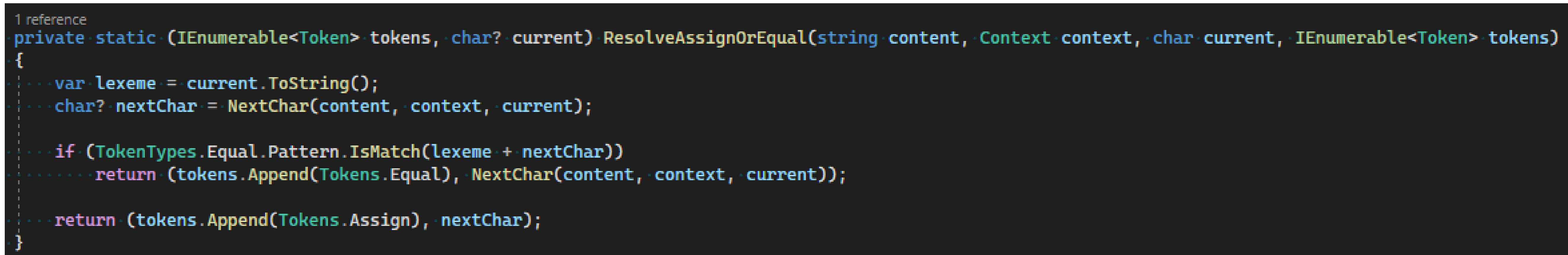
Análise - Matching

- Match direto:



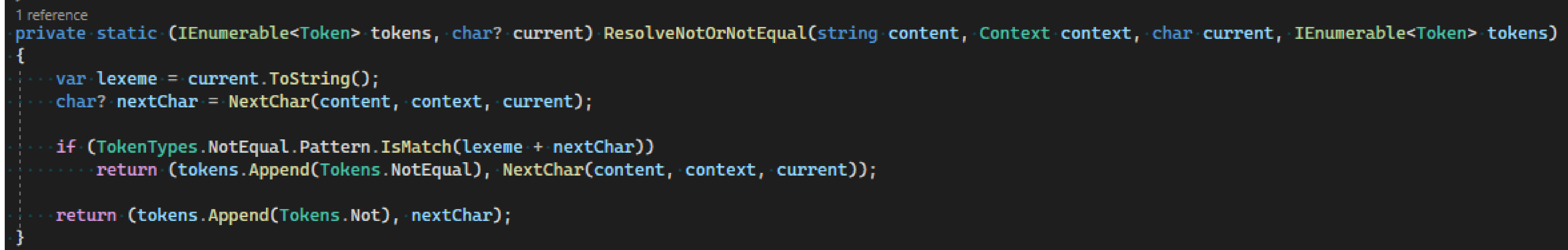
```
12 references
private static (IEnumerable<Token> tokens, char? current) ResolveToken(Token token, string content, Context context, char current, IEnumerable<Token> tokens)
    => (tokens.Append(token), NextChar(content, context, current));
```

- Match composto simples:
  - = | ==



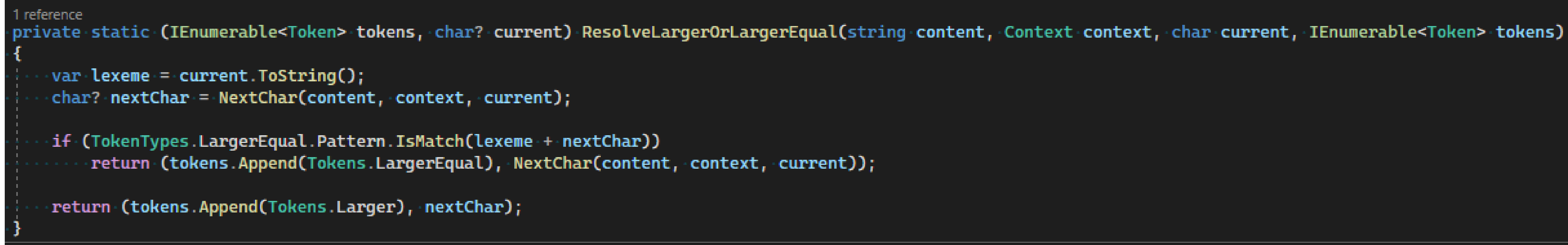
```
1 reference
private static (IEnumerable<Token> tokens, char? current) ResolveAssignOrEqual(string content, Context context, char current, IEnumerable<Token> tokens)
{
    var lexeme = current.ToString();
    char? nextChar = NextChar(content, context, current);

    if (TokenTypes.Equal.Pattern.IsMatch(lexeme + nextChar))
        return (tokens.Append(Tokens.Equal), NextChar(content, context, current));

    return (tokens.Append(Tokens.Assign), nextChar);
}
```

  - ! | !=



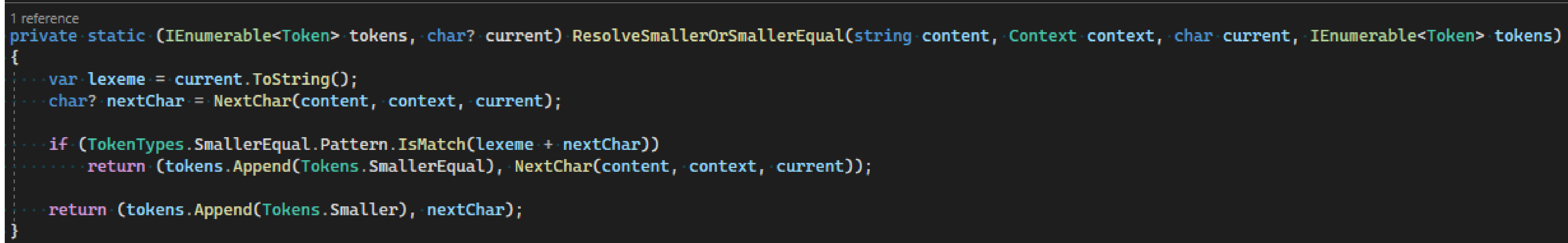
```
1 reference
private static (IEnumerable<Token> tokens, char? current) ResolveNotOrNotEqual(string content, Context context, char current, IEnumerable<Token> tokens)
{
    var lexeme = current.ToString();
    char? nextChar = NextChar(content, context, current);

    if (TokenTypes.NotEqual.Pattern.IsMatch(lexeme + nextChar))
        return (tokens.Append(Tokens.NotEqual), NextChar(content, context, current));

    return (tokens.Append(Tokens.Not), nextChar);
}
```

  - > | >=



```
1 reference
private static (IEnumerable<Token> tokens, char? current) ResolveLargerOrLargerEqual(string content, Context context, char current, IEnumerable<Token> tokens)
{
    var lexeme = current.ToString();
    char? nextChar = NextChar(content, context, current);

    if (TokenTypes.LargerEqual.Pattern.IsMatch(lexeme + nextChar))
        return (tokens.Append(Tokens.LargerEqual), NextChar(content, context, current));

    return (tokens.Append(Tokens.Larger), nextChar);
}
```

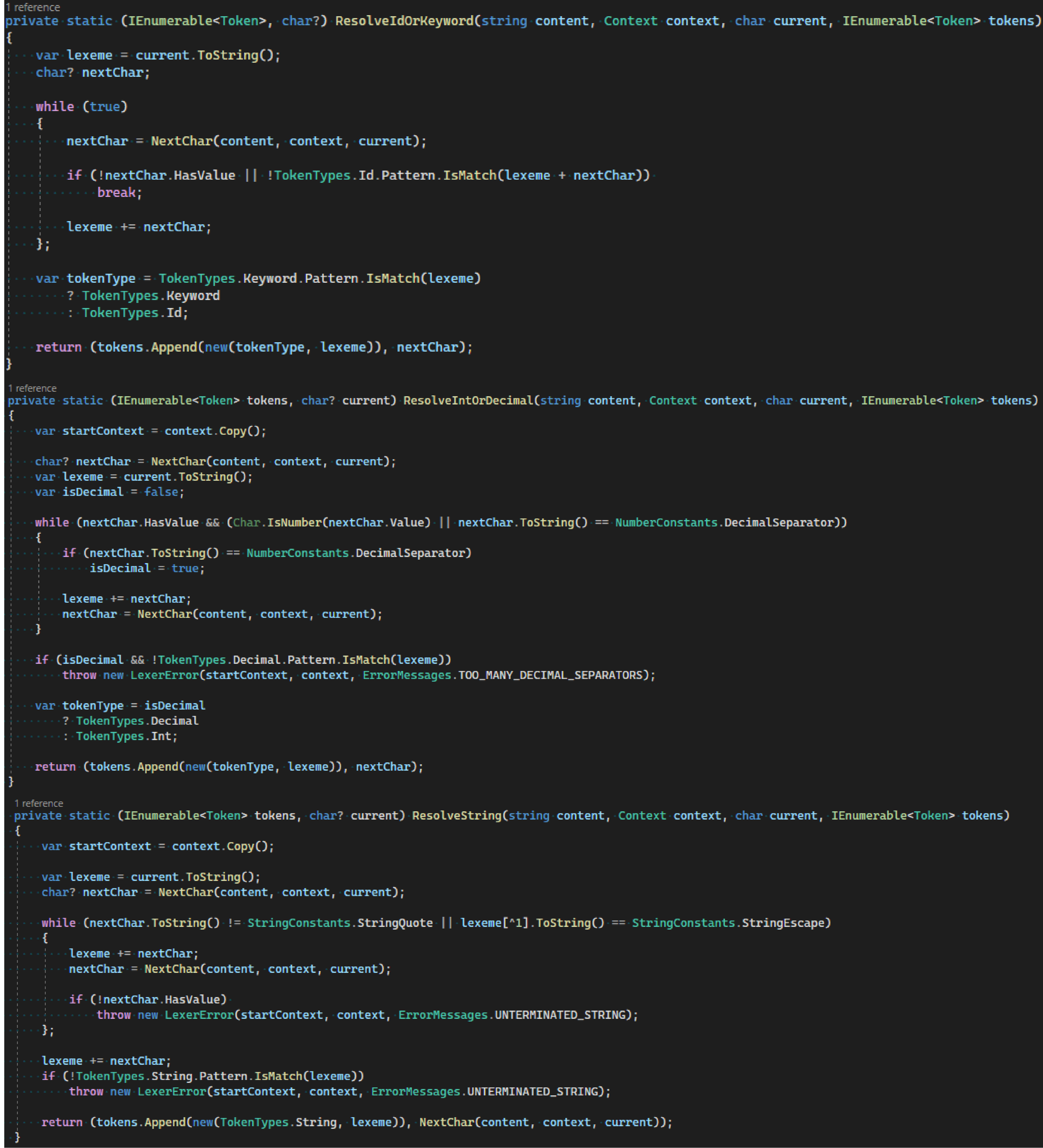
  - < | <=



```
1 reference
private static (IEnumerable<Token> tokens, char? current) ResolveSmallerOrSmallerEqual(string content, Context context, char current, IEnumerable<Token> tokens)
{
    var lexeme = current.ToString();
    char? nextChar = NextChar(content, context, current);

    if (TokenTypes.SmallerEqual.Pattern.IsMatch(lexeme + nextChar))
        return (tokens.Append(Tokens.SmallerEqual), NextChar(content, context, current));

    return (tokens.Append(Tokens.Smaller), nextChar);
}
```

Análise - Matching

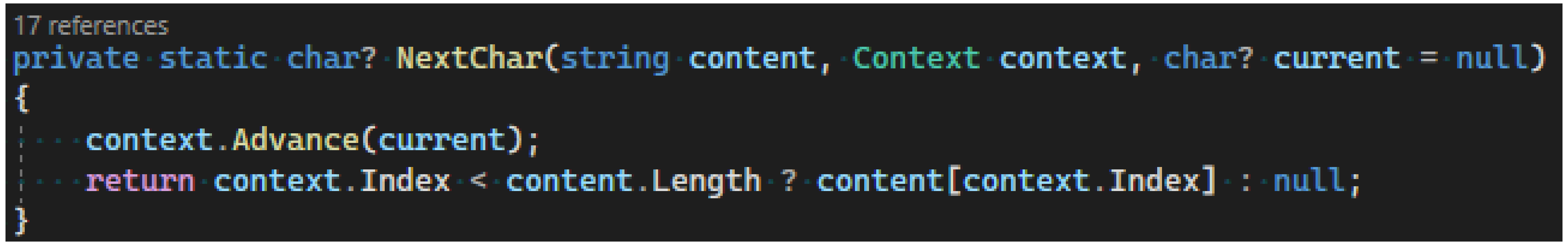
- Match composto complexo:
  - Id ou Palavra-chave
  - Inteiro ou Decimal
  - String



```
1 reference
private static (IEnumerable<Token>, char?) ResolveIdOrKeyword(string content, Context context, char current, IEnumerable<Token> tokens)
{
    var lexeme = current.ToString();
    char? nextChar;

    while (true)
    {
        nextChar = NextChar(content, context, current);

        if (!nextChar.HasValue || !TokenTypes.Id.Pattern.IsMatch(lexeme + nextChar))
            break;

        lexeme += nextChar;
    };

    var tokenType = TokenTypes.Keyword.Pattern.IsMatch(lexeme)
        ? TokenTypes.Keyword
        : TokenTypes.Id;

    return (tokens.Append(new(tokenType, lexeme)), nextChar);
}

1 reference
private static (IEnumerable<Token> tokens, char? current) ResolveIntOrDecimal(string content, Context context, char current, IEnumerable<Token> tokens)
{
    var startContext = context.Copy();

    char? nextChar = NextChar(content, context, current);
    var lexeme = current.ToString();
    var isDecimal = false;

    while (nextChar.HasValue && (Char.IsNumber(nextChar.Value) || nextChar.ToString() == NumberConstants.DecimalSeparator))
    {
        if (nextChar.ToString() == NumberConstants.DecimalSeparator)
            isDecimal = true;

        lexeme += nextChar;
        nextChar = NextChar(content, context, current);
    }

    if (isDecimal && !TokenTypes.Decimal.Pattern.IsMatch(lexeme))
        throw new LexerError(startContext, context, ErrorMessages.TOO_MANY_DECIMAL_SEPARATORS);

    var tokenType = isDecimal
        ? TokenTypes.Decimal
        : TokenTypes.Int;

    return (tokens.Append(new(tokenType, lexeme)), nextChar);
}

1 reference
private static (IEnumerable<Token> tokens, char? current) ResolveString(string content, Context context, char current, IEnumerable<Token> tokens)
{
    var startContext = context.Copy();

    var lexeme = current.ToString();
    char? nextChar = NextChar(content, context, current);

    while (nextChar.ToString() != StringConstants.StringQuote || lexeme[^1].ToString() == StringConstants.StringEscape)
    {
        lexeme += nextChar;
        nextChar = NextChar(content, context, current);

        if (!nextChar.HasValue)
            throw new LexerError(startContext, context, ErrorMessages.UNTERMINATED_STRING);
    };

    lexeme += nextChar;
    if (!TokenTypes.String.Pattern.IsMatch(lexeme))
        throw new LexerError(startContext, context, ErrorMessages.UNTERMINATED_STRING);

    return (tokens.Append(new(TokenTypes.String, lexeme)), NextChar(content, context, current));
}
```

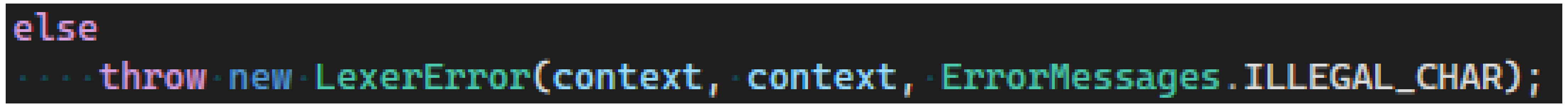
Serviço de Lexer

- Avanço de carácter

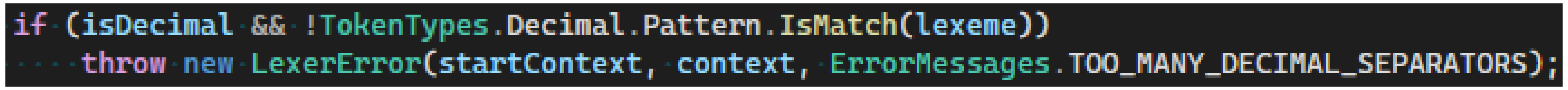


```
17 references
private static char? NextChar(string content, Context context, char? current = null)
{
    context.Advance(current);
    return context.Index < content.Length ? content[context.Index] : null;
}
```

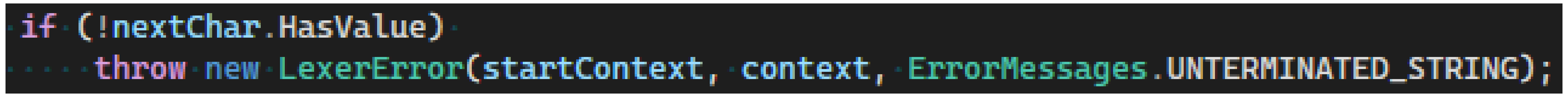
- Tratamento de erros
  - Try-Catch na análise
  - Throw na resolução



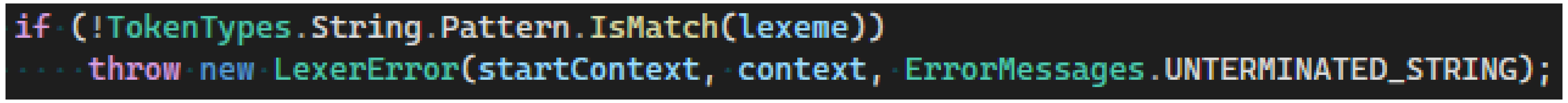
```
else
    throw new LexerError(context, context, ErrorMessages.ILLEGAL_CHAR);
```



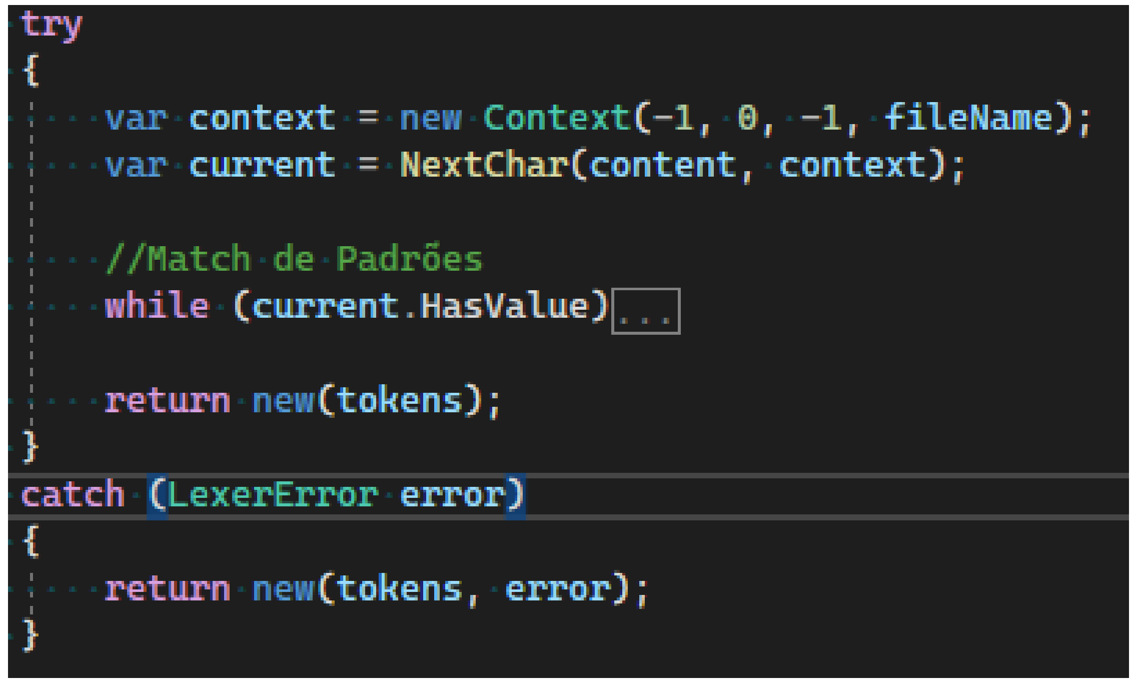
```
if (isDecimal && !TokenTypes.Decimal.Pattern.IsMatch(lexeme))
    throw new LexerError(startContext, context, ErrorMessages.TOO_MANY_DECIMAL_SEPARATORS);
```



```
if (!nextChar.HasValue)
    throw new LexerError(startContext, context, ErrorMessages.UNTERMINATED_STRING);
```



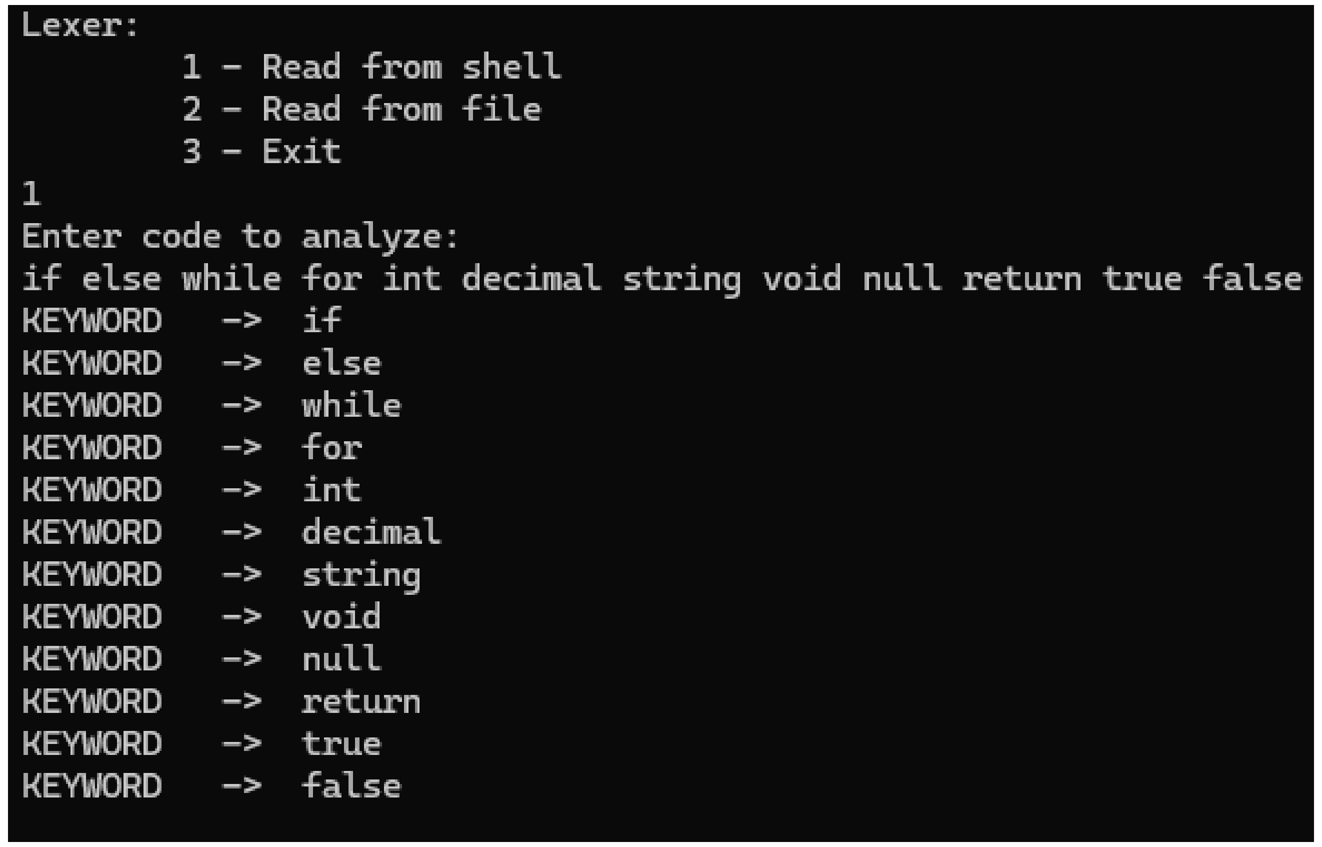
```
if (!TokenTypes.String.Pattern.IsMatch(lexeme))
    throw new LexerError(startContext, context, ErrorMessages.UNTERMINATED_STRING);
```



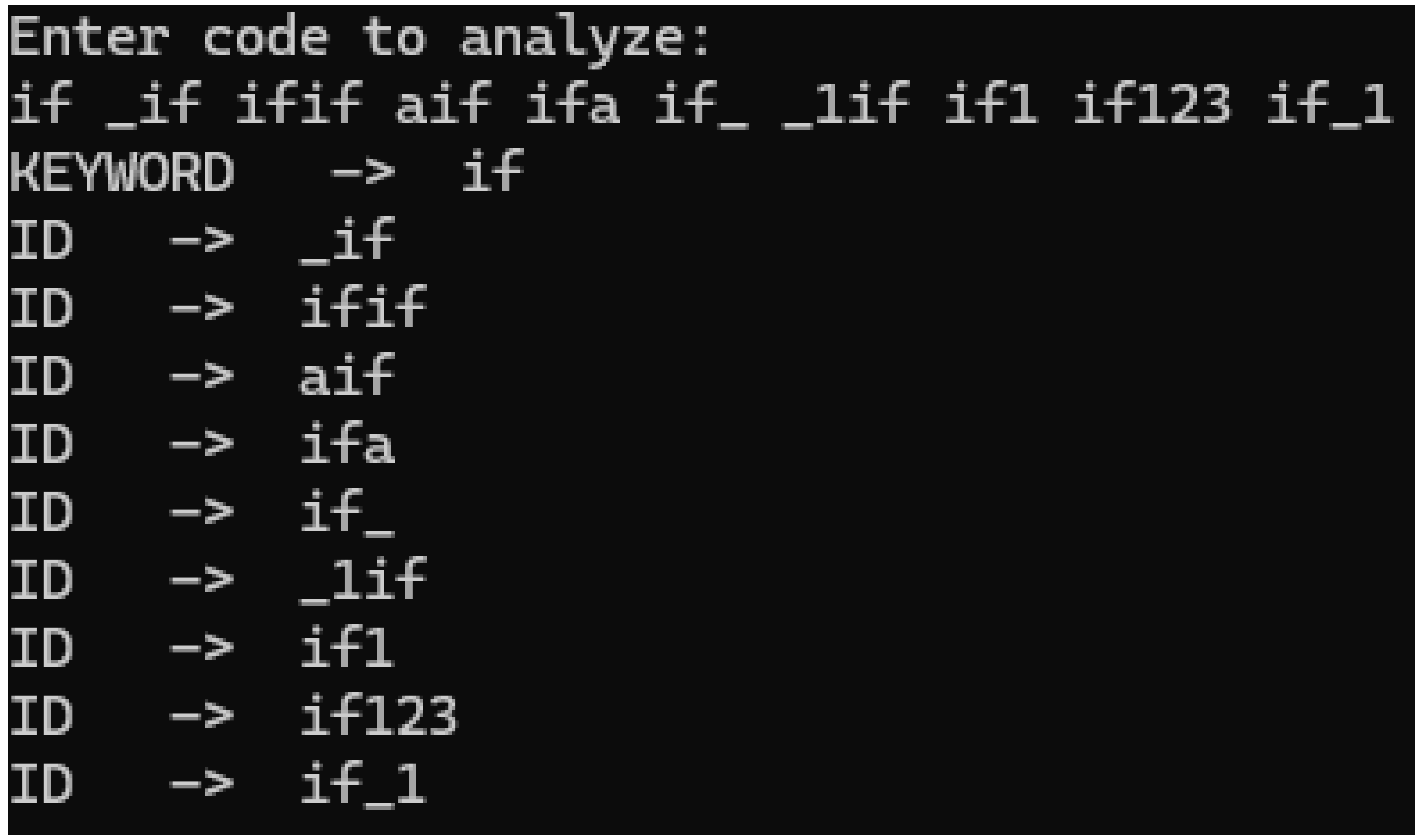
```
try
{
    var context = new Context(-1, 0, -1, fileName);
    var current = NextChar(content, context);

    //Match de Padrões
    while (current.HasValue)...

    return new(tokens);
}
catch (LexerError error)
{
    return new(tokens, error);
}
```

Testes

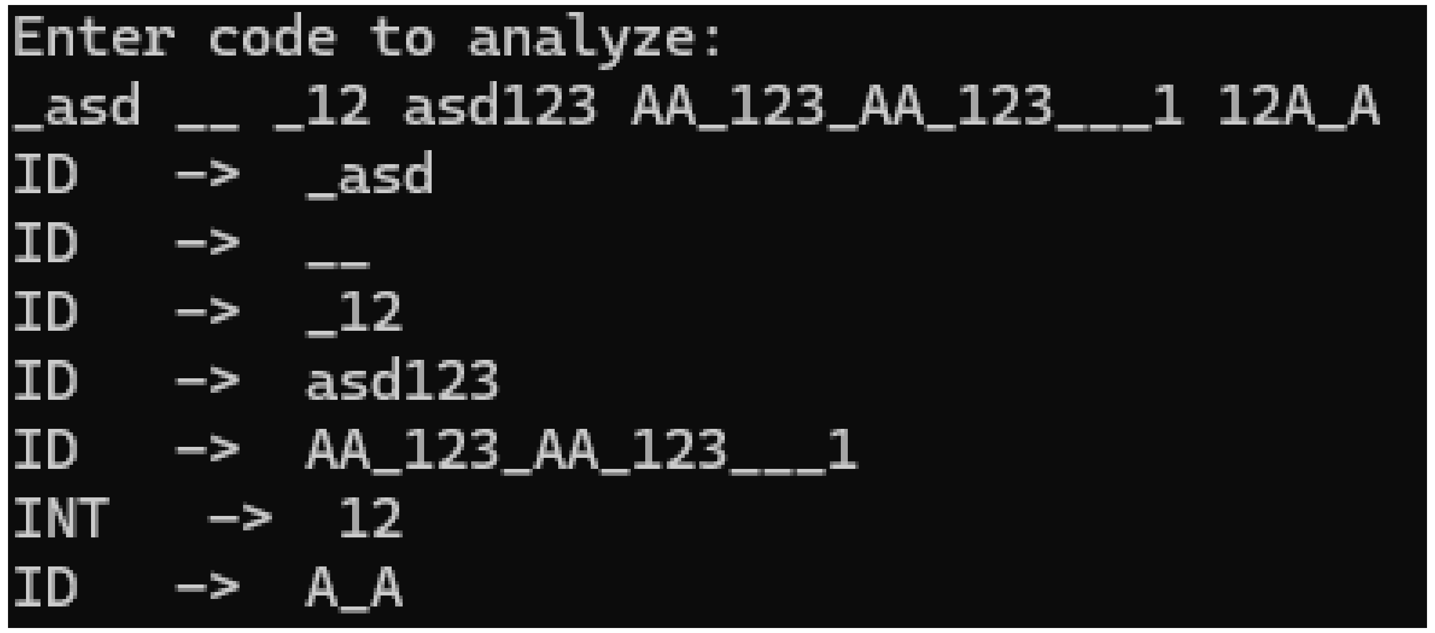
Identificadores e Palavras-Chave



```
Lexer:
        1 - Read from shell
        2 - Read from file
        3 - Exit
1
Enter code to analyze:
if else while for int decimal string void null return true false
KEYWORD   ->  if
KEYWORD   ->  else
KEYWORD   ->  while
KEYWORD   ->  for
KEYWORD   ->  int
KEYWORD   ->  decimal
KEYWORD   ->  string
KEYWORD   ->  void
KEYWORD   ->  null
KEYWORD   ->  return
KEYWORD   ->  true
KEYWORD   ->  false
```

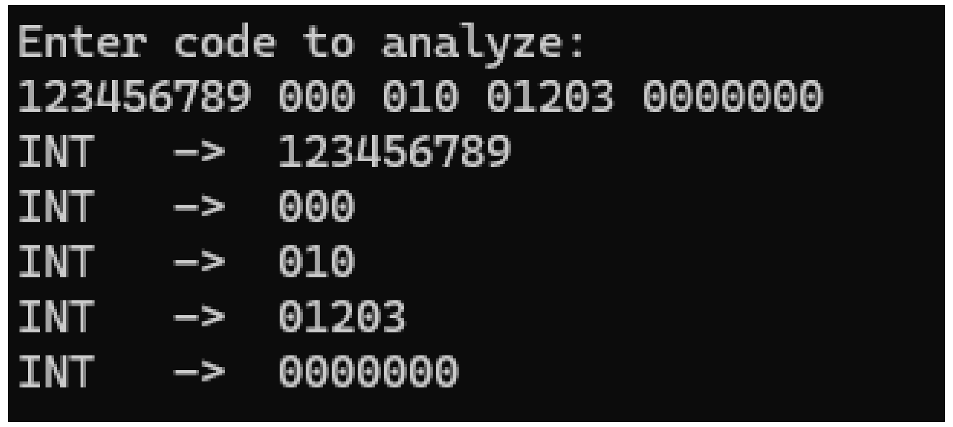


```
Enter code to analyze:
if _if ifif aif ifa if_ _1if if1 if123 if_1
KEYWORD   ->  if
ID   ->  _if
ID   ->  ifif
ID   ->  aif
ID   ->  ifa
ID   ->  if_
ID   ->  _1if
ID   ->  if1
ID   ->  if123
ID   ->  if_1
```

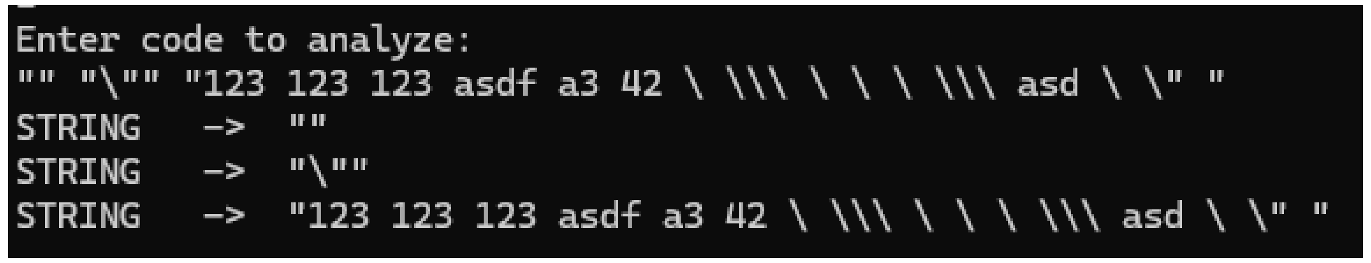


```
Enter code to analyze:
_asd __ _12 asd123 AA_123_AA_123___1 12A_A
ID   ->  _asd
ID   ->  __
ID   ->  _12
ID   ->  asd123
ID   ->  AA_123_AA_123___1
INT   ->  12
ID   ->  A_A
```

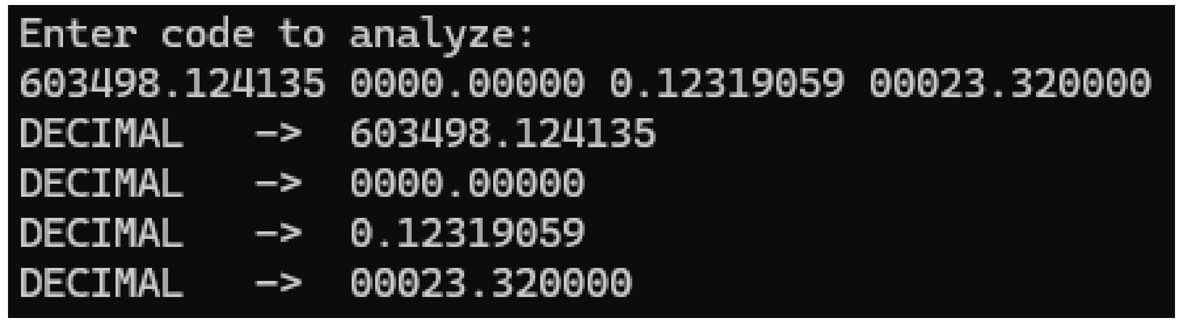
Strings, Inteiros e Decimais



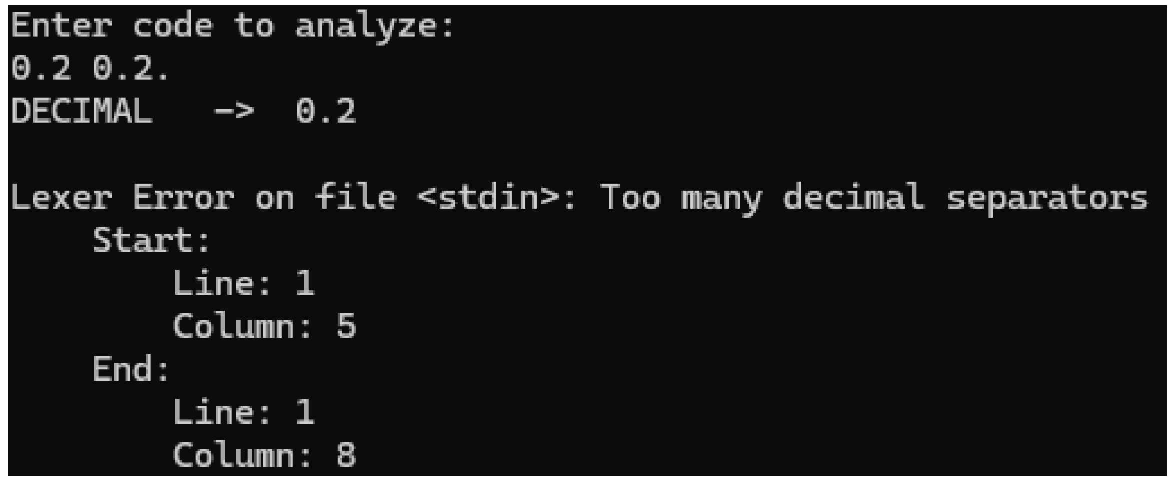
```
Enter code to analyze:
123456789 000 010 01203 0000000
INT    ->  123456789
INT    ->  000
INT    ->  010
INT    ->  01203
INT    ->  0000000
```



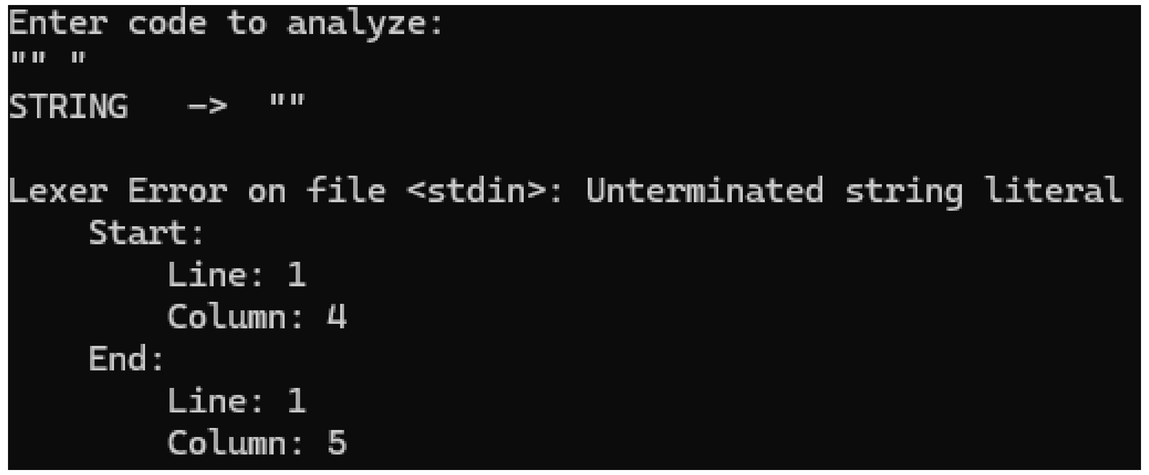
```
Enter code to analyze:
"" "\"" "123 123 123 asdf a3 42 \ \\\ \ \ \ \\\ asd \ \" "
STRING   ->  ""
STRING   ->  "\""
STRING   ->  "123 123 123 asdf a3 42 \ \\\ \ \ \ \\\ asd \ \" "
```



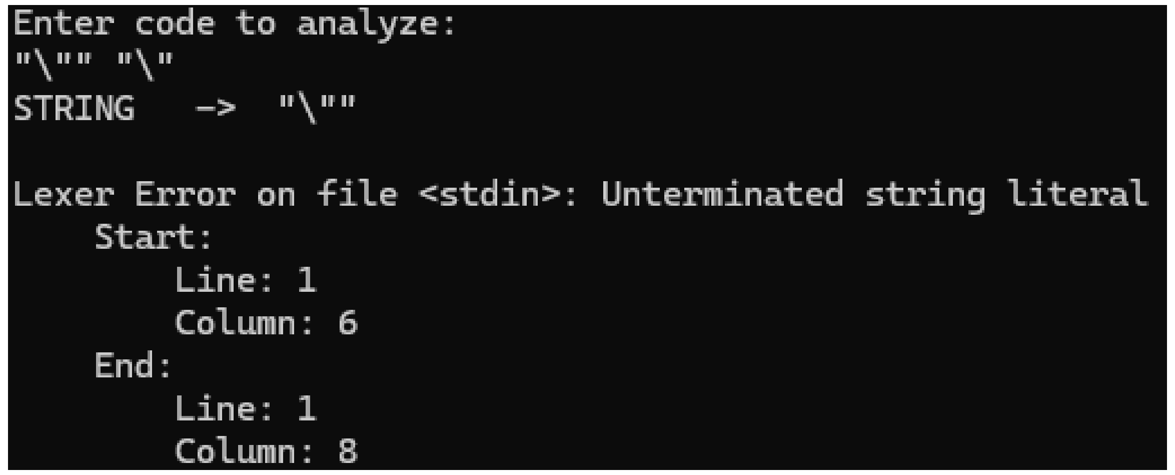
```
Enter code to analyze:
603498.124135 0000.00000 0.12319059 00023.320000
DECIMAL    ->  603498.124135
DECIMAL    ->  0000.00000
DECIMAL    ->  0.12319059
DECIMAL    ->  00023.320000
```



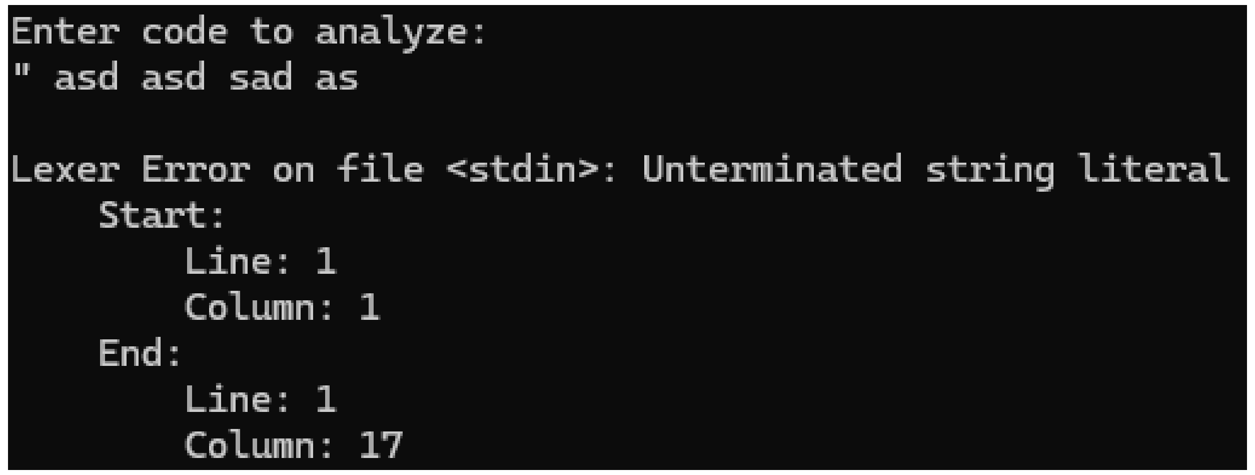
```
Enter code to analyze:
0.2 0.2.
DECIMAL    ->  0.2

Lexer Error on file <stdin>: Too many decimal separators
    Start:
        Line: 1
        Column: 5
    End:
        Line: 1
        Column: 8
```



```
Enter code to analyze:
"" "
STRING   ->  ""

Lexer Error on file <stdin>: Unterminated string literal
    Start:
        Line: 1
        Column: 4
    End:
        Line: 1
        Column: 5
```

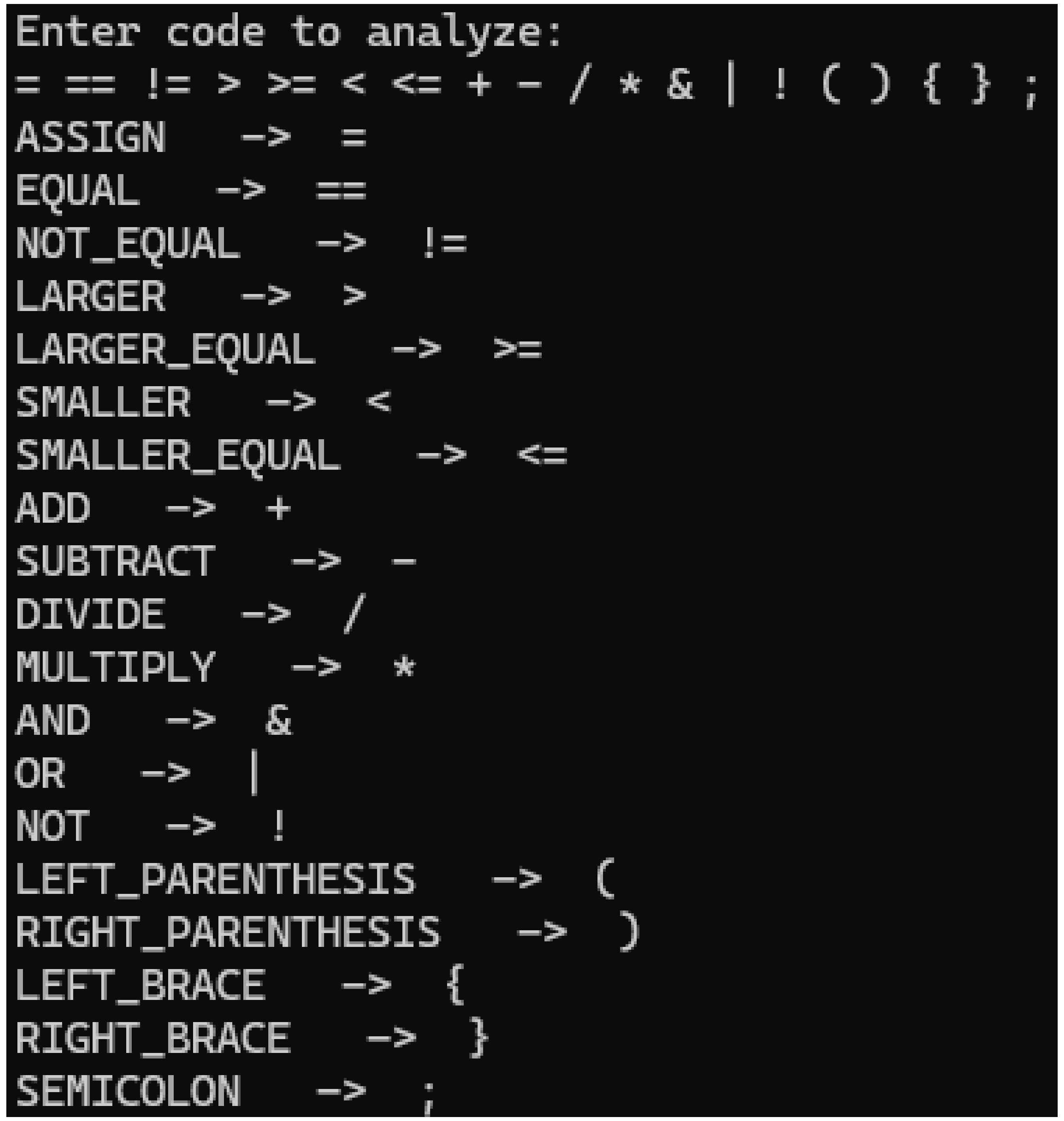


```
Enter code to analyze:
"\"" "\"
STRING   ->  "\""

Lexer Error on file <stdin>: Unterminated string literal
    Start:
        Line: 1
        Column: 6
    End:
        Line: 1
        Column: 8
```

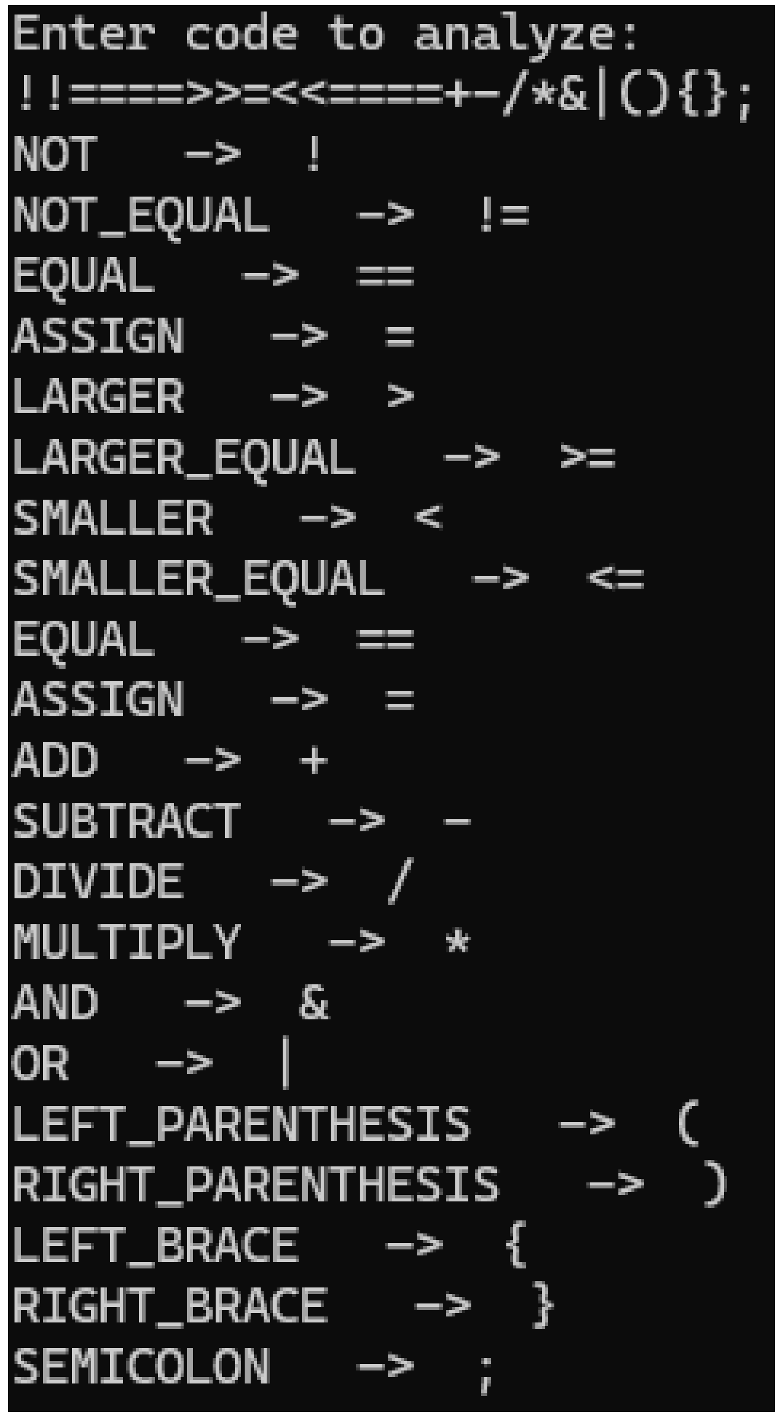


```
Enter code to analyze:
" asd asd sad as

Lexer Error on file <stdin>: Unterminated string literal
    Start:
        Line: 1
        Column: 1
    End:
        Line: 1
        Column: 17
```

Operadores e Delimitadores

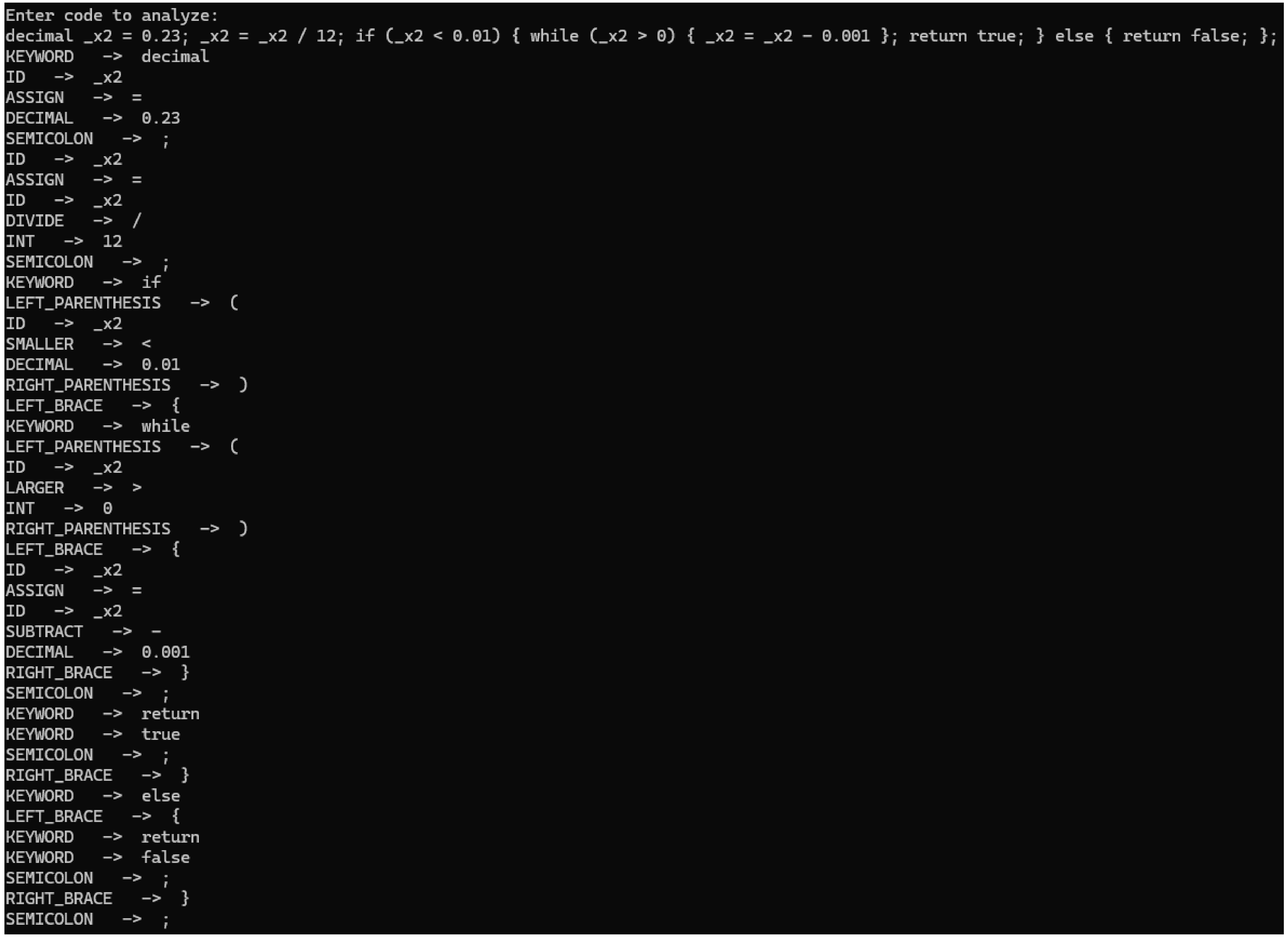


```
Enter code to analyze:
= == != > >= < <= + - / * & | ! ( ) { } ;
ASSIGN   ->  =
EQUAL   ->  ==
NOT_EQUAL   ->  !=
LARGER   ->  >
LARGER_EQUAL   ->  >=
SMALLER   ->  <
SMALLER_EQUAL   ->  <=
ADD   ->  +
SUBTRACT   ->  -
DIVIDE   ->  /
MULTIPLY   ->  *
AND   ->  &
OR   ->  |
NOT   ->  !
LEFT_PARENTHESIS   ->  (
RIGHT_PARENTHESIS   ->  )
LEFT_BRACE   ->  {
RIGHT_BRACE   ->  }
SEMICOLON   ->  ;
```



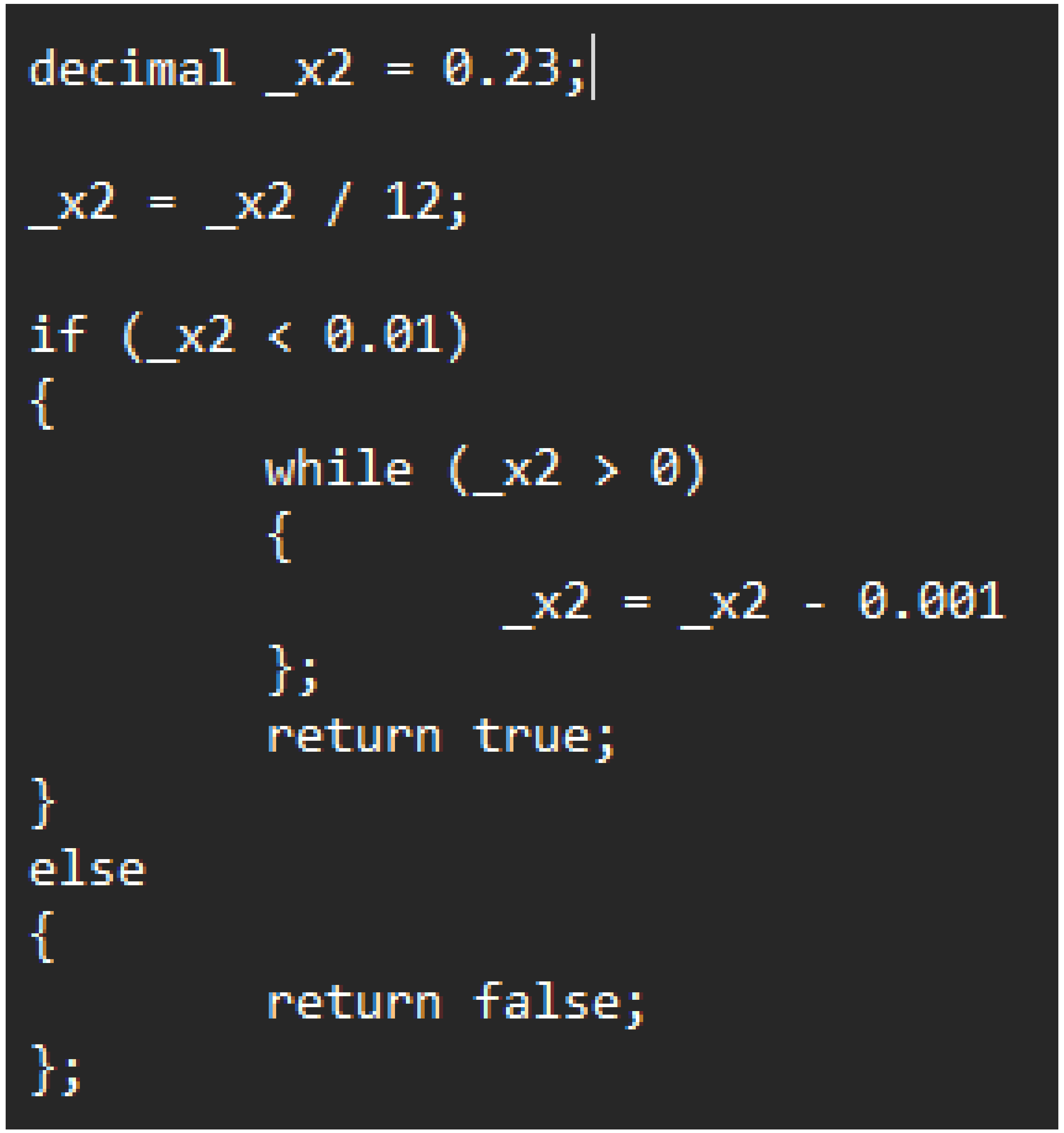
```
Enter code to analyze:
!!====>>=<<====+-/*&|(){};
NOT   ->  !
NOT_EQUAL   ->  !=
EQUAL   ->  ==
ASSIGN   ->  =
LARGER   ->  >
LARGER_EQUAL   ->  >=
SMALLER   ->  <
SMALLER_EQUAL   ->  <=
EQUAL   ->  ==
ASSIGN   ->  =
ADD   ->  +
SUBTRACT   ->  -
DIVIDE   ->  /
MULTIPLY   ->  *
AND   ->  &
OR   ->  |
LEFT_PARENTHESIS   ->  (
RIGHT_PARENTHESIS   ->  )
LEFT_BRACE   ->  {
RIGHT_BRACE   ->  }
SEMICOLON   ->  ;
```

Geral

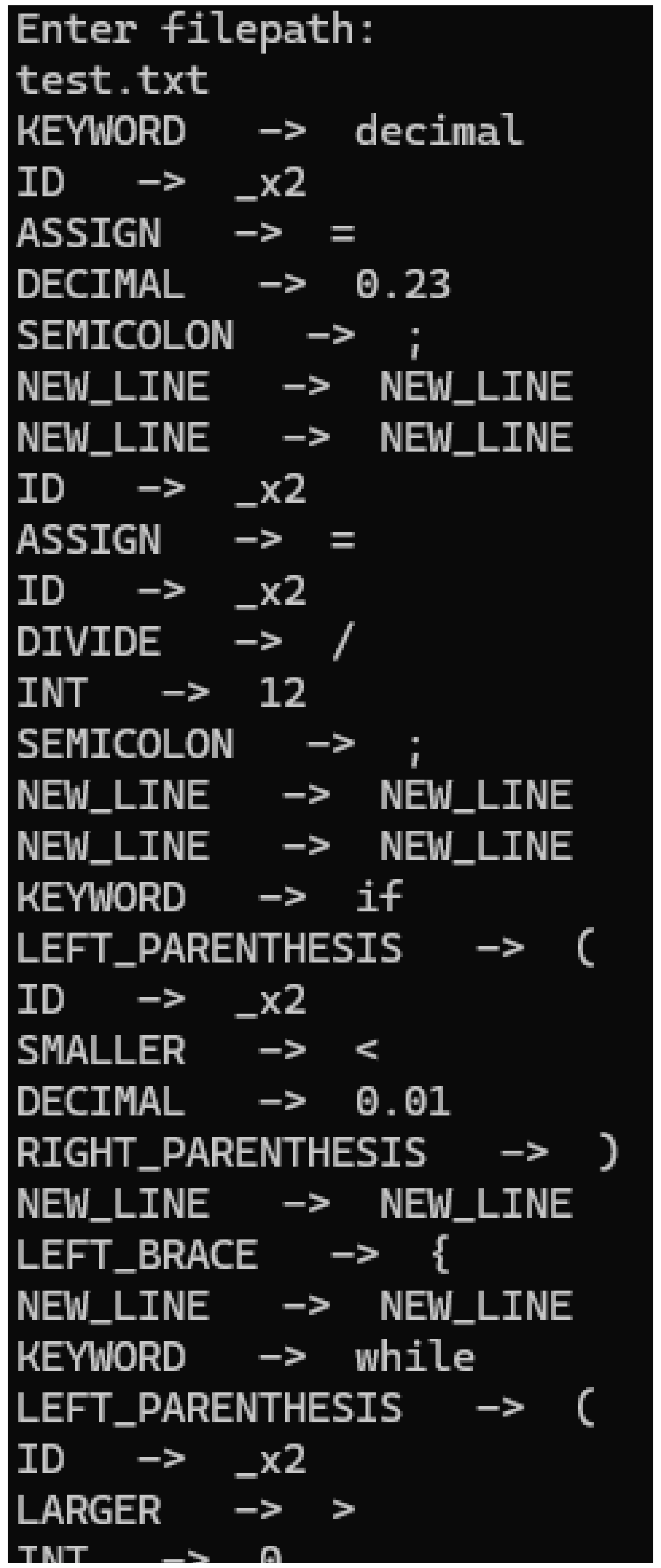


```
Enter code to analyze:
decimal _x2 = 0.23; _x2 = _x2 / 12; if (_x2 < 0.01) { while (_x2 > 0) { _x2 = _x2 - 0.001 }; return true; } else { return false; };
KEYWORD   ->  decimal
ID   ->  _x2
ASSIGN   ->  =
DECIMAL   ->  0.23
SEMICOLON   ->  ;
ID   ->  _x2
ASSIGN   ->  =
ID   ->  _x2
DIVIDE   ->  /
INT   ->  12
SEMICOLON   ->  ;
KEYWORD   ->  if
LEFT_PARENTHESIS   ->  (
ID   ->  _x2
SMALLER   ->  <
DECIMAL   ->  0.01
RIGHT_PARENTHESIS   ->  )
LEFT_BRACE   ->  {
KEYWORD   ->  while
LEFT_PARENTHESIS   ->  (
ID   ->  _x2
LARGER   ->  >
INT   ->  0
RIGHT_PARENTHESIS   ->  )
LEFT_BRACE   ->  {
ID   ->  _x2
ASSIGN   ->  =
ID   ->  _x2
SUBTRACT   ->  -
DECIMAL   ->  0.001
RIGHT_BRACE   ->  }
SEMICOLON   ->  ;
KEYWORD   ->  return
KEYWORD   ->  true
SEMICOLON   ->  ;
RIGHT_BRACE   ->  }
KEYWORD   ->  else
LEFT_BRACE   ->  {
KEYWORD   ->  return
KEYWORD   ->  false
SEMICOLON   ->  ;
RIGHT_BRACE   ->  }
SEMICOLON   ->  ;
```

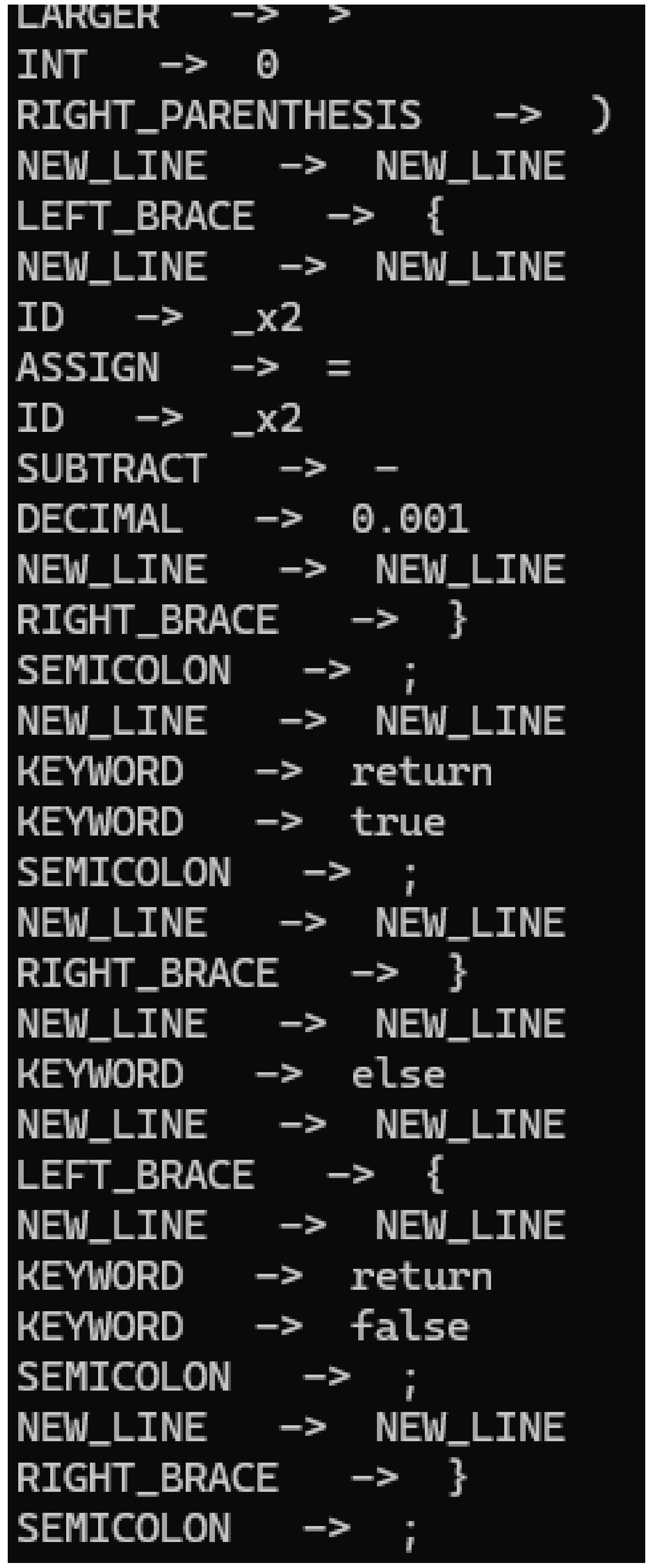
Arquivo



```
decimal _x2 = 0.23;

_x2 = _x2 / 12;

if (_x2 < 0.01)
{
        while (_x2 > 0)
        {
                _x2 = _x2 - 0.001
        };
        return true;
}
else
{
        return false;
};
```



```
Enter filepath:
test.txt
KEYWORD   ->  decimal
ID   ->  _x2
ASSIGN   ->  =
DECIMAL   ->  0.23
SEMICOLON   ->  ;
NEW_LINE   ->  NEW_LINE
NEW_LINE   ->  NEW_LINE
ID   ->  _x2
ASSIGN   ->  =
ID   ->  _x2
DIVIDE   ->  /
INT   ->  12
SEMICOLON   ->  ;
NEW_LINE   ->  NEW_LINE
NEW_LINE   ->  NEW_LINE
KEYWORD   ->  if
LEFT_PARENTHESIS   ->  (
ID   ->  _x2
SMALLER   ->  <
DECIMAL   ->  0.01
RIGHT_PARENTHESIS   ->  )
NEW_LINE   ->  NEW_LINE
LEFT_BRACE   ->  {
NEW_LINE   ->  NEW_LINE
KEYWORD   ->  while
LEFT_PARENTHESIS   ->  (
ID   ->  _x2
LARGER   ->  >
INT   ->  0
```



```
LARGER   ->  >
INT   ->  0
RIGHT_PARENTHESIS   ->  )
NEW_LINE   ->  NEW_LINE
LEFT_BRACE   ->  {
NEW_LINE   ->  NEW_LINE
ID   ->  _x2
ASSIGN   ->  =
ID   ->  _x2
SUBTRACT   ->  -
DECIMAL   ->  0.001
NEW_LINE   ->  NEW_LINE
RIGHT_BRACE   ->  }
SEMICOLON   ->  ;
NEW_LINE   ->  NEW_LINE
KEYWORD   ->  return
KEYWORD   ->  true
SEMICOLON   ->  ;
NEW_LINE   ->  NEW_LINE
RIGHT_BRACE   ->  }
NEW_LINE   ->  NEW_LINE
KEYWORD   ->  else
NEW_LINE   ->  NEW_LINE
LEFT_BRACE   ->  {
NEW_LINE   ->  NEW_LINE
KEYWORD   ->  return
KEYWORD   ->  false
SEMICOLON   ->  ;
NEW_LINE   ->  NEW_LINE
RIGHT_BRACE   ->  }
SEMICOLON   ->  ;
```