Ministerul Educaţiei și Cercetării al Republicii Moldova Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Laboratory work 3:

Lexer & Scanner.

Elaborated:

st. gr. FAF-223 Ciornii Alexandr

Verified:

asist. univ. Dumitru Cretu

Chişinău - 2024

Overview

The term lexer comes from lexical analysis which, in turn, represents the process of extracting lexical tokens from a string of characters. There are several alternative names for the mechanism called lexer, for example tokenizer or scanner. The lexical analysis is one of the first stages used in a compiler/interpreter when dealing with programming, markup or other types of languages. The tokens are identified based on some rules of the language and the products that the lexer gives are called lexemes. So basically the lexer is a stream of lexemes. Now in case it is not clear what's the difference between lexemes and tokens, there is a big one. The lexeme is just the byproduct of splitting based on delimiters, for example spaces, but the tokens give names or categories to each lexeme. So the tokens don't retain necessarily the actual value of the lexeme, but rather the type of it and maybe some metadata.

Objectives:

1.Understand what lexical analysis is.

2.Get familiar with the inner workings of a lexer/scanner/tokenizer.

3.Implement a sample lexer and show how it works.

Implementation description

I have public enum TokenType that determine literally type of the token, also I have class Token, Lexer creates objects of it. Class Lexer takes string as input and turns that string in List of Tokens. Method SkipWhiteSpaces allows WS in input string. Class Calculator calculates value of the expression. Main method takes input string (it's easy to turn it to Console.ReadLine, but I don't see a point for this) than there are instantiated variable of Lexer type and List of tokens, then foreach loop to write every piece of information about tokens and finally is calculated value of the expression and it is printed in console

```
static class Program
{
    static void Main(string[] args)
    {
        string input = "(10 + 5) * 2 - 6 / 3";
        Lexer lexer = new Lexer(input);
        List<Token> tokens = lexer.Tokenize();

        foreach (var token in tokens)
        {
            Console.WriteLine($"Type: {token.Type}, Value: {token.Value}");
        }

        Calculator calculator = new Calculator(tokens);
        double result = calculator.Calculate();
        Console.WriteLine($"Result: {result}");
    }
}
```

Here’s exact output for that input:

```
Type: LeftParenthesis, Value: (
Type: Number, Value: 10
Type: Plus, Value: +
Type: Number, Value: 5
Type: RightParenthesis, Value: )
Type: Multiply, Value: *
Type: Number, Value: 2
Type: Minus, Value: -
Type: Number, Value: 6
Type: Divide, Value: /
Type: Number, Value: 3
Type: EOF, Value:
Result: 28
```

Conclusions and Results

In this laboratory work, we delved into the fundamental concepts of lexing and scanning, essential components of the early stages of compiler construction. As software engineering students, this practical exercise provided us with valuable insights into the process of transforming raw input text into a structured sequence of tokens, which can be further processed by a parser.

Through the implementation of a lexer and scanner in C#, we gained a deeper understanding of how lexical analysis plays a crucial role in breaking down complex input expressions into manageable units, such as numbers, operators, and parentheses. We learned about the importance of tokenization in facilitating subsequent stages of compilation, such as parsing and semantic analysis.

Furthermore, we encountered challenges such as handling different token types, managing whitespace, and ensuring error handling for invalid input expressions. These challenges allowed us to sharpen our problem-solving skills and develop a robust understanding of error detection and recovery strategies in lexing and scanning.

Overall, this laboratory work equipped us with practical knowledge and hands-on experience in building essential components of a compiler infrastructure. As aspiring software engineers, understanding lexing and scanning lays a solid foundation for tackling more advanced topics in compiler construction and language processing.