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

Introduction: In modern software engineering, understanding the intricacies of compiler construction is paramount. A fundamental aspect of this process involves lexical analysis, often referred to as lexing or scanning. Lexing serves as the initial step in parsing source code, transforming raw text into a structured sequence of tokens. These tokens lay the groundwork for subsequent stages of compilation, making them essential components of any compiler or interpreter.

Objective: The primary objective of this laboratory work was to delve deep into the principles of lexing and scanning. By implementing a lexer and scanner in C#, we aimed to gain practical experience in tokenization, whitespace handling, and error detection mechanisms. Our goal was to understand how lexing and scanning contribute to the overall compilation process and develop insights that would aid us in future software engineering endeavors.

Implementation: The implementation of the lexer and scanner involved careful consideration of various factors. We utilized object-oriented programming principles to create modular and extensible components. The lexer was responsible for breaking down the input text into individual tokens, recognizing numbers, operators, and parentheses while gracefully handling whitespace. Error handling mechanisms were integrated to detect and report any invalid input expressions, providing informative messages for debugging purposes.

Building upon the lexer, the scanner further refined the token stream, categorizing tokens into distinct types. This refinement process set the stage for subsequent compilation stages, such as parsing and semantic analysis. Through iterative testing and refinement, we ensured the robustness and efficiency of our lexer and scanner implementations.

Calculator Functionality: In addition to the lexer and scanner, we implemented a simple calculator using the token stream generated by the lexer. The calculator parses the token stream and evaluates the arithmetic expression represented by the tokens. It utilizes a recursive descent parsing technique to handle expressions, terms, and factors, following the operator precedence rules. By recursively evaluating sub-expressions, the calculator computes the final result of the expression.

Challenges Faced: The implementation process presented several challenges that deepened our understanding of lexing and scanning. Managing different token types and ensuring accurate tokenization required meticulous attention to detail. Handling whitespace effectively while maintaining code readability posed additional challenges. Implementing robust error handling mechanisms was crucial for detecting and reporting errors, contributing to the overall reliability of our lexer and scanner implementations.

Insights Gained: Through this laboratory work, we gained valuable insights into the intricate workings of lexing and scanning. We learned how these processes serve as the foundation of compiler construction, enabling the transformation of source code into a form that can be parsed and analyzed further. Understanding lexing and scanning principles is essential for aspiring software engineers, providing a solid foundation for tackling complex problems in software development.

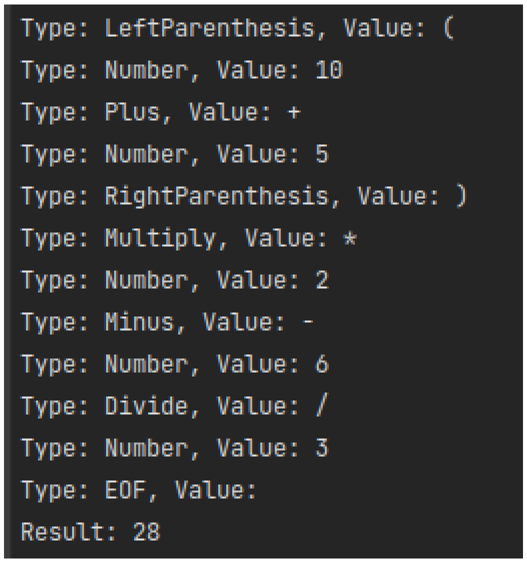
Conclusion: In conclusion, our laboratory work on lexing and scanning provided us with invaluable hands-on experience in essential components of compiler construction. By implementing a lexer and scanner in C#, we deepened our understanding of tokenization, whitespace handling, and error detection mechanisms. This practical exercise equipped us with foundational knowledge and problem-solving skills necessary for tackling real-world challenges in software development. Moving forward, we are confident that the insights gained from this experience will serve us well in our future endeavors as software engineering professionals.

```csharp
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
```