Ministerul Educaţiei și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică

Laboratory work nr. 3
Course: Formal languages and finite automata
Topic: Lexer & Scanner

Elaborated:
st. gr. FAF-221 Revenco Victor

Verified:
asist. univ. Cretu Dumitru

Chişinău - 2024

Theory

In programming, lexers and scanners are tools used in the early stages of compiling or interpreting code. They are responsible for breaking down the source code into smaller units called tokens, which are the building blocks of the language. These tokens can be keywords, identifiers, literals (like numbers or strings), or symbols (such as punctuation marks).

A lexer or scanner works by reading the source code character by character and grouping them into tokens based on predefined rules. These rules are usually expressed as regular expressions that define the patterns of valid tokens. For example, a rule might specify that an identifier token must start with a letter followed by zero or more letters or digits.

Once the lexer identifies a token, it categorizes it into its corresponding type and may also provide additional information, such as the token's value or position in the source code. This tokenization process creates a structured representation of the code that can be easily processed by the compiler or interpreter.

In summary, lexers and scanners are essential components of programming language processing, as they help in understanding and analyzing the source code. They play a crucial role in breaking down complex code into simpler tokens, making it easier for compilers and interpreters to work with the code.

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.

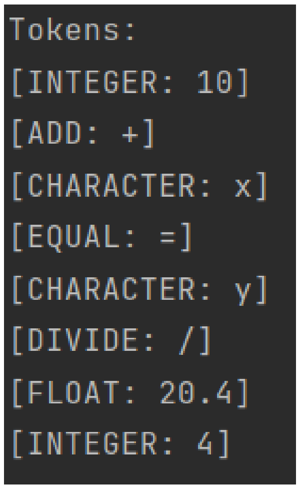
Implementation Description

```java
public static List<Token> tokenize(String input) {
    List<Token> tokens = new ArrayList<>();
    StringBuilder currentToken = new StringBuilder();

    for (char c : input.toCharArray()) {
        if (Character.isDigit(c)) {
            currentToken.append(c);
        } else if (c == '.') {
            currentToken.append(c);
        } else if (Character.isLetter(c)) {
            if (currentToken.length() > 0) {
```

```
tokens.add(createToken(currentToken.toString()));
                currentToken.setLength(0);
            }
            tokens.add(createToken(String.valueOf(c)));
        } else if (c == '+' || c == '-' || c == '*' || c == '/'
|| c == '=') {
            if (currentToken.length() > 0) {

tokens.add(createToken(currentToken.toString()));
                currentToken.setLength(0);
            }
            tokens.add(createToken(String.valueOf(c)));
        } else if (c == ' ') {
            if (currentToken.length() > 0) {

tokens.add(createToken(currentToken.toString()));
                currentToken.setLength(0);
            }
        }
    }

    if (currentToken.length() > 0) {
        tokens.add(createToken(currentToken.toString()));
    }

    return tokens;
}
```

This is the function that will tokenize our string. First of all it initializes an empty list of tokens which will be returned at the end and a string builder "currentToken" which will be added to the list of tokens whenever a token is identified. I used a for loop to go through each character in the input by first making the string into an array of characters. If the character is a digit or a letter we add it to the currentToken. If at any point in the loop we find a space (' ') or an operator ('+', '-', etc.) we check if the currentToken has any elements in it. If it does, then we add what we have to the list of tokens, after which we reset the string builder. On top of that we add the operator if there is one. At the end of the loop we check if the string builder has any elements which need to be added to the tokens list, and we return the list.

Screenshots



```
Tokens:
[INTEGER: 10]
[ADD: +]
[CHARACTER: x]
[EQUAL: =]
[CHARACTER: y]
[DIVIDE: /]
[FLOAT: 20.4]
[INTEGER: 4]
```

Conclusions

In conclusion, the lab on Lexer and Scanner provided a practical introduction to the core concepts of lexical analysis in programming. By implementing a simple lexer in Java, we were able to understand how source code can be broken down into tokens.

This implementation highlights the basic principles of lexical analysis, where characters are processed sequentially to identify tokens based on predefined rules. While this lexer is limited in functionality and may not cover all possible cases in a real programming language, it serves as a starting point for understanding how lexers can be implemented to tokenize source code.

Overall, lexers play a crucial role in the compilation or interpretation process of programming languages, as they provide a structured representation of the source code that can be further processed by compilers or interpreters.