**Ministerul Educaţiei și Cercetării al Republicii Moldova Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

Laboratory work nr. 2

Course: Formal languages and finite automata

Topic: Lexer & Scanner

Elaborated:

st. gr. FAF-221 Cusnir Grigorii

Verified:

asist. univ. Cretu Dumitru

Chişinău - 2024

**Theory:**

Lexical analysis is an essential phase in compiler design, where the input source code is scanned and converted into a sequence of tokens for further processing by the parser. The main task of a lexer (or tokenizer) is to break down the input text into meaningful units called tokens, which are the building blocks of the programming language.

In this project, we implemented a simple lexer in Java using regular expressions to recognize and classify different tokens such as identifiers, keywords, integers, strings, operators, and other symbols. The lexer operates by sequentially scanning the input text, matching patterns, and constructing tokens based on predefined rules.

**Objectives:**

1. **Understand Lexical Analysis:** Gain practical experience in implementing a basic lexer for tokenizing source code.
2. **Apply Regular Expressions:** Utilize Java's regex capabilities to define token patterns and extract tokens from input text.
3. **Token Classification:** Implement logic to recognize and classify different types of tokens based on language grammar.
4. **Integration with Compiler Design:** Explore the initial phase of compiler design by implementing a lexer component.

**Implementation Description:**

The lexer implementation consists of two main classes:

**Token Class:** Represents a token with a type and value. This class is used to encapsulate information about individual tokens identified during lexical analysis.



**Lexer Class:** Implements the core functionality of the lexer. It initializes with the input text and provides methods to tokenize the text based on predefined rules.

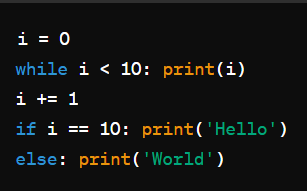
**Tokenization Process:** The lexer scans the input text character by character. It uses regular expressions to match specific patterns corresponding to different token types.

Token Types: Tokens are classified into categories such as identifiers, integers, strings, operators, keywords, and special symbols.

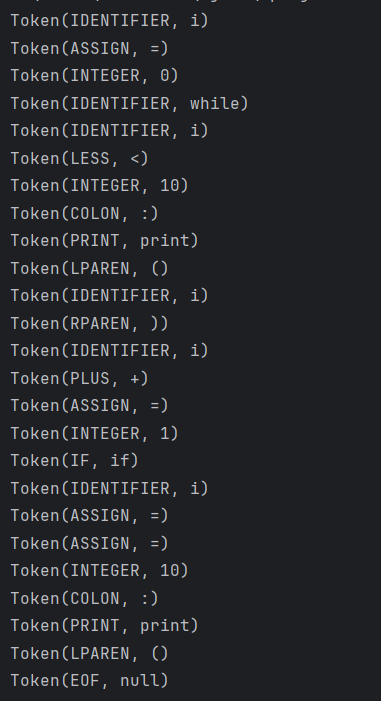
Whitespace Handling: The lexer skips over whitespace and newline characters to focus on meaningful tokens.

**Screenshots**

Below are screenshots demonstrating the execution of the lexer on sample input:



Input



Output

**Conclusion:**

In conclusion, this project provided hands-on experience in implementing a lexer using Java for basic tokenization of source code. We successfully achieved the objectives of understanding lexical analysis, applying regular expressions for pattern matching, and classifying tokens based on predefined rules.

The lexer demonstrated in this report can serve as a fundamental component in a larger compiler or interpreter project, enabling subsequent phases such as parsing and semantic analysis. Future enhancements could include extending token classification, handling more complex language features, and integrating with higher-level compiler components.

Overall, this project contributed to a deeper understanding of compiler design principles and practical implementation of language processing tools.