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

**Facultatea Calculatoare, Informatică și Microelectronică**

**Report**

Laboratory work 3:

*Formal Languages & Finite Automata.*

Elaborated:

st. gr. FAF-223 Irina Nedealcova

Verified:

asist. univ. Cretu Dumitru

Chişinău – 2024

**Introduction**

In this lab, we focused on constructing a lexer, which is a critical component in the process of lexical analysis, an essential step in parsing and compiling programming languages. The lexer, also known as a tokenizer or scanner, converts a sequence of input characters into a series of tokens, based on the syntax rules of the language.

**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:**

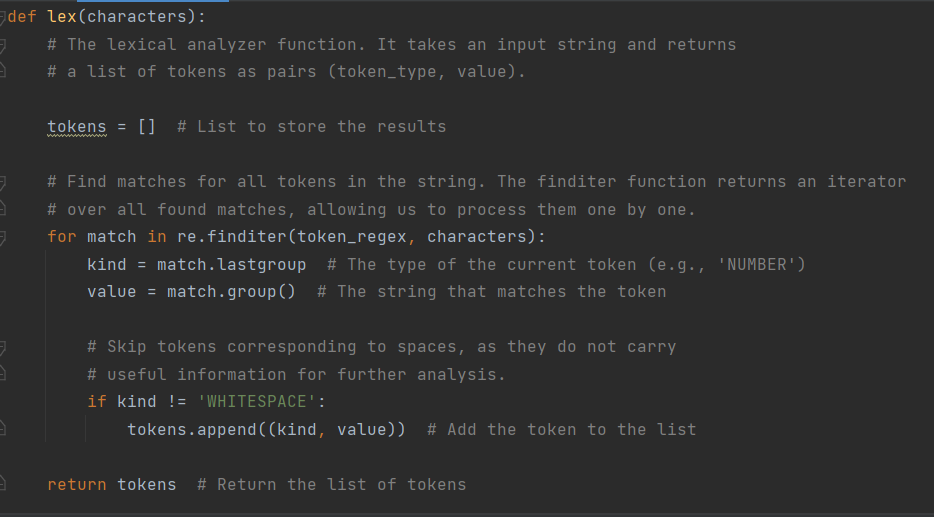
* Understand what lexical analysis [1] is.
* Get familiar with the inner workings of a lexer/scanner/tokenizer.
* Implement a sample lexer and show how it works.

**Lexical Analysis and the Inner Workings of a Lexer**

Lexical analysis is a fundamental process in the compilation and interpretation of programming languages. It involves reading the source code as a stream of characters and grouping these characters into meaningful sequences known as tokens. Tokens can represent identifiers (like variable names), literals (like numbers or string constants), symbols (like operators or punctuation), and keywords (reserved words that have special meaning in the language). The primary goal of lexical analysis, often performed by a component called a lexer or scanner, is to simplify the job of the parser, which comes next in the compilation pipeline. The parser uses the tokens produced by the lexer to build a syntactic structure (often a tree), which then undergoes further analysis and transformation by the compiler or interpreter.

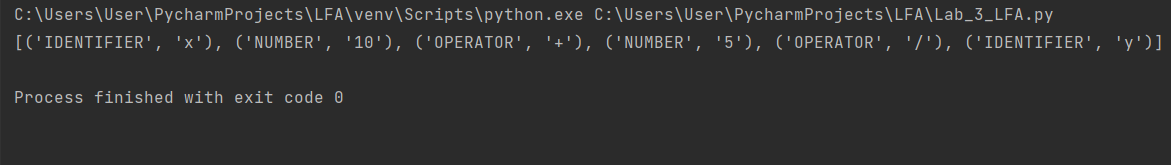
**Implementation of a Lexer in Python**

The provided implementation demonstrates a simple lexer in Python, utilizing the re module for pattern matching. The lexer is designed to identify four types of tokens: numbers, identifiers, mathematical operators, and whitespace. The lexer skips whitespace, as it is typically not meaningful for further syntactic analysis. Each token type is associated with a regular expression that describes the valid characters for that token. These expressions are combined into a single pattern where each part is named according to the corresponding token type, allowing the lexer to identify the type of each matched token.



The lex function is the core of this lexer. It takes an input string and returns a list of tokens identified in the string. It uses the re.finditer function to iterate over all matches of the compiled pattern in the input string. For each match, it determines the type of the token based on which part of the pattern matched and includes the token in the result list if it is not whitespace. This implementation is a straightforward example of how to perform lexical analysis in Python, showcasing the use of regular expressions for token identification and the overall structure of a lexer.

**Results**



**Conclusion:**

The provided code serves as a concise and effective illustration of how lexical analysis can be implemented in Python. By leveraging regular expressions, the implementation is both flexible and efficient, capable of identifying multiple token types in a single pass through the input text. This approach simplifies the parsing stage, as the parser can work with a structured sequence of tokens instead of raw text. Although the example lexer is basic, it lays the foundation for more complex lexical analyzers capable of handling a wider variety of token types and more sophisticated language features. Understanding the principles and techniques demonstrated in this implementation is crucial for anyone interested in language processing, whether for developing compilers, interpreters, or other tools that require parsing and analyzing text.