Abstract:

The interface between the source code and the latter stages of the compiler or interpreter is a lexical analyzer's abstract. It generates a stream of tokens that the parser uses as input as it continues its syntactic analysis and builds an abstract syntax tree (AST) to represent the program's structure. The output of the lexical analyzers aids in later compilation or interpretation procedures, allowing the compiler or interpreter to examine and run the program.

Introduction:

To identify and categorize tokens, the lexical analyzer acts on a character stream and applies a set of predefined rules. Usually, finite automata or regular expressions are used to define these rules. The lexical analyzer examines the text character by character as it processes the input, looking for patterns that

match to various token kinds. Characters into tokens, giving each token a certain type or group.

Objectives:

A lexical analyzer, commonly referred to as a lexical analyzers or scanner, has the following primary goals:

1. Tokenization: The input source code is divided into a series of tokens by the lexical analyzer. The smallest significant components in a programming language are called tokens, and they include words like "keywords," "identifiers," "operators," "literals," and "punctuation." Each token is identified and categorized by the lexical analyzers according to predetermined rules.

2. Elimination of Whitespace and Comments: The lexical analyzers remove any unused whitespace characters from the source code, including spaces, tabs, and newline characters.

3. Generation of Symbol Tables: The lexical analyzers are capable of producing a symbol table, which is a data structure

used to keep track of identifiers (variables, functions, and classes) that are encountered during tokenization.

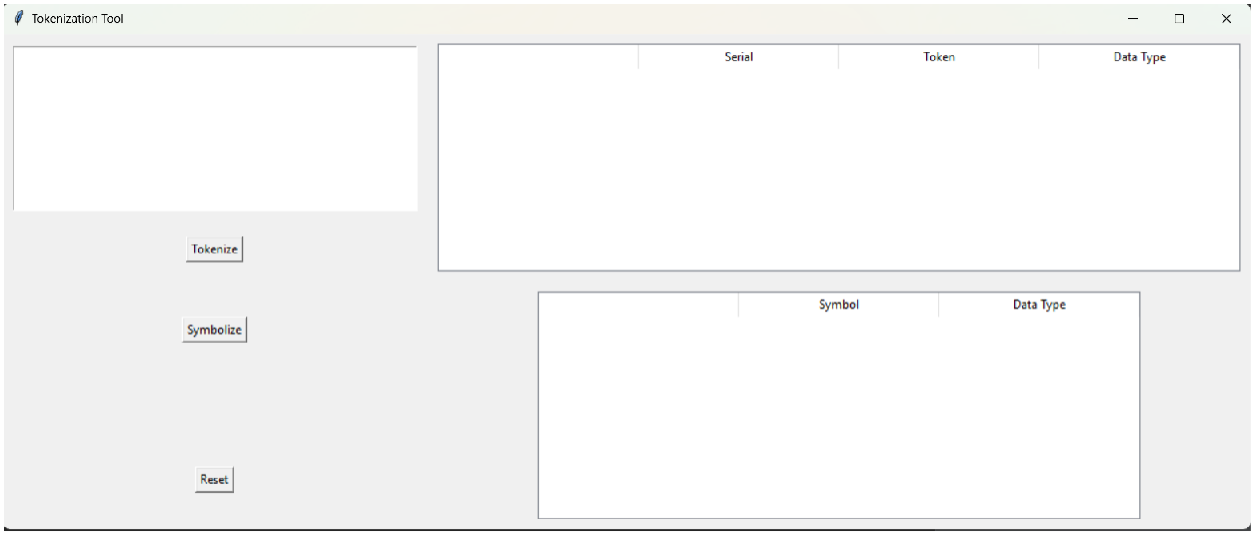
Overall, a lexical analyzer's primary goal is to convert the input source code into a stream of tokens, laying the groundwork for a compiler or interpreter's next stages.

Design and Implementation:

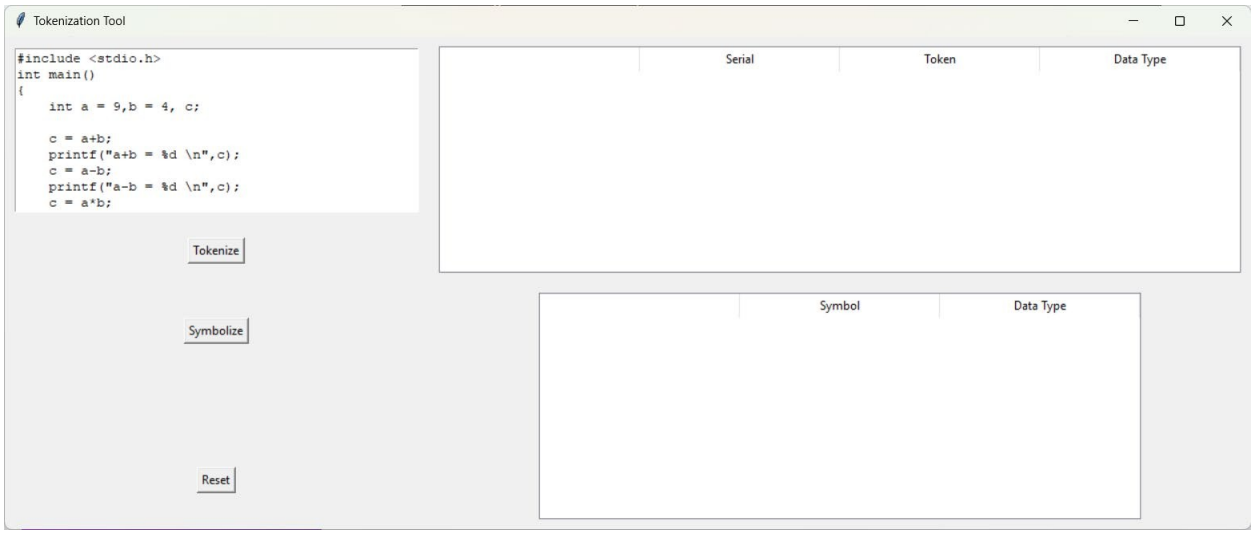
The task of tokenizing input source code, the smallest meaningful units of a programming language, is performed by a lexical analyzer, also referred to as a scanner.

An overview of the design and execution process is provided below:

Project interface:

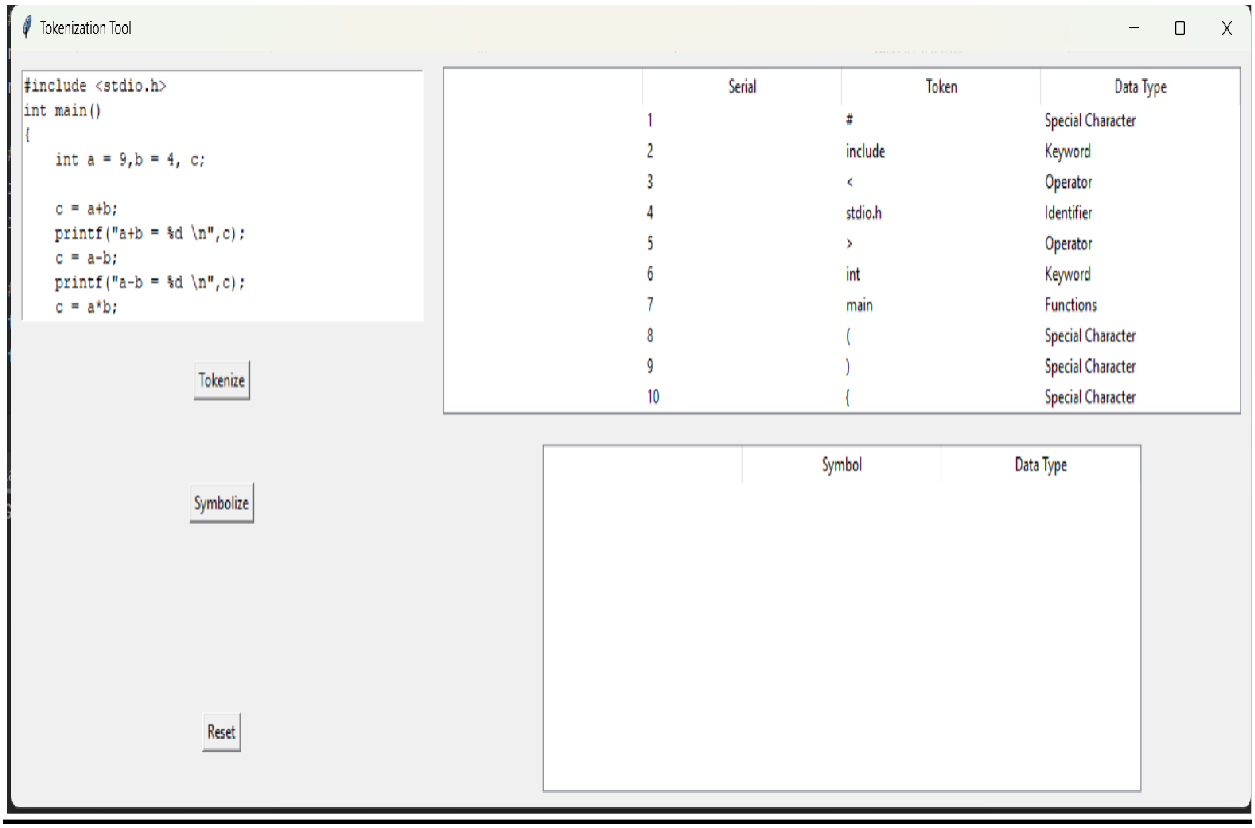


Input Buffering:



The source code was entered using the C programming language.

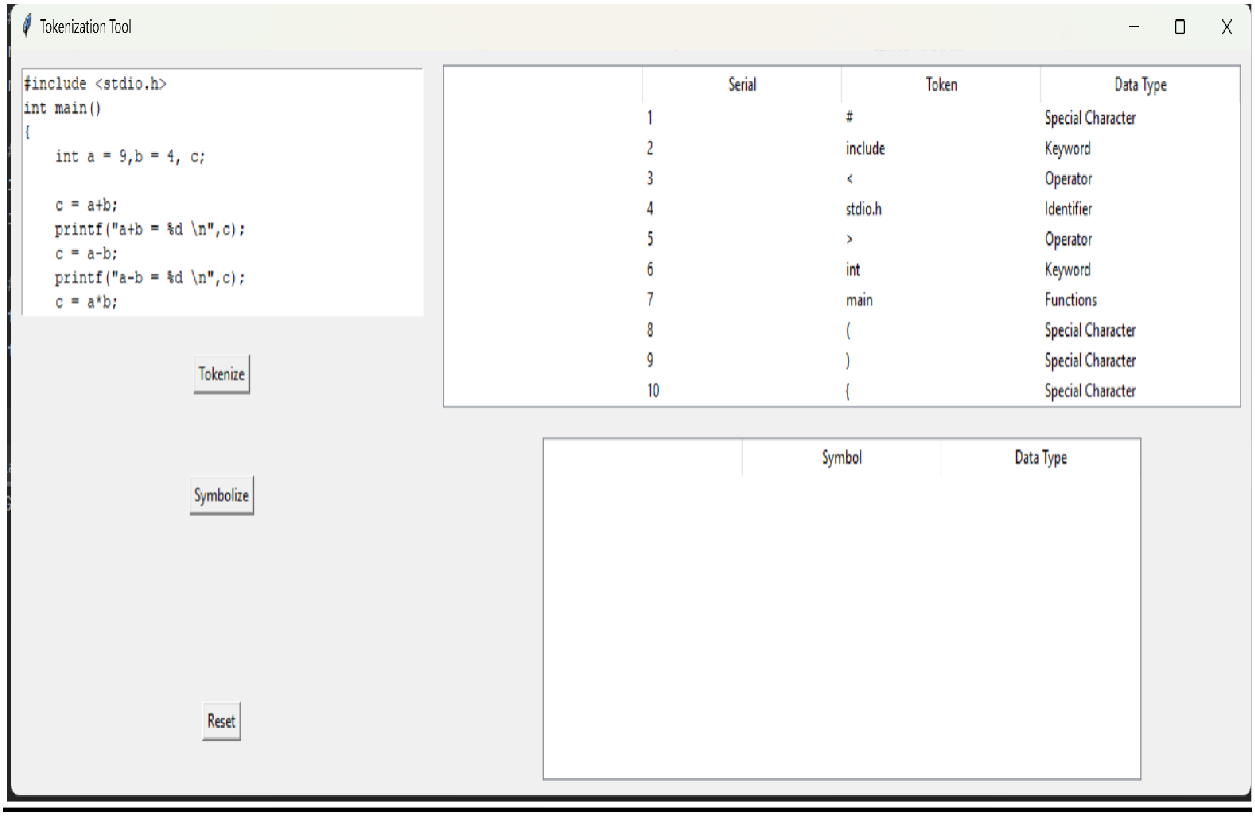
Tokenization:



The method of tokenization involves dividing a string of characters, such as source code, into tokens. Tokenization in the context of a lexical analyzer refers to the process of recognizing and classifying the various token types present in the source code. A meaningful unit, such as a term, identifier, operator, or literal value, is represented by each token.

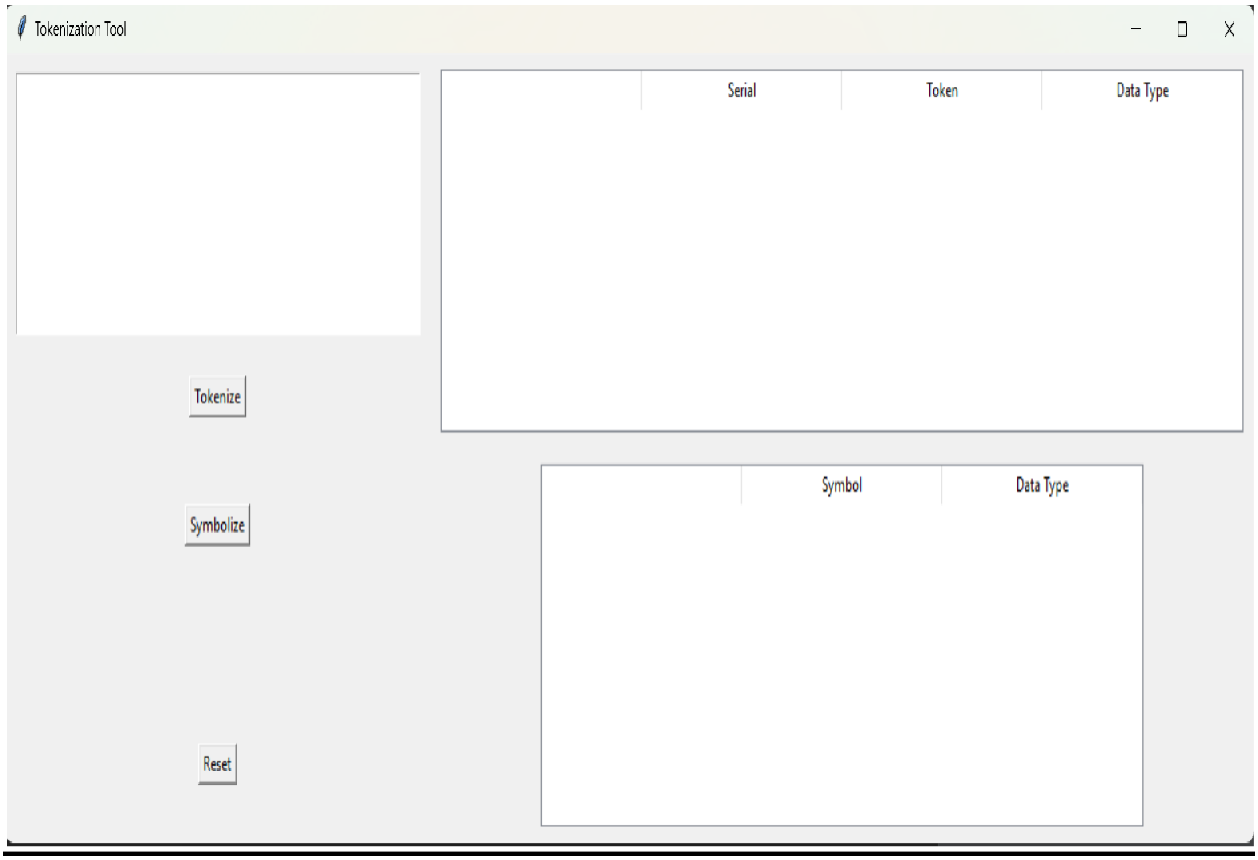
Symbol Table:

The only information in the symbol table for the name, line, and value of the identifier that has yet to be developed is a tokenized C code.



Reset:

A button that resets all form values to their starting points.



Conclusion:

By decomposing the source code into understandable tokens, identifying lexical mistakes, controlling the symbol table, and producing the token stream for subsequent processing, the lexical analyzer plays a critical part in the compilation process. It acts as the first stage in converting source code that is comprehensible by humans into a form that the compiler or interpreter can comprehend and process.