Compiler Design

Assignment No 1 :

C Syntax Analyzer and Tokenizer

Submitted To:

Eng / Ahmed Hesham

Compiler Design Assistant Lecturer

Submitted By:

Name: Ahmed Hussien El-Sayed Abd El Hamid

Academic ID: 15210408

Name: Ahmed Ibrahim El-Sayed Radwan

Academic ID: 15210405

Table of Contents

Introduction

This project involves creating a C syntax analyzer and tokenizer using Python. The analyzer is designed to tokenize C code, identify lexical components, check for basic syntax errors, and validate compliance with foundational C syntax rules. This tool is essential for understanding how compilers handle lexical analysis and syntax validation during the compilation process.

Lexical Analysis

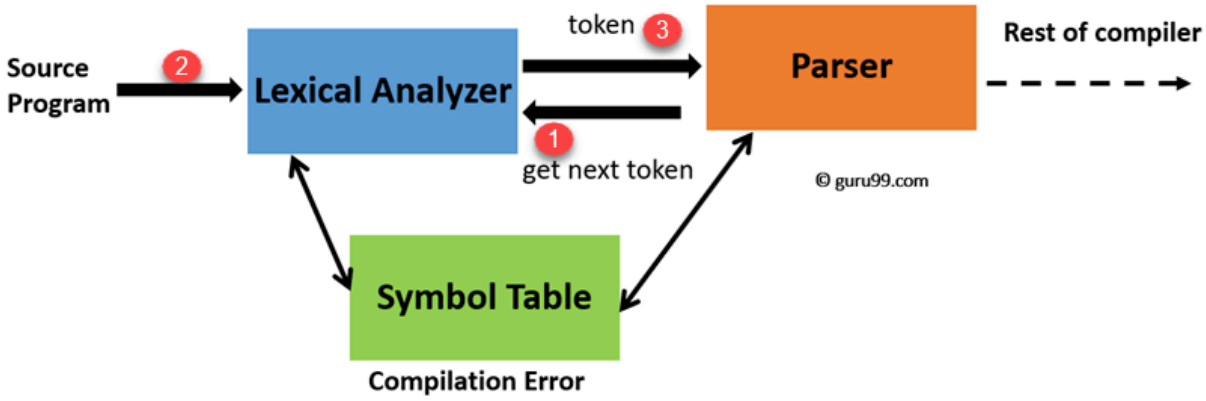
Lexical Analysis is the initial phase in compiler design, where the source code is transformed into tokens. A lexical analyzer, or lexer, processes the code by removing spaces and comments and dividing it into logical units called tokens, which represent language elements like keywords, operators, or identifiers. The primary tasks of lexical analysis include:

- ★ **Token Recognition**: Recognizing patterns and generating tokens from source code.
- ★ **Error Detection**: Reporting lexical errors such as invalid characters or misplaced keywords.
- ★ **Symbol Table Insertion**: Adding recognized tokens to the symbol table for use in later compilation phases.

For instance, in the sentence "int maximum(int x, int y)", lexical analysis would yield tokens like int (Keyword), maximum (Identifier), ((Operator), x (Identifier), etc.

Key Terminologies in Lexical Analysis

- **Lexeme**: A sequence of characters in source code that matches the pattern of a token.
- **Token**: A categorized unit of data from the lexeme, representing a keyword, identifier, operator, etc.
- **Pattern**: The rule or description that matches a lexeme to generate a token.



C Syntax Analyzer and Tokenizer

This script serves as a basic syntax analyzer and tokenizer for C code, utilizing regular expressions to identify and classify components of the code, such as keywords, operators, identifiers, and comments. The analyzer not only identifies tokens but also checks for syntax rules typical in C programming.

Features

- **Tokenization**: Identifies tokens such as keywords, preprocessor directives, identifiers, comments, literals, operators, and punctuators.
- **Syntax Checking**:
  - Verifies that #include directives are at the beginning of the code.
  - Ensures the presence of the main function.
  - Confirms balanced {} braces.
  - Validates that statements are terminated with semicolons.
- **Error Detection**: Reports issues including:
  - Keywords misused as identifiers.
  - Unmatched braces and missing semicolons.

Usage:

1. Run the script in a Python environment.
2. Input C code line by line.
3. Type done when done to start the analysis.
4. The script outputs tokenized data and lists any detected syntax errors.

Example Input:

Example Output

```
Tokenized Output:
=======================================
Token Type: PREPROCESSOR   | Token Value:  #include <stdio.h> | Line: 1
Token Type: KEYWORD        | Token Value: int                 | Line: 2
Token Type: IDENTIFIER     | Token Value: main                | Line: 2
Token Type: PUNCTUATOR     | Token Value: {                   | Line: 2
Token Type: KEYWORD        | Token Value: int                 | Line: 3
Token Type: IDENTIFIER     | Token Value: x                   | Line: 3
Token Type: ASSIGNMENT     | Token Value: =                   | Line: 3
Token Type: CONSTANTINT    | Token Value: 10                  | Line: 3
Token Type: PUNCTUATOR     | Token Value: ;                   | Line: 3
Token Type: FUNCTION       | Token Value: printf              | Line: 4
Token Type: STRING         | Token Value: "Hello World"       | Line: 4
Token Type: PUNCTUATOR     | Token Value: ;                   | Line: 4
Token Type: KEYWORD        | Token Value: return              | Line: 5
Token Type: CONSTANTINT    | Token Value: 0                   | Line: 5
Token Type: PUNCTUATOR     | Token Value: ;                   | Line: 5
Token Type: PUNCTUATOR     | Token Value: }                   | Line: 6
=======================================

Conclusion:
No syntax errors detected.
```

Conclusion

The C Syntax Analyzer and Tokenizer provides essential insights into lexical analysis and basic syntax validation for C code. By recognizing token types and flagging syntax errors, this tool supports learning about compiler operations and helps enforce language standards in C programming.