A black background with grey leaves

AI-generated content may be incorrect.

Lexical Analyzer

Build Scanner

A grey logo on a black background

AI-generated content may be incorrect.

**Prepared By**

Omar Mahmoud Ibrahim

Id:200042847

**Under Supervision**

Dr. Nahal Abdel Salam

A **compiler** is a program that translates source code from a high-level programming language into machine code, making it executable by a computer. The compilation process consists of several phases, each responsible for a different aspect of translation.

One of the most important phases in a compiler is the **Lexical Analysis phase**, which scans the source code and converts it into a sequence of tokens. This report provides an implementation of a **lexical analyzer** that can process a subset of the C++ programming language.

* 1. **Phases**

A compiler operates in multiple phases, including:

1. **Lexical Analysis:**
   * Converts source code into tokens (small meaningful units).
   * Detects identifiers, keywords, numbers, operators, and symbols.
2. **Syntax Analysis (Parsing):**
   * Checks if the sequence of tokens follows the correct grammar.
   * Constructs a **Parse Tree**.
3. **Semantic Analysis:**
   * Ensures the meaning of expressions is valid.
   * Detects **type mismatches** and **undefined variables**.
4. **Intermediate Code Generation:**
   * Converts the syntax tree into an intermediate representation.
5. **Code Optimization:**
   * Improves efficiency by reducing redundant instructions.
6. **Code Generation:**
   * Produces the final **machine code** for execution.
7. **Error Handling:**
   * Detects and reports errors at different compilation stages.
8. **Lexical Analyzer**

A **Lexical Analyzer (Scanner)** is responsible for breaking the input program into tokens. It reads the **source code character by character** and groups them into meaningful units such as:

* **Keywords:** int, float, if, else, while, return
* **Identifiers:** Variable names like x, y, myFunction
* **Numbers:** Integer and floating-point values like 10, 3.14
* **Strings:** "Hello World"
* **Operators:** +, -, \*, /, =, <, >, <=, >=, ==, !=
* **Special Symbols:** {, }, (, ), [, ], ;, ,

1. **Software Tools**

To implement the lexical analyzer, we used the following software tools:

* 1. **Computer Program**

 **Python 3.x:** Used for writing the lexical analyzer.

 **Regular Expressions (re module):** Used for pattern matching.

 **Text Editor (VS Code, PyCharm, .):** Used for writing and testing the code.

* 1. **Programming Language**

The lexical analyzer is designed to analyze the c++ lexiems into token using Pyhon programming language

1. **Implementation of a Lexical Analyzer**
2. **import** re
3. *# Define token types for C++*
4. TOKEN\_TYPES **=** {
5. "KEYWORD"**:** r"\b(int**|**float**|**if**|**else**|**while**|**for**|**return**|**void**|**char**|**string**|**double)\b"**,**
6. "IDENTIFIER"**:** r"\b[a-zA-Z\_][a-zA-Z0-9\_]**\***\b"**,**
7. "NUMBER"**:** r"\b\d**+**(\.\d**+**)**?**\b"**,**
8. "STRING"**:** r'"[**^**"]**\***"'**,** *# Matches anything inside double quotes*
9. "OPERATOR"**:** r"[+\-\*/=<>!]**+**"**,**
10. "SPECIAL\_SYMBOL"**:** r"[{}()\[\],;]"**,**
11. }
12. **def** lexical\_analyzer(**code**)**:**
13. tokens **=** []
14. *# Combine regex patterns*
15. combined\_regex **=** "|"**.**join(f"(?P<{name}>{pattern})" **for** name**,** pattern **in** TOKEN\_TYPES**.**items())
16. **for** match **in** re**.**finditer(combined\_regex**,** code)**:**
17. token\_type **=** match**.**lastgroup
18. lexeme **=** match**.**group(token\_type)
19. tokens**.**append((lexeme**,** token\_type))
20. **return** tokens
21. *# Sample C++ code as a string*
22. cpp\_code **=** """
23. int main() {
24. int x = 10;
25. float y = 4.78;
26. if (x < y) {
27. x = x + 1;
28. y = x + 3;
29. }
30. return 0;
31. }
32. """
33. tokens **=** lexical\_analyzer(cpp\_code)
34. print("\nTokens:")
35. **for** lexeme**,** token\_type **in** tokens**:**
36. print(f"{lexeme} --> {token\_type}")

**4.2 The Output:**

Tokens:

int --> KEYWORD

main --> IDENTIFIER

( --> SPECIAL\_SYMBOL

) --> SPECIAL\_SYMBOL

{ --> SPECIAL\_SYMBOL

int --> KEYWORD

x --> IDENTIFIER

= --> OPERATOR

10 --> NUMBER

; --> SPECIAL\_SYMBOL

float --> KEYWORD

y --> IDENTIFIER

= --> OPERATOR

4.78 --> NUMBER

; --> SPECIAL\_SYMBOL

if --> KEYWORD

( --> SPECIAL\_SYMBOL

x --> IDENTIFIER

< --> OPERATOR

y --> IDENTIFIER

) --> SPECIAL\_SYMBOL

{ --> SPECIAL\_SYMBOL

x --> IDENTIFIER

= --> OPERATOR

x --> IDENTIFIER

+ --> OPERATOR

1 --> NUMBER

; --> SPECIAL\_SYMBOL

y --> IDENTIFIER

= --> OPERATOR

x --> IDENTIFIER

+ --> OPERATOR

3 --> NUMBER

; --> SPECIAL\_SYMBOL

} --> SPECIAL\_SYMBOL

return --> KEYWORD

0 --> NUMBER

; --> SPECIAL\_SYMBOL

} --> SPECIAL\_SYMBOL

Process finished with exit code 0

## **5. References**

1. **Compilers: Principles, Techniques, and Tools** – Aho, Lam, Sethi, Ullman.
2. **Python Regular Expressions (re Module)** – Python Documentation.
3. **Lexical Analysis in C++ and Python** – GeeksforGeeks.
4. **Compiler Design – Lexical Analysis** – TutorialsPoint.