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**

Student : Yossef Samy Abdelhafiz  
ID: 200048891

**Under Supervision**

Nehal Abdelsalam

**Introduction  
This C++ program is a simple lexical analyzer (also known as a lexer or scanner) designed to analyze and classify components of a mathematical expression. The program reads an expression like:**

**"num1 \* (temp + 42) - result2"**

**and breaks it down into tokens — the smallest meaningful units. Tokens are the building blocks that a compiler or interpreter uses to understand the structure of the code. The lexer performs the first step in this process, which is tokenization.**

**The main features of this program include:**

1. **Identifying Tokens: The program scans the expression and identifies different types of tokens such as:**
   * **Identifiers (e.g., num1, temp, result2),**
   * **Operators (e.g., +, -, \*, /),**
   * **Parentheses (e.g., (, )),**
   * **Numbers (e.g., 42).**
2. **Handling Special Characters: The lexer detects different characters in the expression, including letters, digits, and operators, and classifies them appropriately.**
3. **Output: For each token identified, the program outputs the token type (e.g., IDENTIFIER, NUM) and its lexeme (the actual sequence of characters).**

**By executing this lexer, we can analyze a given expression, break it into meaningful parts, and prepare it for further processing (such as parsing or evaluation). This program is a fundamental component of a compiler or interpreter, helping to convert raw source code into a structured form that can be further interpreted or compiled.**

**Phases of Compiler:**

** Lexical Analysis (Scanning):**

* **What it does: This is what your code primarily performs. It reads the source code character by character and groups them into meaningful units called tokens. These tokens represent keywords, identifiers, operators, literals, and punctuation.**
* **Output: A stream of tokens.**
* **Your code's role: The readChar, skipSpaces, and scan functions work together to perform lexical analysis on the expr string. They identify identifiers (num1, temp, result2), operators (\*, +, -), numbers (42), and parentheses ((, )).**

** Syntax Analysis (Parsing):**

* **What it does: This phase takes the stream of tokens from the lexical analyzer and checks if they form a grammatically correct structure according to the rules of the programming language (the syntax). It builds a hierarchical structure of the code, often represented as an Abstract Syntax Tree (AST).**
* **Output: An Abstract Syntax Tree (AST) or a similar intermediate representation of the program's structure.**
* **Your code's role: Your code does not perform syntax analysis. It only identifies the individual tokens.**

** Semantic Analysis:**

* **What it does: This phase checks the meaning and consistency of the program. It performs tasks like type checking (ensuring operations are valid for the data types involved), variable declaration checks, and scope resolution.**
* **Output: An annotated Abstract Syntax Tree (AST) with type and other semantic information.**
* **Your code's role: Your code does not perform semantic analysis.**

** Intermediate Code Generation:**

* **What it does: In this phase, the compiler translates the semantically analyzed AST into an intermediate representation (IR). This IR is often a machine-independent code that is easier to optimize and translate into the target machine code. Examples of IR include three-address code.**
* **Output: Intermediate code.**
* **Your code's role: Your code does not perform intermediate code generation.**

** Code Optimization:**

* **What it does: This phase aims to improve the efficiency of the intermediate code by applying various optimizations, such as removing redundant computations, improving loop performance, and allocating registers effectively.**
* **Output: Optimized intermediate code.**
* **Your code's role: Your code does not perform code optimization.**

** Code Generation:**

* **What it does: This is the final phase in a compiler. It translates the optimized intermediate code into the target machine code (the specific instructions that the computer's processor can execute).**
* **Output: Target machine code.**
* **Your code's role: Your code does not perform code generation**

**Lexical Analyzer :**1 #include <iostream> // Input/output operations

2 #include <string> // String manipulation

3 #include <cctype> // Character handling (isalpha, isdigit, isspace)

4

5 using namespace std; // Use standard namespace

6

7 string expr = "num1 \* (temp + 42) - result2"; // Expression to analyze

8 string token; // Current token

9 char currentChar; // Current character

10 int currentClass; // Current character type

11 int currentPos = 0; // Current position in 'expr'

12

13 #define ID 1 // Identifier token type

14 #define NUM 2 // Number token type

15 #define OP 3 // Operator token type

16 #define PAREN 4 // Parenthesis token type

17 #define DONE -1 // End of input

18

19 void readChar() { // Read next character

20 if (currentPos < expr.length()) {

21 currentChar = expr[currentPos++];

22 if (isalpha(currentChar))

23 currentClass = ID;

24 else if (isdigit(currentChar))

25 currentClass = NUM;

26 else if (currentChar == '(' || currentChar == ')')

27 currentClass = PAREN;

28 else if (currentChar == '+' || currentChar == '-' || currentChar == '\*' || currentChar == '/')

29 currentClass = OP;

30 else

31 currentClass = -2; // Unknown

32 } else {

33 currentClass = DONE;

34 }

35 }

36

37 void skipSpaces() { // Skip whitespace

38 while (isspace(currentChar))

39 readChar();

40 }

41

42 int scan() { // Identify and process next token

43 token.clear();

44 skipSpaces();

45

46 switch (currentClass) {

47 case ID: // Identifier

48 while (isalnum(currentChar)) {

49 token += currentChar;

50 readChar();

51 }

52 cout << "Token: IDENTIFIER, Lexeme: " << token << endl;

53 return ID;

54

55 case NUM: // Number

56 while (isdigit(currentChar)) {

57 token += currentChar;

58 readChar();

59 }

60 cout << "Token: INTEGER, Lexeme: " << token << endl;

61 return NUM;

62

63 case OP: // Operator

64 token += currentChar;

65 cout << "Token: OPERATOR, Lexeme: " << token << endl;

66 readChar();

67 return OP;

68

69 case PAREN: // Parenthesis

70 token += currentChar;

71 cout << "Token: PARENTHESIS, Lexeme: " << token << endl;

72 readChar();

73 return PAREN;

74

75 case DONE: // End of input

76 cout << "End of expression." << endl;

77 return DONE;

78

79 default: // Unknown character

80 cout << "Unknown character: " << currentChar << endl;

81 readChar();

82 return -1;

83 }

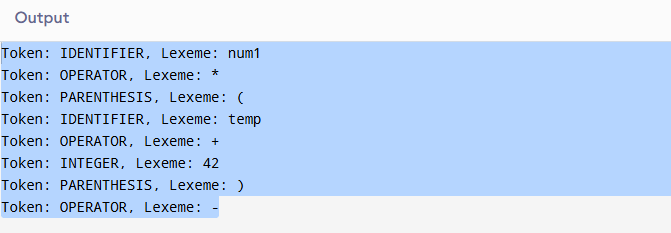
84 }

85

86 int main() { // Main function

87 readChar();

88 while (scan() != DONE);

89 return 0;   
90 }  
  


1. **3. Software Tools**
   1. **Computer Program :** visual studio
   2. **Programming Language : C++**

**Implementation of a Lexical Analyzer**

|  |  |
| --- | --- |
| **num1** | **Identifier** |
| \* | **Operator** |
| ( | **Left Parenthesis** |
| **Temp** | **Identifier** |
| **+** | **Operator** |
| **42** | **Integer Literal** |
| **)** | **Right Parenthesis** |
| - | **Right Parenthesis** |

|  |  |
| --- | --- |
| result2 | Identifier |

**References  
Book of Concepts of Programming Languages – Sebesta -E12**

**Important Note: -**

Technical reports include a mixture of text, tables, and figures. Consider how you can present the information best for your reader. Would a table or figure help to convey your ideas more effectively than a paragraph describing the same data?

Figures and tables should: -

* Be numbered
* Be referred to in-text, e.g. *In Table 1*…, and
* Include a simple descriptive label - above a table and below a figure.