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**

Name : Gomana ahmed

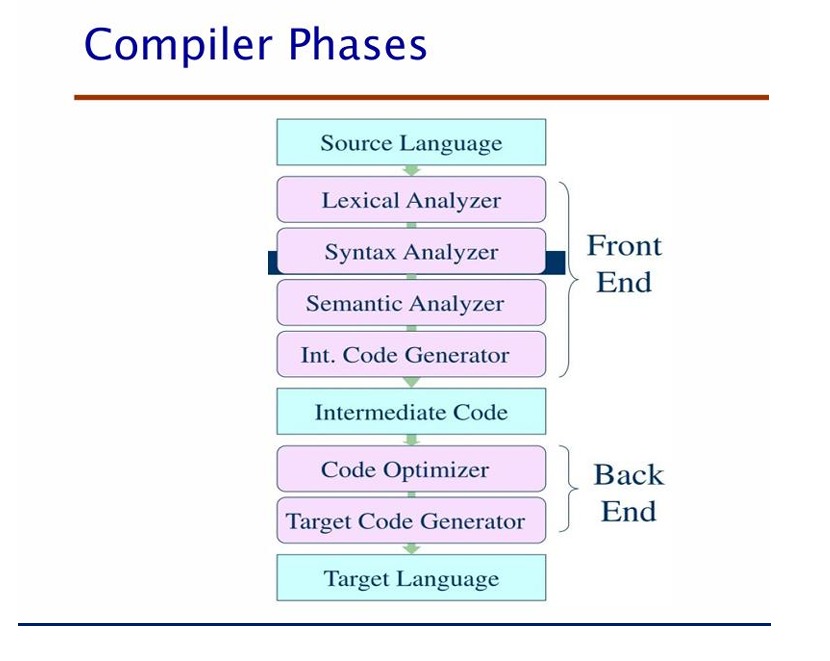
Id : 200042007

**Under Supervision**

Dr/ Nehal Abdelsalam

T.A : Fares eltommy

1. **Introduction** 
   1. **Phases of Compiler**

****

1. **Lexical Analyzer**

A **lexical analyzer** (also known as a lexer or scanner) is a critical component of a compiler or interpreter, responsible for breaking down source code into a sequence of tokens that can be further analyzed by the parser. In this project, we implement a simple **lexical analyzer** for arithmetic expressions, which can recognize identifiers, integer literals, operators, and parentheses.

The goal of this assignment is to create a **Lexical Analyzer System** that can process basic arithmetic expressions such as variable assignments, arithmetic operations (addition, subtraction, multiplication, division), and parentheses. The lexer classifies characters into different "character classes" (letters, digits, unknown symbols) and then maps these characters into predefined token types such as integer literals, identifiers, and operators.

This system reads an input file containing an arithmetic expression, processes it character by character, and outputs a list of tokens that can later be used for further syntax analysis. The lexer uses standard techniques like state machines to identify and categorize sequences of characters, which are essential for later stages of language processing.

The lexer is implemented in **C++** for efficiency and ease of development, and it leverages standard libraries such as fstream for file handling and cctype for character classification. The system is designed to be extensible, allowing for the addition of more complex features like floating-point numbers, string literals, and more sophisticated error handling.

This documentation will outline the design and functionality of the lexical analyzer, detailing its components, methods, and how it processes the input to identify valid tokens. It will also include a brief explanation of the core logic and state transitions that allow the analyzer to function correctly.

1. **Software Tools**
   1. **Computer Program**

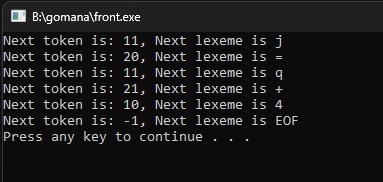
**( VISUAL STUDIO CODE )**

* 1. **Programming Language**

**( C++)**

1. **Implementation of a Lexical Analyzer**

**Code : j = q + 4**

****

1. **References**

|  |  |  |
| --- | --- | --- |
| **TOKEN NUMBER** | **TOKEN TYPE** | **LEXEME** |
| **1** | **IDENT** | **j** |
| **2** | **ASSIGN\_OP** | **=** |
| **3** | **IDENT** | **q** |
| **4** | **ADD\_OP** | **+** |
| **5** | **INT\_LIT** | **4** |

**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.