Lexical Analyzer  
Build Scanner

Prepared By  
Mahmoud Ahmed Hasballah  
Student ID: 200040427

Under Supervision  
Dr. Fares

# Introduction

A lexical analyzer, also known as a scanner, is a fundamental component of a compiler. Its primary role is to read the source code as a stream of characters and convert it into a stream of tokens. Tokens are sequences of characters that represent the smallest unit of meaning, such as keywords, identifiers, constants, operators, and punctuation.

# Phases of Compiler

The compiler is typically divided into several phases:  
1. Lexical Analysis  
2. Syntax Analysis  
3. Semantic Analysis  
4. Intermediate Code Generation  
5. Code Optimization  
6. Code Generation  
7. Error Handling

# Lexical Analyzer

The lexical analyzer reads the input program and divides it into tokens. It removes whitespace and comments and detects lexical errors. It plays a crucial role in ensuring the syntactic correctness of the source code.

# Software Tools

The lexical analyzer can be implemented using tools like Lex (flex in modern systems), which generate scanners based on regular expressions and rules defined by the user.

# Computer Program

The following is a simplified version of a lexical analyzer implemented in C++ that recognizes identifiers, numbers, and basic operators.

# Programming Language

The implementation is done in C++ due to its efficiency and widespread usage in system-level programming.

# Implementation of a Lexical Analyzer

```cpp  
#include <iostream>  
#include <cctype>  
using namespace std;  
  
void lexicalAnalyzer(string input) {  
 for (int i = 0; i < input.length(); i++) {  
 if (isalpha(input[i])) {  
 cout << "Identifier: " << input[i] << endl;  
 } else if (isdigit(input[i])) {  
 cout << "Number: " << input[i] << endl;  
 } else if (input[i] == '+' || input[i] == '-') {  
 cout << "Operator: " << input[i] << endl;  
 } else {  
 cout << "Unknown: " << input[i] << endl;  
 }  
 }  
}  
  
int main() {  
 string input = "a+3-b";  
 lexicalAnalyzer(input);  
 return 0;  
}  
```

# References

1. Aho, A.V., Lam, M.S., Sethi, R., & Ullman, J.D. (2006). Compilers: Principles, Techniques, and Tools.  
2. https://en.wikipedia.org/wiki/Lexical\_analysis  
3. https://www.geeksforgeeks.org/introduction-of-compiler-design/