A black background with grey leaves

AI-generated content may be incorrect.

Lexical Analyzer

Create a Scanner of the C-Minus Compiler

A grey logo on a black background

AI-generated content may be incorrect.

**Prepared By**

Belal Ahmed A. Saqer

200023029

**Under Supervision**

Dr. Nehal Abdel Salam

T.A. Mariam Farid

**Abstract**

**This project implements a lexical analyzer for the C-Minus compiler using Flex. The scanner reads C-Minus code and identifies tokens such as keywords, identifiers, numbers, and symbols. Flex simplifies token recognition through regular expressions. The analyzer produces a token stream that serves as the first step in the compilation process.**

##### ***Table of Contents***

|  |  |
| --- | --- |
| **Content** | **Page** |
| **Table of Contents ………………………………………………………………….** | 4 |
| **List of Tables ………………………………………………………………………** | 5 |
| **List of Figures ……………………………………………………………………..** | 6 |
| * 1. **Introduction ……………………………………………………………………** | **7** |
| * 1. **Description of Frontend of the Compiler …………………………………….** | **7** |
| * + 1. **Lexical Analyzer …………………………………………………………** | **7** |
| * + 1. **Syntax Analyzer ………………………………………………………….** | **8** |
| * + 1. **Semantic Analyzer ……………………………………………………….** | **8** |
| * 1. **Programming Language ……………………………………………….……...** | **9** |
| * + 1. **List of Reserved Words ………………………………………………….** | **9** |
| * + 1. **List of Keywords …………………………………………………………** | **9** |
| * + 1. **Basic Grammar Rules …………………………………………………...** | **9,10** |
| * 1. **Software Tool(s) ………….………….………….………….………….………** | **10,11** |
| * 1. **Discussion ………………………………………….…………………………...** | **11** |
| * + 1. **Input of Lexical Analyzer ……………………………………………….** | **11** |
| * + 1. **Output of Lexical Analyzer ……………………………………………..** | **11,12** |

##### ***List of Tables***

|  |  |  |
| --- | --- | --- |
| **Table No.** | **Title** | **Page** |
| Table 1 | List of Tokens for C-Minus Scanner | 10 |
| Table 2 | List of Project Files | 13,14 |
|  |  |  |

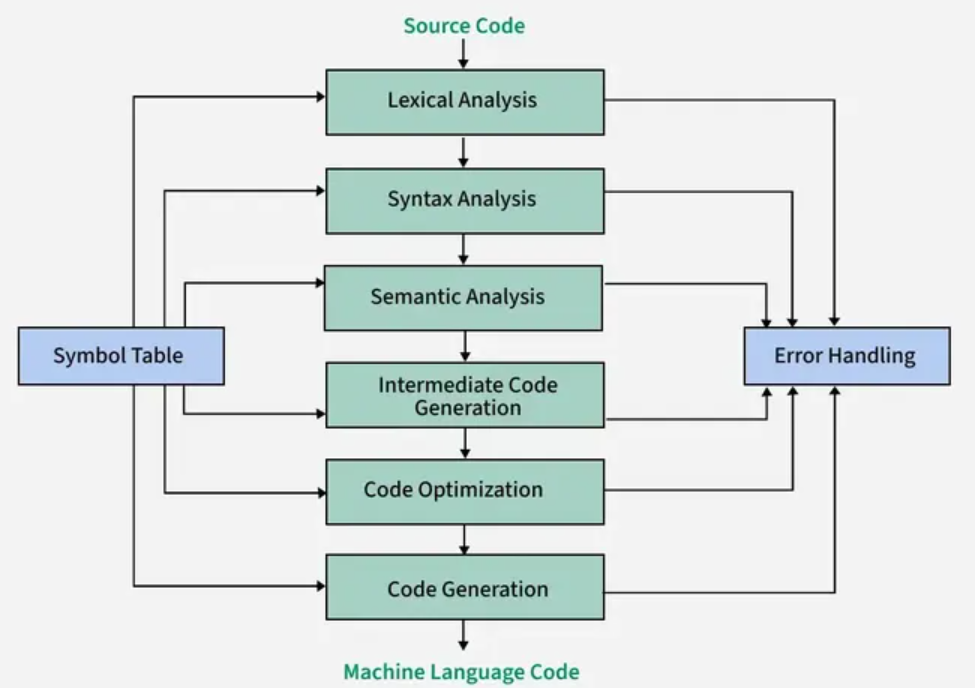
##### ***List of Figures***

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Caption** | **Page** |
| Figure 1 | Phases of Compiler | 7 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. **Introduction**

**A compiler is a fundamental tool that translates source code written in a high-level programming language into a lower-level language, typically machine code. The compilation process is divided into several phases, each responsible for a specific aspect of translation and optimization. These phases include lexical analysis, syntax analysis, semantic analysis, optimization, and code generation.**

* 1. **Phases of Compiler**

****

**As illustrated in Figure 1, the first phase, known as Lexical Analysis, plays a crucial role in reading the source code, breaking it down into meaningful units called tokens, and filtering out unnecessary characters like white spaces and comments. This step prepares the code for subsequent stages of compilation.**

**In this project, we focus on building a Lexical Analyzer for the C-Minus programming language. We use Flex, a powerful lexical analyzer generator, to automate the recognition of tokens such as keywords, identifiers, operators, and symbols. This scanner serves as the first step in constructing a full C-Minus compiler.**

1. **Description of Frontend of the Lexical Analyzer**

**The frontend of a compiler is responsible for analyzing the source code to ensure it is syntactically and semantically correct before generating intermediate or target code. It mainly includes three critical phases: lexical analysis, syntax analysis, and semantic analysis.**

* 1. **Lexical Analyzer**

**The lexical analyzer (scanner) is the first phase of the compiler. It reads the raw source code character by character and groups them into meaningful sequences called tokens. Tokens represent entities such as keywords, identifiers, constants, operators, and symbols. In this project, Flex was used to implement the scanner, where each token type is defined by a regular expression. The scanner also removes comments and whitespace from the input source code.**

* 1. **Syntax Analyzer**

**The syntax analyzer (parser) processes the sequence of tokens generated by the lexical analyzer to determine the grammatical structure of the source code. It checks whether the code follows the rules of the language’s grammar, typically described by context-free grammar (CFG). Syntax errors are detected at this stage, and a parse tree is often constructed to represent the hierarchical syntactic structure of the code.**

* 1. **Semantic Analyzer**

**The semantic analyzer uses the parse tree generated by the syntax analyzer to ensure that the program’s meaning is correct. It checks for semantic errors, such as type mismatches, undeclared variables, or incorrect function calls. It also manages the symbol table, which keeps track of variable and function declarations.**

1. **Programming Language**

**The C-Minus language is a simplified version of the C programming language, designed primarily for educational purposes to study compiler construction. It includes basic programming constructs such as variables, control structures, input/output operations, and function definitions.**

* 1. **List of Reserved Words**

**The reserved words are predefined identifiers that cannot be used for other purposes like naming variables or functions. In C-Minus, the reserved words include:**

**If, else, int, return, void, while, read, write, for**

**These words have specific meanings defined by the language syntax and semantics.**

* 1. **List of Keywords**

**Keywords in C-Minus are a subset of the reserved words that primarily guide the structure and flow of a program. The keywords include:**

**if, else, int, return, void, while, read, write, for**

**(Note: In C-Minus, all reserved words are also treated as keywords.)**

* 1. **Basic Grammar Rules**

**The basic grammar of C-Minus can be represented using context-free grammar (CFG) rules. Some examples include:**

**Program → DeclarationList**

**DeclarationList → Declaration DeclarationList | ε**

**Declaration → VarDeclaration | FunDeclaration**

**VarDeclaration → TypeSpecifier ID ; | TypeSpecifier ID [ NUM ] ;**

**FunDeclaration → TypeSpecifier ID ( Params ) CompoundStmt**

**TypeSpecifier → int | void**

**These rules help define how tokens are structured into syntactically valid programs.**

|  |  |
| --- | --- |
| **Token Type** | **Ex. Tokens** |
| **KeyWord** | **If, else, int, return, void, while, read, write, for** |
| **Symbol** | **{,},(,),=,;,<,>=** |
| **Identifier** | **Max,findMax,a,b,numValues,maxValue,I,Value,result** |
| **Operator** | **=,>,<=** |

1. **Software Tool(s)**

**In this project, several software tools were used to implement and test the lexical analyzer for the C-Minus compiler:**

**4.1 Flex (Fast Lexical Analyzer Generator):**

**Flex is a tool for generating scanners (lexical analyzers). It takes as input a set of token definitions written as regular expressions and produces a C program (lex.yy.c) that performs the scanning. Flex greatly simplifies the development of the lexical analyzer by automating the token recognition process.**

**4.2 GCC (GNU Compiler Collection):**

**GCC was used to compile the generated C files (lex.yy.c) into an executable (lex.exe). This executable reads the C-Minus source code and produces a sequence of tokens as output.**

**4.3 CMD (Command Prompt):**

**The Command Prompt (CMD) was used to execute Flex commands, compile the C files using GCC, and run the resulting executable. It provided a simple and efficient environment for testing and debugging the scanner.**

**4.4 VS Code (Visual Studio Code):**

**Visual Studio Code was used as the code editor to write, edit, and organize the project files, along with extensions for C/C++ and Flex support.**

1. **Discussion**

**This section discusses the functionality and behavior of the lexical analyzer, focusing on its input and output handling.**

* 1. **Input of Lexical Analyzer**

**The input to the lexical analyzer is a source code file written in the C-Minus programming language.**

**Typically, the source file contains variable declarations, control structures like if and for statements, function definitions, and input/output operations using read and write.**

**The scanner reads this file character by character and processes it according to the token definitions provided through regular expressions in the Flex file.**

**For example, the input file tiny.txt contains a simple C-Minus program that reads numbers, compares values, and writes the maximum value.**

* 1. **Output of Lexical Analyzer**

**The output of the lexical analyzer is a sequence of recognized tokens.**

**Each token is identified by its type (such as KEYWORD, IDENTIFIER, NUMBER, SYMBOL, or OPERATOR) and is optionally accompanied by a lexeme (the exact text from the source file).**

**The output is typically written to a result file (such as result.txt) or displayed in the console.**

**An example of the output could look like:**

**1: reserved word: int**

**1: ID, name= max**

**1: (**

**1: reserved word: int**

**1: ID, name= a**

**1: ,**

**1: reserved word: int**

**1: ID, name= b**

**1: )**

**1: {**

**2: reserved word: if**

**2: (**

**2: ID, name= a**

**2: >**

**2: ID, name= b**

**2: )**

**Etc…**

1. **Conclusion**

**In this project, a lexical analyzer for the C-Minus compiler was successfully designed and implemented using Flex.**

**The scanner effectively reads C-Minus source code, identifies various types of tokens such as keywords, identifiers, numbers, operators, and symbols, and produces a structured sequence of tokens as output.**

**The use of Flex significantly simplified the development process by allowing regular expressions to define token patterns automatically, reducing manual coding errors.**

**The resulting lexical analyzer serves as the foundation for the next stages of compilation, namely syntax analysis and semantic analysis.**

**Overall, this project demonstrated the importance of lexical analysis in the compiler construction process and provided practical experience in using automated tools like Flex to generate reliable and efficient scanners.**

1. **References**

* **Lecture from (CS-352)**
* **Compilers: Principles, Techniques, & Tools (Second Edition)**
* **Compiler Construction**
* **GeeksforGeeks**

1. **Appendices**

|  |  |
| --- | --- |
| **File Name** | **Purpose** |
| **GLOBALS.H** | **Global variables and constants definitions** |
| **lex.yy / lex.yy.c** | **Lexical analyzer generated by Flex** |
| **lex.c** | **Scanner implementation file** |
| **lex.exe** | **Executable file of the scanner** |
| **SCAN.H** | **Header file for scanner functions** |
| **UTIL.C** | **Utility functions implementation** |
| **UTIL.H** | **Header file for utility functions** |
| **result.txt** | **Output results from scanning** |
| **tiny.txt** | **Test input file for the scanner** |

**\*Attachment at eLearining.\***

* **Appendix A: Source Code of the Scanner (Flex File)**
* **Appendix B: Sample Input File (tiny.txt)**
* **Appendix C: Sample Output File (result.txt)**
* **Appendix D: Global Definitions and Utility Files (GLOBALS.H, UTIL.C, UTIL.H)**

**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.