A black background with white leaves

Description automatically generated

**LEX/FLEX**

**Scanner Generator in C**



**Prepared By**

Student ID

Mohamed Yasser 200013244

Momen Ashraf 200013269

Nader Mamdouh 200013285

Eman Khaled 200013190

Shahd Hatim 200018584

**Under Supervision**

Name of Doctor: Nehal A. Mohammed

Name of T. A: Toka Abdelhamid

**Summary**

A compiler is a software tool that translates human-readable source code written in a programming language into machine-readable code, typically in the form of binary instructions that a computer can execute. The compilation process involves several stages, starting with lexical analysis and ending with code generation and linking.

The lexical analyzer, often referred to as the laxer or scanner, is the initial phase of a compiler. Its primary task is to break down the source code into meaningful tokens. These tokens are the basic building blocks of the language and include keywords, identifiers, literals, and operators. The lexical analyzer ignores whitespace and comments, focusing solely on extracting tokens

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

|  |  |
| --- | --- |
| **Content** | **Page** |
| **Table of Contents ………………………………………………………………….** |  |
| **List of Tables ………………………………………………………………………** |  |
| **List of Figures ……………………………………………………………………..** |  |
| * 1. **Introduction ……………………………………………………………………** |  |
| * 1. **Description of Frontend of the Compiler …………………………………….** |  |
| * + 1. **Lexical Analyzer ……………………………………………………** | **1** |
| * + 1. **Syntax Analyzer …………………………………………………….** |  |
| * 1. **Programming Language ……………………………………………….……...** |  |
| * + 1. **List of Reserved Words ………....………………………………….** |  |
| * + 1. **List of Keywords ………....…………………………………………** |  |
| * + 1. **Basic Grammar Rules ……………….………....…………………...** |  |
| * 1. **Software Tools ………….………….………….………….………….………...** |  |
| * 1. **Screenshots ………………………………………….…………………………** |  |
| * + 1. **Screenshots of Input of Frontend of the Compiler ……………….**        1. **Input of Lexical Analyzer …………………………………..**        2. **Input of Syntax Analyzer …………………………………..** |  |
| * + 1. **Screenshots of Output of Frontend of the Compiler ……………..**        1. **Output of Lexical Analyzer ………………………………...**        2. **Output of Syntax Analyzer ………………………………...** |  |

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

|  |  |  |
| --- | --- | --- |
| **Table No.**  **1.1**  **1.2**  **1.2.1**  **1.2.2**  **1.3**  **1.3.1**  **1.3.2**  **1.3.3**  **1.4**  **1.5**  **1.5.1**  **1.5.1.1**  **1.5.1.2**  **1.5.2**  **1.5.2.1**  **1.5.2.2** | **Title**  **Introduction**  **Description of Frontend of the Compiler**  **Lexical Analyzer**  **Syntax Analyzer**  **Programming Language**  **List of Reserved Words**  **List of Keywords**  **Basic Grammar Rules**  **Software Tools**  **Screenshots**  **Screenshots of Input of Frontend of the Compiler**  **Input of Lexical Analyzer**  **Input of Syntax Analyzer**  **Screenshots of Output of Frontend of the Compiler**  **Output of Lexical Analyzer**  **Output of Syntax Analyzer** | **Page**  **4**  **4**  **4**  **5**  **5**  **5**  **6**  **6**  **7**  **8**  **8**  **9**  **10**  **10**  **13** |

***List of Figures***

|  |
| --- |
| **Figure No. Caption Page**  **1.5.1.1 Input of Lexical Analyzer 8**  **1.5.1.2 Input of Syntax Analyzer 9**  **1.5.2.1 Output of Lexical Analyzer 10**  **1.5.2.2 Output of Syntax Analyzer 13** |

* 1. **Introduction**

Compilers are essential software tools that translate high-level programming languages into machine-readable code, enabling computers to execute programs. By undergoing a series of phases like lexical analysis, syntax parsing, semantic analysis, optimization, and code generation, compilers convert abstract code structures into efficient machine code. This succinct introduction aims to highlight the critical role and intricate processes involved in compilers, essential for software development across various computing platforms.

* 1. **Description of frontend of the compiler**

The frontend of a compiler is responsible for the initial stages of the compilation process, where the source code is analyzed and transformed into an intermediate representation that can be further processed by the backend. Here's a breakdown of the typical components and their functions in the frontend of a compiler:

* + 1. **Lexical Analyzer**

lexical analysis is the initial processing step that breaks down a stream of characters (source code) into meaningful units called tokens. It's like dissecting a sent

* + 1. **Syntax Analyzer**

A syntax analyzer, also called a parser, is another program that works in conjunction with the lexical analyzer during compilation. Following the lexical analyzer's work, the syntax analyzer takes over to analyze the structure of the code.

**1.3 Programming Language**

C-Minus is a simplified version of the C language designed for educational purposes. It removes some of the more complex features of C to focus on core programming concepts like data types, control flow, and functions. This makes it easier for beginners to learn the fundamentals of programming without getting overwhelmed by the intricacies of a full-fledged language like C.

**1.3.1 List of Reserved Word**

In C-Minus, as with most programming languages, there are reserved words that have special meanings and cannot be used for other purposes like variable or function names. While a definitive list online might be challenging to find due to C-Minus being a more specialized language, here's what you can expect:

* Common keywords found in C, such as:
  + if, else, for, while
  + int, float, char
  + return, void

**1.3.2** **List of Keywords**

As mentioned earlier, there might be a slight terminology difference here. In most programming languages, including C-Minus, there's no real distinction between reserved words and keywords. Both terms refer to the same concept: words with special meanings within the language. Therefore, the information provided in 1.3.1 about reserved words applies to keywords in C-Minus as well.

**1.3.3 Basic Grammar Rules**

C-Minus follows a similar grammar structure to C. Here are some fundamental grammar rules you can expect:

* **Programs:** A program consists of functions and variable declarations.
* **Statements:** These are instructions that the program executes. Examples include assignments, control flow statements (if, for, while), function calls.
* **Expressions:** These combine variables, constants, and operators to compute values.
* **Variables:** These are named storage locations that hold data. They need to be declared with a specific data type before use.
* **Data Types:** C-Minus likely supports basic data types like int (integers), float (floating-point numbers), and char (characters).
* **Functions:** These are reusable blocks of code that perform specific tasks. They can take arguments and return values.
  1. **Software Tools**:
* **TINY compiler**:

is a simple programming language designed for educational purposes.

* **Flex Software:**

is a powerful tool for generating lexical analyzers (scanners) for programming languages. It is often used alongside Bison (a parser generator) in the development of compilers.

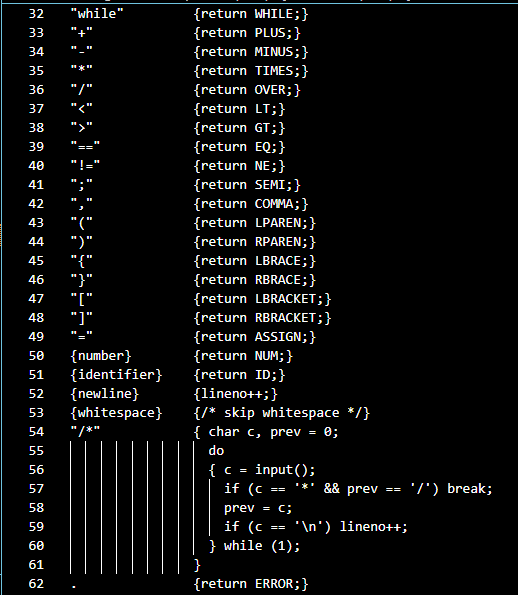
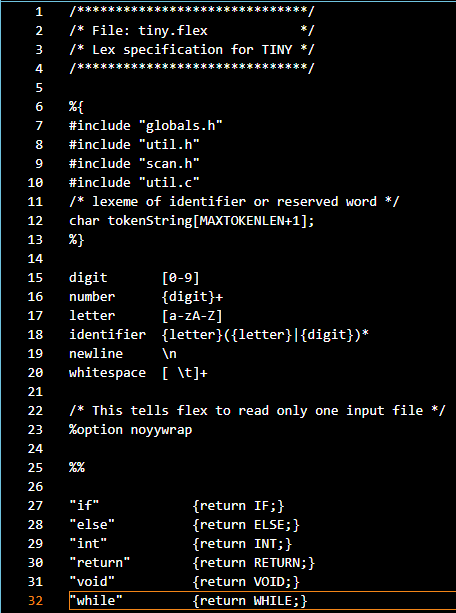
Flex simplifies the process of creating lexical analyzers by allowing developers to specify patterns for token recognition using regular expressions. These patterns are then compiled into efficient C code for use within a compiler.

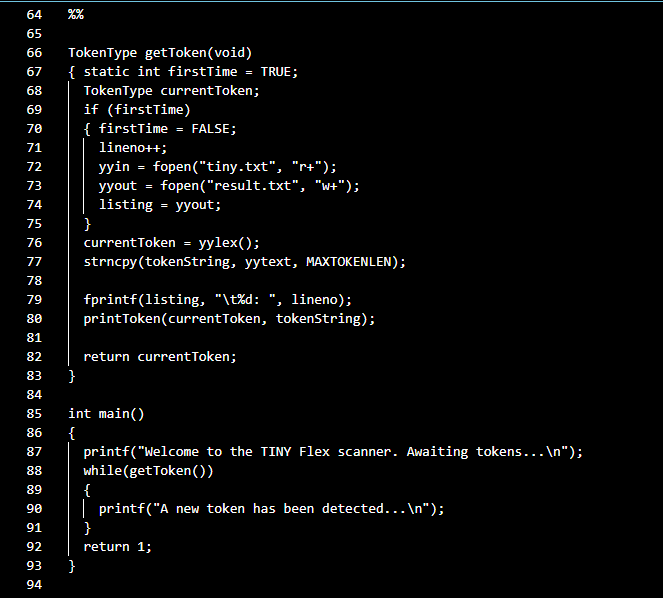
* **C-Minus Language:** is a simple subset of the C programming language, designed for educational purposes. It includes a minimal set of features found in C, making it suitable for teaching compiler construction and systems programming concepts.

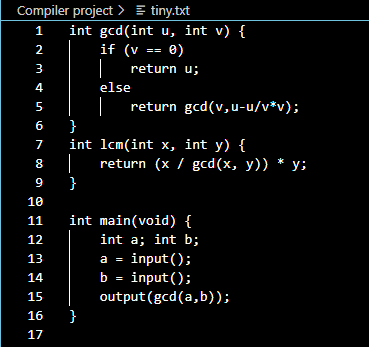
C-Minus serves as a practical vehicle for teaching compiler design and implementation. Its simplicity allows students to focus on core compiler concepts without being overwhelmed by the complexities of a full-fledged language like C.

**1.5.1 Screenshots of Input of Frontend of the Compiler**

**1.5.1.1 Input of Lexical Analyzer**

****

****

****

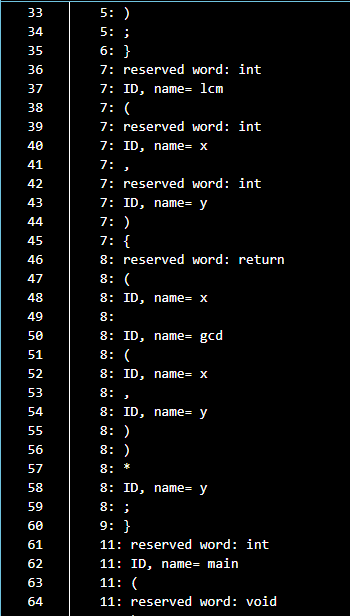
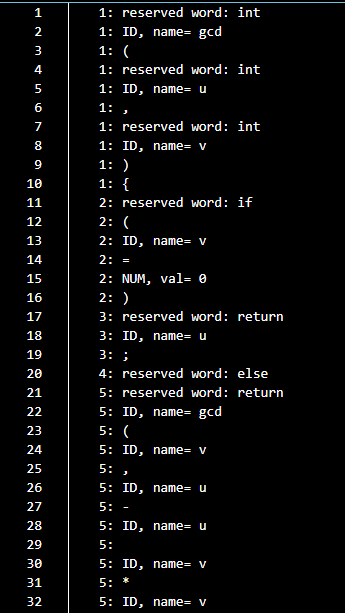
**1.5.1.2 Input of Syntax Analyzer**

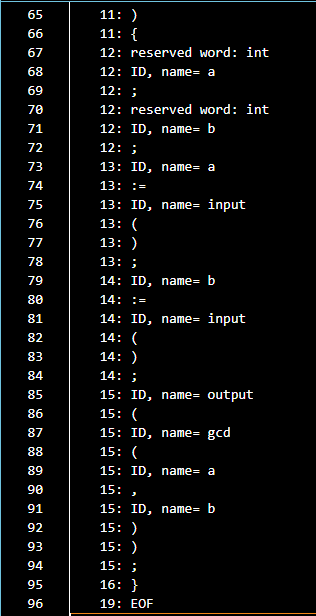
**A computer code on a black background

Description automatically generated**

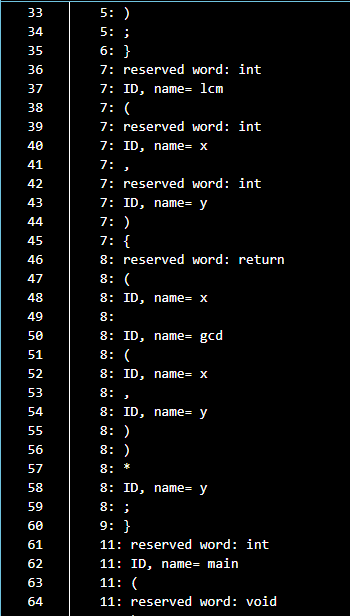
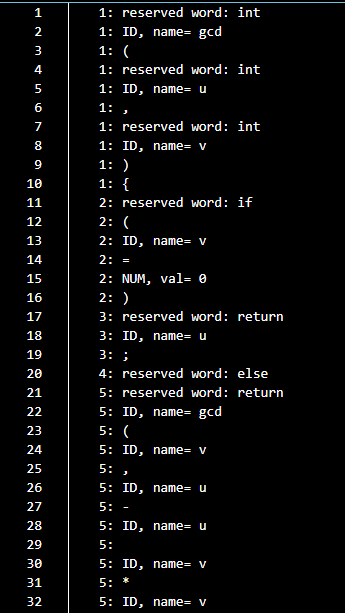
**1.5.2 Screenshots of Output of Frontend of the Compiler**

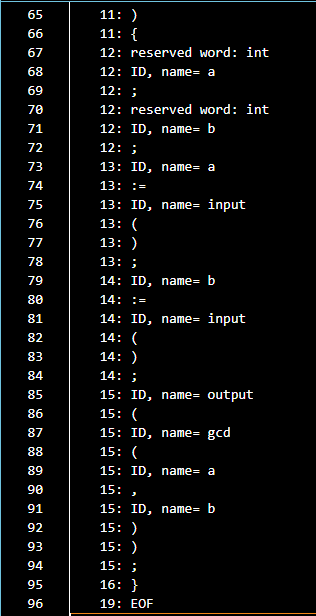
**1.5.2.1 Output of Lexical Analyzer**

****

****

**1.5.2.1 Output of Syntax Analyzer**

****

****

* **Conclusion**

compilers and lexical analyzers are essential for software development, translating high-level code into machine-readable instructions. Compilers meticulously process code through various stages, ensuring efficient execution across platforms. Meanwhile, lexical analyzers break down source code into tokens, facilitating this process. Together, they enable programmers to express complex logic, driving innovation in the digital sphere.

* **References**
* geeks
* **Appendices**
* Compilers Principles & techniques and Tools