****

**Project compiler design: Check Positive or Negative**

**CS - 352**

****

**Prepared By**

**Eyad Hesham Almallah 200010290**

**Habiba Helmy Emam 200012093**

**Farah Atef Ayad 200013197**

**Mayar Magdy 200013290**

**Under Supervision**

**D\Nehal A. Mohamed**

**T. A.\Toka Abdelhamed**

**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. **Output of Lexical Analyzer ………………………………...** |  |
| * + 1. **Screenshots of Output of Frontend of the Compiler ……………..**        1. **input of syntax Analyzer ………………………………...**        2. **Output of Syntax Analyzer ………………………………...** |  |

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.**

**1.2 Description of Frontend of the Compiler:**

**The frontend of the compiler is responsible for processing the source code and converting it into a form that is suitable for further analysis and translation.**

**1.2.1 Lexical Analyzer:**

**The Lexical Analyzer, also known as a scanner or tokenizer, is the first phase of the compiler frontend. It breaks down the source code into a sequence of tokens. Tokens are the smallest units of a programming language, such as keywords, identifiers, operators, and literals. The Lexical Analyzer removes comments and whitespace and identifies each token along with its corresponding lexeme.**

**1.2.2 Syntax Analyzer:**

**The Syntax Analyzer, also known as a parser, is the second phase of the compiler frontend. It takes the stream of tokens produced by the Lexical Analyzer and analyzes their structure according to the grammar rules of the programming language. The Syntax Analyzer constructs a parse tree or syntax tree, which represents the syntactic structure of the source code. This parse tree serves as input for further stages of the compiler, such as semantic analysis and code generation**

**1.3 Programming Language**

**1.3.1 List of Reserved Words:**

**if**

**else**

**int**

**return**

**void**

**while**

**Equal**

**ASSIGN**

**identifier**

**1.3.2 List of Keywords:+**

**-**

**\***

**/**

**+**

**<**

**>**

**<=**

**>=**

**==**

**!=**

**;**

**,**

**(**

**)**

**{**

**}**

**[**

**]**

**1.3.3 Basic Grammar Rules:**

**Identifiers:**

**Start with a letter (lowercase or uppercase).**

**Followed by zero or more letters or digits.**

**Numbers:**

**Consist of one or more digits.**

**Whitespace:**

**Spaces or tabs.**

**Newline:**

**Denoted by \n.**

**Comment**

**1.4 Software Tools**

* **TINY compiler: is a simple programming language designed for educational purposes. It is often used in compiler construction courses to demonstrate the implementation of basic compiler components.**

**The TINY compiler serves as a learning tool for understanding the fundamentals of compiler design and implementation. It typically involves building a compiler that translates TINY source code into machine code or an intermediate representation.**

* **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 Lexical Analyzer**

**A screenshot of a computer code

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**1.5.1.2Screenshots of output of Lexical Analyzer**

**A screenshot of a computer program

Description automatically generated**

**1.5.2.1 Screenshots of input of syntax Analyzer**

**A screenshot of a computer program

Description automatically generated**

**1.5.2.2 Screenshots of output of syntax Analyzer**

**A screenshot of a computer program

Description automatically generated**

1. **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.

1. **References**
2. **geeks**
3. **ChatGPT**
4. **Appendices**
5. **Compiler Constructions (Kenneth c. Louden)**
6. **Compilers Principles & techniques and Tools**