CISC 458

PHASE 1: LEXER ANALYSIS

Shams Sajid

2025

# Lexical Analyzer Documentation

## 1. Overview

This document provides an overview of the lexical analyzer (lexer) implemented for Phase 1 of the compiler project. The lexer scans source code, identifies tokens, and reports lexical errors.

## 2. Token Grammar

The lexer recognizes the following token types:

#### 2.1 Keywords

* if
* repeat
* until

#### 2.2 Identifiers

* A valid identifier starts with a letter (a-z, A-Z) and can contain letters, digits (0-9), and underscores (\_).
* Example: var\_name, x1, someVar

#### 2.3 Numbers

* A number consists of one or more digits (0-9).
* Example: 123, 4567

#### 2.4 Operators

* Arithmetic Operators: +, -, \*, /
* Assignment Operator: =
* Relational Operators (planned for future phases): ==, !=, <=, >=

#### 2.5 String Literals

* Strings are enclosed in double quotes ("").
* Supports escape sequences: \", \\, \n
* Example: "Hello, world!"

#### 2.6 Delimiters

* ;, {, }, (, )

#### 2.7 Comments

* Single-line comments: // This is a comment
* Multi-line comments: /\* This is a multi-line comment \*/

## 3. Features Implemented

* Token recognition for numbers, keywords, identifiers, operators, strings, delimiters, and comments.
* Error handling for invalid characters, unterminated strings, and unterminated comments.
* Line tracking for accurate error reporting.
* Escape sequence support in string literals.
* Comment handling, skipping over single-line (//) and multi-line (/\* \*/) comments.

## 4. Error Codes and Handling

The lexer reports the following lexical errors:

|  |  |
| --- | --- |
| **Error Type** | **Description** |
| ERROR\_INVALID\_CHAR | Invalid character detected in input. |
| ERROR\_INVALID\_NUMBER | Incorrect number format (not applicable yet). |
| ERROR\_CONSECUTIVE\_OPERATORS | Consecutive operators detected (e.g., ++). |
| ERROR\_UNTERMINATED\_STRING | String is missing a closing quote ("). |
| ERROR\_UNTERMINATED\_COMMENT | Multi-line comment is missing a closing \*/. |

## 5. Design Decisions

#### 5.1 Distinguishing Between Keywords and Identifiers

* The lexer first checks if a sequence of characters matches a keyword.
* If not, it is treated as an identifier.
* Implemented using a keyword lookup function (is\_keyword()).

#### 5.2 Multi-Character Operators

* The lexer does not currently support ==, !=, <=, >=, but they can be added by modifying the operator recognition logic.

#### 5.3 Error Messages

* Errors are reported with line numbers and descriptions.
* Example: Lexical Error at line 3: Unterminated string

#### 5.4 Handling Comments

* The lexer skips over single-line (//) and multi-line (/\* \*/) comments.
* If a multi-line comment is not closed, an error is generated.

## 6. Test Cases and Expected Outputs

#### 6.1 Valid Input Example

**Input:**

**A screenshot of a computer program

Description automatically generated**

**Expected Output:**

**A screenshot of a computer screen

Description automatically generated**

## 7. Conclusion

The lexical analyzer is fully implemented, supporting token recognition, error handling, and essential language constructs. It serves as a foundation for the next phase: Syntax Analysis (Parsing).