**Modern Lexical Analyzer Project**

**Compiler Construction Project - Report**

**Project Title:** Lexical Analysis


**Team Members**

1. **Muhammad Awais** - Lead Developer & Code Implementation
2. **Mohsin Ali** - Documentation & Report Creation

**Report Updates:** Muhammad Awais


## Project Specifications

**Course Information:**

**Subject:** Compiler Construction

**Project Title:** Lexical Analysis

**Institution:** Sukkur IBA University

**Subject Supervisor:** Madam Faryal Shamsi

**Academic Year:** 2025

# Project Overview

A sophisticated desktop application that performs lexical analysis on C programming language source code, featuring a modern GUI with comprehensive token visualization and symbol table generation capabilities.

# Core Features

## 1. Tokenization Engine

- Comprehensive pattern recognition using regular expressions
- Supports multiple token types: Keywords, Data Types, Operators, Separators, Identifiers, Literals, Comments
- Line-by-line parsing with accurate position tracking
- Handles complex C language constructs including headers, strings, and nested comments

## 2. Token Classification

The analyzer identifies and categorizes:

- **Keywords:** if, else, for, while, return, break, continue, switch, case, etc.
- **Data Types:** int, float, double, char, void, bool, struct, enum, union
- **Operators:** Arithmetic (+, -, *, /), Relational (==, !=, <, >), Logical (&&, ||, !)
- **Separators:** Parentheses, braces, semicolons, commas
- **Identifiers:** Variable and function names
- **Literals:** Numeric values, strings, characters

## 3. Symbol Table Generation

- Tracks all identifiers throughout the source code
- Records occurrence count for each identifier
- Maintains line number references
- Provides detailed visualization in a dedicated window

## 4. Modern User Interface

- **Dual-panel layout:** Source code editor (left) and token analysis viewer (right)
- **Syntax highlighting:** Color-coded display for different token types

- **Line numbering:** Integrated line counter for easy reference
- **Custom buttons:** Modern hover-effect buttons with gradient design
- **Status tracking:** Real-time token count and analysis status

### Technical Architecture

### Technologies Used

- **Language:** Python 3
- **GUI Framework:** Tkinter with ttk extensions
- **Pattern Matching:** Regular expressions (re module)
- **Additional Libraries:** ScrolledText, datetime, webbrowser

## Key Components

### Token Specification System:

- COMMENT: Single-line (//) and multi-line (/* */) comments

- HEADER: Preprocessor directives (#include)

- STRING/CHAR: Literal values with escape sequence support

- NUMBER: Integer and floating-point constants

- OPERATORS: All C operators with precedence handling

- SYMBOLS: Delimiters and separators

### Analysis Pipeline:

1. Source code input via text editor
2. Line-by-line tokenization using master regex pattern
3. Token classification based on type matching
4. Symbol table population for identifiers
5. Result display in TreeView with detailed attributes

# Functional Capabilities

## Analysis Operations

- **Analyze:** Performs complete lexical analysis and populates token table
- **Clear Source:** Resets the source code editor
- **Clear Tokenize:** Removes all tokens and resets symbol table
- **Sample Load:** Loads pre-configured C code example
- **Symbol Table View:** Opens dedicated window showing identifier statistics

## Developer Profile Integration

- Interactive developer information panel
- Contact information with clickable email and LinkedIn links
- Course and supervisor details
- Professional presentation with modern card design

## Output Information

The analyzer provides comprehensive output including:

- **Token Table:** Line number, token value, token type, column position
- **Statistics:** Total token count, identifier count
- **Symbol Table:** Identifier names, occurrence count, line references
- **Visual Feedback:** Color-coded syntax highlighting in source editor

## Use Cases

1. **Educational Tool:** Teaching lexical analysis concepts in compiler design
2. **Code Analysis:** Understanding token structure in C programs
3. **Debugging Aid:** Identifying tokenization issues in source code
4. **Learning Resource:** Demonstrating compiler frontend operations

## Development Approach:

- Object-oriented design with custom widget classes

- Event-driven architecture for user interactions
- Modular code structure for maintainability
- Comprehensive error handling and user feedback

## Future Enhancement Possibilities

- Support for additional programming languages
- Syntax error detection and reporting
- Export functionality for token tables
- Integration with parser for complete compilation pipeline
- Batch file processing capabilities
-

## Conclusion

This Modern Lexical Analyzer successfully demonstrates the fundamental phase of compilation through an intuitive, feature-rich interface. It serves as both a practical tool for code analysis and an educational resource for understanding tokenization processes in compiler design.