Design and Implementation of a Real-Time Syntax Highlighter for a Custom Programming Language

Bursa Technical University

Ahmet Furkan Öcel:
23360859729

Abstract

This paper presents the design and implementation of a real-time syntax highlighter for a custom mini-language called PCAL. The system consists of a lexical analyzer based on regular expressions and a top-down parser that together ensure accurate syntax analysis and highlighting. Unlike standard approaches, no external libraries were used for syntax highlighting. The project emphasizes clarity in parsing logic, extensibility of grammar rules, and responsive feedback through a Python-based graphical user interface (GUI). The proposed system highlights more than five token types and demonstrates robust performance in processing real-time user input.

---

1. Introduction

Syntax highlighting is a fundamental feature in modern development environments, aiding programmers in understanding and debugging code efficiently. This project explores how such a system can be implemented from scratch for a custom language, PCAL, which mimics basic programming constructs like variable assignment, arithmetic operations, and expression evaluation.

To meet academic and technical requirements, all components—lexer, parser, and GUI—were built without the use of third-party libraries for syntax highlighting. The resulting tool offers live feedback to users as they write PCAL code, enabling a more intuitive coding experience.

---

2. Related Work

Prior studies and tools (such as those integrated into IDEs like Visual Studio Code or Sublime Text) rely on mature libraries or language

servers for syntax highlighting. However, this project differentiates itself by manually implementing each step of the process, thereby contributing educational value in compiler design and user interface integration. Related academic works often discuss lexical analysis or parsing separately; this work unifies them under a GUI-driven real-time application.

---

3. Language Design and Grammar

PCAL supports:

- Variable declarations and assignments (e.g., `x = 5,`)
- Arithmetic operations (`+`, `-`, `*`, `/`)
- Grouping with parentheses
- Queries using the `?` operator

The grammar is designed to be simple yet sufficient for demonstrating parsing and highlighting techniques. Token types include:

- **Identifiers**
- **Numbers**
- **Operators**
- **Parentheses**
- **Delimiters**
- **Equals Sign**
- **Question Mark**

Each token is color-coded in the GUI for real-time feedback.

---

4. System Architecture

The application consists of three main modules:

- **Lexer:** Converts the raw input into a stream of tokens using regular expressions.

- **Parser:** A recursive descent (top-down) parser interprets the token stream and performs semantic actions like evaluating expressions and assigning values.
- **GUI:** Built using Python's `tkinter`, the GUI updates syntax highlighting in real time as users type.

These modules interact in a loop that processes every key event and re-renders the highlighted content immediately.

---

5. Implementation Details

- **Programming Language:** Python
- **Lexical Analysis:** Regular expressions compiled into a deterministic finite automaton
- **Parsing Approach:** Top-down recursive descent
- **GUI Toolkit:** `tkinter`
- **Highlighting:** Implemented via custom tags on text widgets for each token type

Special care was taken to optimize performance for real-time feedback by reducing redundant parsing operations and efficiently updating only the modified text regions.

---

6. Evaluation

The system was tested with numerous valid and invalid PCAL inputs. It correctly highlights more than five token types and evaluates expressions with variables and operators. Syntax errors are flagged, and the parser gracefully handles unexpected tokens with user-friendly messages.

---

7. Conclusion and Future Work

This paper demonstrated the viability of building a real-time syntax highlighter without external libraries. The integration of a hand-written lexer and parser with a responsive GUI presents a comprehensive learning example in language processing.

In future work, the system could be extended with support for more complex grammar, additional semantic analysis (e.g., type checking), and language server protocol (LSP) compatibility.

References

1. Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools*. Pearson Education.
2. Appel, A. W. (1998). *Modern Compiler Implementation in Java*. Cambridge University Press.
3. Reppy, J. (2007). *Lexing and Parsing*. University of Chicago, Lecture Notes.
4. Tkinter Documentation – https://docs.python.org/3/library/tkinter.html