Design and Implementation of a Structured Document Language

Abstract

This document presents the design and implementation of a structured document description language inspired by LaTeX but simplified for compiler construction and AST-based processing. The language focuses on hierarchical organization, deterministic parsing, and easy transformation into formats such as PDF, HTML, and Markdown.

Introduction

Modern document preparation systems are powerful but complex. This project explores a minimal alternative that preserves structure while remaining easy to parse and extend.

Motivation

Writing parsers for real-world languages is difficult due to ambiguity, complex syntax, and historical baggage.

- Simpler grammar enables faster parsing
- Hierarchical structure maps cleanly to ASTs
- Deterministic parsing reduces bugs

The motivation behind this language is to focus on *structure over presentation* and allow clean semantic processing. **This language is designed first as a compiler project.**

Goals

The primary goals of this project are listed below.

- Design an unambiguous grammar
- Build a clean hierarchical AST
- Enable easy backend transformations

These goals guide every design decision made in the language.

Language Design

The language is based on explicit structural markers and block-based scoping rules.

Core Constructs

The core constructs of the language include headings, subheadings, paragraphs, and inline formatting blocks.

- Headings define major sections
- Subheadings define subsections
- Paragraphs contain plain text

Inline formatting such as *italics* and **bold text** are expressed using explicit commands.

Grammar Simplicity

The grammar is intentionally restricted to avoid ambiguity.

- No implicit nesting
- Explicit block delimiters
- Predictable token stream

This makes the language ideal for LR parsing using tools such as Bison and Flex.

Implementation

The implementation follows a traditional compiler pipeline.

Lexical Analysis

The lexer converts the input document into a stream of tokens.

- Keywords such as @heading and @subheading
- Structural symbols like braces and parentheses
- Text blocks captured as TOKEN_TEXT

Lexical simplicity ensures robustness and clarity.

Parsing and AST Construction

The parser consumes tokens and constructs an abstract syntax tree.

- Each structural element becomes a Node
- Children represent hierarchical containment
- Order is preserved naturally

The AST mirrors the logical structure of the document rather than its surface syntax.

Semantic Processing

Once constructed, the AST enables semantic analysis and transformation.

- Validation of document structure
- Transformation to output formats
- Static analysis of document consistency

This design allows future extensions without modifying the grammar.

Applications

The language has several practical applications.

Academic Writing

Research papers benefit from strong structural guarantees.

- Clear section hierarchy
- Automated formatting
- Consistent output

This makes the language suitable for academic and technical writing.

Documentation Systems

Software documentation often requires structured content.

- API documentation
- Design specifications
- Developer guides

The AST-based approach enables automated documentation generation.

Conclusion

This project demonstrates that a well-designed grammar and AST can greatly simplify document processing. **The focus on structure enables powerful transformations.** Future work includes adding references, figures, and mathematical notation while preserving grammar simplicity.