From text to diagrams

- **Design and implementation of a lexer and parser for a custom diagram language.**

Introduction

- What is a lexer?
- What is a parser?
- Why are they important?
- Goal: Convert a diagram code to structured data

Background

- Domain-Specific Language(DSL)
- Simple syntax for defining diagrams

```
diagram flowchart {
  node A "Start"
  node B

  A -> B "Next"
}
```

The Lexer

- Scans input text
- Converts into tokens
- Recognises
    - Keywords
    - Identifiers
    - Symbols
    - Strings
    - Arrows

```
node A "Start"

TOKEN_KEYWORD: node


TOKEN_IDENTIFIER: A


TOKEN_STRING: Start
```

```
diagram flowchart (layout=vertical) {
      node A "Start" (color=lightgreen, shape=ellipse, text=black)
      node B "Process"
      node C "END"

      A -> B "Step 1" (color=red, width=3)
      B -> C "Step 2"
}
```

The Parser

- Consumes tokens
- Builds structure
  - Diagram -> Node + Edges
- Uses recursive descent

Syntax rules

- `diagram <type> { ... }`
- `node <id> "label"`
- `<id> -> <id> "label"`
- Attributes like:
  - `(color=blue, shape=rect)`

Attribute handling

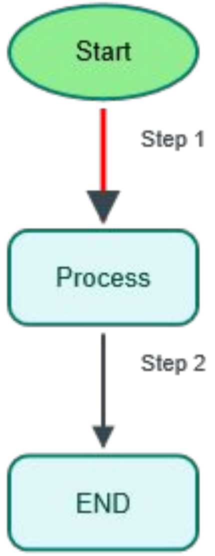
- Optional attributes
- Parentheses: `(key=value, ...)`
- Defaults applied when not present

Implementation highlights

- Unicode-aware lexer
- Simple token structure
- Default styling
- Descriptive errors
- Easy to extend

Conclusion

- Text input -> structured diagram
- Lexer and parser as core tools
- Useful for learning language processing
- Rendering


Start
Step 1
Process
Step 2
END