Team Hmm

Lexer Implementation

Imperative (I) Language

Team Members

Mikhail Trifonov

Kirill Efimovich

Technology Stack

Lexer Implementation Details

Component	Technology
Source Language	Imperative (I)
Implementation Language	Java
Lexical Analysis	Hand-written FSM
Token Position Tracking	Line/Column (1-based)
Target Integration	Bison-compatible tokens

Implementation Architecture

Core Components

- **TokenType Enum** - 43 distinct token types
- **Token Class** - Encapsulates type, lexeme, position
- **Lexer Class** - Main FSM implementation
- **LexerException** - Error reporting with context

Key Features

- ✅ Maximal munch principle - ✅ Unicode character support
- ✅ Position tracking - ✅ Comment handling (single-line `//`, multi-line `/* */`)

Finite State Machine Design

Scanner Implementation

```java
public Token nextToken() throws LexerException {
    while (!eofReached) {
        switch (currentChar) {
            case ':' -> { return scanColonOrAssign(); }
            case '.' -> { return scanDotOrRange(); }
            case '"' -> { return scanStringLiteral(); }
            // ... other cases
            default -> {
                if (Character.isLetter(currentChar))
                    return scanIdentifierOrKeyword();
                if (Character.isDigit(currentChar))
                    return scanNumberLiteral();
            }
        }
    }
}
```

Example 1: Variable Declaration

Source Code:

```
var x: integer is 42;
```

Recognized Tokens:

```
[0] VAR:var@1:1
[1] IDENTIFIER:x@1:5
[2] COLON::@1:6
[3] INTEGER:integer@1:8
[4] IS:is@1:16
[5] INTEGER_LITERAL:42@1:19
[6] SEMICOLON:;@1:21
[7] EOF:@1:22
```

Example 2: Range Operations

Source Code:

```
for i in 1..10 loop
    print i;
end
```

Recognized Tokens:

```
[0] FOR:for@1:1                                         [7] PRINT:print@2:5
[1] IDENTIFIER:i@1:5                                    [8] IDENTIFIER:i@2:11
[2] IN:in@1:7                                           [9] SEMICOLON:;@2:12
[3] INTEGER_LITERAL:1@1:10                              [10] END:end@3:1
[4] RANGE:..@1:11    ← Maximal munch: .. not . + .      [11] EOF:@3:4
[5] INTEGER_LITERAL:10@1:13
[6] LOOP:loop@1:16
```

Example 3: Comment Handling

Source Code:

```
var x: integer; // This is a comment
/* Multi-line
   comment block */
var y: real;
```

Recognized Tokens:

```
[0] VAR:var@1:1                                         [7] COLON::@3:6
[1] IDENTIFIER:x@1:5                                    [8] REAL:real@3:8
[2] COLON::@1:6                                         [9] SEMICOLON:;@3:12
[3] INTEGER:integer@1:8                                 [10] EOF:@3:13
[4] SEMICOLON:;@1:15
[5] VAR:var@3:1         ← Comments completely skipped
[6] IDENTIFIER:y@3:5
```

Error Handling & Position Tracking

Invalid Character Example:

```
var x @ 42;  // @ is invalid
```

Error Output:

```
LexerException: Invalid character: '@' (ASCII 64) at line 1, column 7
```

Questions?