# A Report on

## *LEXER IMPLEMENTATION USING ANTLR4 (SCANNER PART)*

**Submitted By:**

***Tanisha Majumder***
***Vaishak Balachandra***

**Under the Guidance of**

***Hairong Zhao, PhD***
Professor of Computer Science; Graduate Advisor

**Purdue University Northwest**
Department of Computer Science
2025-26

# *LEXER GENERATOR SELECTION REPORT*
# *(GROUP – 4)*

## *Selection: ANTLR4 (ANother Tool for Language Recognition) and Python*

After evaluating several lexer generator tools available for compiler construction, our team has selected ANTLR4 as our lexer generator. This report outlines the reasoning behind this decision and demonstrates the implementation of our lexer. Also, ANTLR4 has excellent support for Python as a target language. This makes it straightforward to integrate Python with ANTLR4's lexer generation process. The seamless integration enables us to leverage Python's features while using the powerful lexer generator provided by ANTLR4.

## *Justification for Choosing ANTLR4*

### 1. Cross-Language Support

ANTLR4 generates lexers in multiple target languages, including Java, C#, Python, JavaScript, and others. This flexibility allows us to integrate the lexer with various development environments and existing codebases, making it suitable for diverse project requirements.

### 2. Powerful Grammar Notation

ANTLR4 provides a clear and expressive grammar notation that closely resembles Extended Backus-Naur Form (EBNF). This makes it easier to specify complex token patterns and handle common lexical structures. The grammar syntax is intuitive for those familiar with formal language theory, reducing the learning curve.

### 3. Integrated Development Tools

ANTLR4 offers excellent integration with development environments through plugins for popular IDEs like IntelliJ IDEA, Eclipse, and Visual Studio Code. These plugins provide syntax highlighting, error checking, and visualization tools that significantly improve productivity during grammar development.

### 4. Error Recovery Mechanisms

ANTLR4 includes sophisticated error recovery mechanisms that allow the lexer to continue processing input even after encountering invalid tokens. This feature is crucial for providing comprehensive error messages during compilation, enhancing the debugging experience for end users.

### 5. Active Community and Documentation

ANTLR4 has a large and active community of users, providing extensive documentation, examples, and third-party resources. The tool is regularly updated and maintained by its creators, ensuring compatibility with modern programming practices and addressing bugs promptly.

### 6. Lexer-Parser Integration

While our current focus is on the lexer component, ANTLR4's seamless integration between lexer and parser components will be beneficial for future expansion of our compiler project. The tool maintains consistency in notation and methodology across both components, simplifying the development process.

### 7. Performance Considerations

ANTLR4 generates efficient lexers that use a deterministic finite automaton (DFA) approach for token recognition. This results in linear-time lexical analysis, which is essential for processing large input files with minimal performance overhead.

### 8. Support for Custom Actions

ANTLR4 allows embedding custom code actions within the grammar, enabling fine-grained control over token processing. This feature facilitates implementing special handling for complex tokens or context-sensitive lexical patterns.

## *Conclusion*

ANTLR4 with Python provides the best balance of flexibility, power, ease of use, and integration capabilities for our lexer implementation. Its widespread adoption in both academic and industry settings further validates its suitability for compiler construction projects. The tool's robust feature set and excellent documentation make it an ideal choice for implementing our lexical analyzer.