Republic of the Philippines
CAMARINES SUR POLYTECHNIC COLLEGES
Nabua, Camarines Sur
ISO 9001:2015

COLLEGE *of* COMPUTER STUDIES

Lexical Analyzer Using Flex

A Project Report in Programming Languages

Submitted To:

GULANE, Elena Clarexe

Submitted By:

BAZAR, Jayp S.

NANALE, Krizia Belle L.

PARAÑAL, Mary Rachel L.

BSCS 3A

I. Project Description

a. Overview

The Multi-language Lexical Analyzer is a software tool designed to perform lexical analysis on source code written in C, Java, and Python. Lexical analysis, the first phase of a compiler, involves breaking down source code into a sequence of tokens such as keywords, identifiers, literals, operators, and symbols. This project leverages Flex, a lexical analyzer generator, to define token patterns for each supported language and integrates a Python-based user interface to facilitate language selection and input processing. The analyzer accepts input either through the terminal or from a text file and outputs categorized tokens, serving as a foundation for further compiler stages or code analysis. The tool aims to demonstrate the principles of lexical analysis and provide a practical utility for educational purposes in compiler design.

b. Features

The Multi-language Lexical Analyzer offers a range of functionalities that demonstrates the core principles of lexical analysis while supporting different programming languages.

- **Multi-language support**: Allow analysis of source code written in C, Java, and Python.
- **Dual input modes**: Users can input code directly through the terminal or load it from a text file.
- **Token classification**: Identifies and categorizes tokens into:
  - **Keywords** - Reserved words such as int, def, public
  - **Identifiers** - Variable and function names like main, sum
  - **Literals** - Numeric values like 123, 3.14
  - **Operators** - Symbols such as +, =, ==
  - **Delimiters** - Syntax-related characters like ;, {, }
- **Error detection and reporting**: Clearly flags unrecognized or invalid tokens.

II. System Requirements

a. Software Requirements

The following software must be installed and accessible from the command line (i.e., added to your system PATH):

- Python 3.10 or higher
- GCC compiler (for compiling C code, e.g., via MinGW, Cygwin, or WSL)
- Flex lexical analyzer generator
- Windows OS (the program uses 'cls' and expects 'lexer.exe)

b. Hardware Requirements

The project can run with the following minimum specifications:

- Operating System: Windows
- Processor: 1GHx or faster
- RAM: 2GB or more
- Storage: At least 500MB of free disk space for software installation and project files.

III. Implementation Details

The Multi-language Lexical Analyzer is implemented using a combination of Flex for token pattern definition and Python for user interaction and process automation. The system comprises the following components:

Flex Files:

***'c_lang.l'*:** Defines token patterns for C, including keywords, identifiers, literals, operators, symbols, preprocessor directives, single-line comments, and escape sequences.

'java_lang.l': Similar to c_lang.l, but tailored for Java with keywords, support for multi-line comments, and Java-specific operators.

***'python_lang.l'**:* Defines Python-specific tokens, including keywords, identifiers, literals, operators, symbols, single-line comments, and escape sequences.

Each Flex file uses regular expressions to match tokens and a 'print_token' function to format output.

Main Program (*'main.py'*)

- Provides a menu-driven interface where users select language (C, Java, or Python) or exit.
- Uses the subprocess module to run Flex on the selected .l file, generating lex.yy.c.
- Compiles lex.yy.c using GCC to produce lexer.exe.
- Prompts for input type (terminal or file) and processes the input through lexer.exe, displaying the token output.
- Clears the screen between runs using cls and pauses for user confirmation.

Token Processing:

The Flex generated lexer scans the input, matches it against defined patterns, and outputs tokens with their categories and subtypes (if applicable). Unrecognized tokens trigger an error message.

The multi-language capability is achieved by maintaining separate '.l' files for each language, with 'main.py' dynamically selecting the appropriate file based on user input. The token patterns are defined using regular expressions in Flex, ensuring accurate identification of language-specific syntax elements.

IV. Testing and Results

To verify the functionality and robustness of the Multi-language Lexical Analyzer, the system was tested with various valid and invalid code inputs across all supported programming languages (C, Java, and Python). Testing involved

both terminal-based and file-based input modes to ensure that token classification, error handling, and user interaction worked as expected.

1. **Functional Testing with Valid Inputs**

   These tests were designed to ensure that the lexical analyzer correctly identifies and classifies valid tokens.

   C Language:

```
Choose a language:
  1. C
  2. Java
  3. Python
  4. Exit
Enter your choice (1-4): 1
Compilation successful. Executable: lexer.exe
Choose input type:
1. Terminal input
2. Text file input
Enter your choice (1-2): 1
Enter the input for the lexer (end with Enter):
int main() { float x = 5.0; }
```

Input

```
Lexer Output:
 KEYWORD: int
IDENTIFIER: main
DELIMITER: (
DELIMITER: )
DELIMITER: {
KEYWORD: float
IDENTIFIER: x
OPERATOR (assignment): =
LITERAL (float): 5.0
DELIMITER: ;
DELIMITER: }

Press Enter to continue...
```

Output

Java Language:

```
C:\Windows\System32\cmd.e
Choose a language:
  1. C
  2. Java
  3. Python
  4. Exit
Enter your choice (1-4): 2
Compilation successful. Executable: lexer.exe
Choose input type:
1. Terminal input
2. Text file input
Enter your choice (1-2): 1
Enter the input for the lexer (end with Enter):
public class HelloWorld { public static void main(String[] args) { int a = 10; int b = 5; int sum = a + b; System.out.println("Sum is: " + sum); }}
```

Input

```
Lexer Output:
 KEYWORD: public
KEYWORD: class
IDENTIFIER: HelloWorld
SYMBOL: {
KEYWORD: public
KEYWORD: static
KEYWORD: void
IDENTIFIER: main
SYMBOL: (
IDENTIFIER: String
SYMBOL: [
SYMBOL: ]
IDENTIFIER: args
SYMBOL: )
SYMBOL: {
KEYWORD: int
IDENTIFIER: a
OPERATOR (assignment): =
LITERAL (integer): 10
SYMBOL: ;
KEYWORD: int
IDENTIFIER: b
OPERATOR (assignment): =
LITERAL (integer): 5
SYMBOL: ;
KEYWORD: int
IDENTIFIER: sum
OPERATOR (assignment): =
IDENTIFIER: a
OPERATOR (arithmetic): +
IDENTIFIER: b
SYMBOL: ;
IDENTIFIER: System
SYMBOL: .
IDENTIFIER: out
SYMBOL: .
IDENTIFIER: println
SYMBOL: (
LITERAL (string): "Sum is: "
OPERATOR (arithmetic): +
IDENTIFIER: sum
SYMBOL: )
SYMBOL: ;
SYMBOL: }
SYMBOL: }

Press Enter to continue...
```

Output

Python Language:

```
C:\Windows\System32\cmd.e
Choose a language:
  1. C
  2. Java
  3. Python
  4. Exit
Enter your choice (1-4): 3
Compilation successful. Executable: lexer.exe
Choose input type:
1. Terminal input
2. Text file input
Enter your choice (1-2): 2
Enter the path to the input text file (with file extension): test.py
```

Edit Selection View Go Run Terminal Help

test.py

C: > Users > Rachel > OneDrive > Desktop > Multi-language_Lexical_A

```
1  def calculate_area(length, width):
2      area = length * width
3      return area
4
5  length = float(input("Enter the length: "))
6  width = float(input("Enter the width: "))
7  result = calculate_area(length, width)
8  print("The area is:", result)
9
```

Input

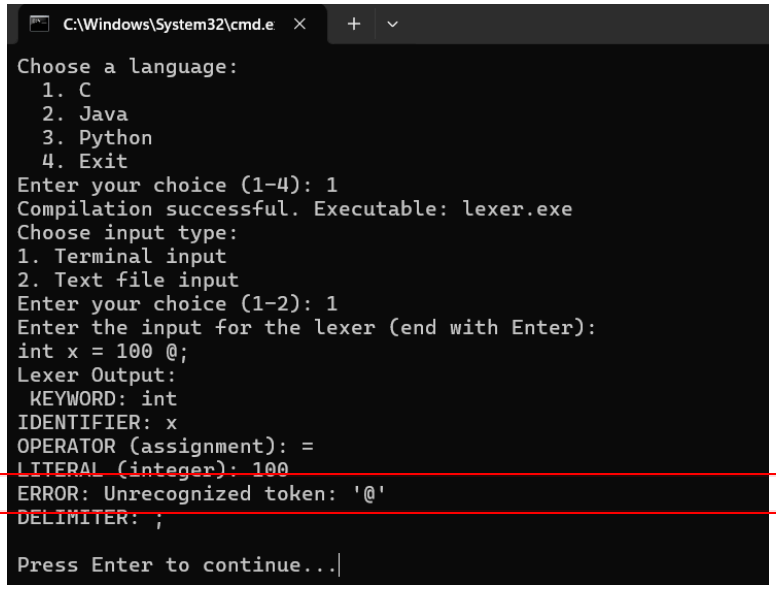
```
Lexer Output:
 KEYWORD: def
IDENTIFIER: calculate_area
DELIMITER: (
IDENTIFIER: length
DELIMITER: ,
IDENTIFIER: width
DELIMITER: )
DELIMITER: :
IDENTIFIER: area
OPERATOR (assignment): =
IDENTIFIER: length
OPERATOR (arithmetic): *
IDENTIFIER: width
KEYWORD: return
IDENTIFIER: area
IDENTIFIER: length
OPERATOR (assignment): =
IDENTIFIER: float
DELIMITER: (
IDENTIFIER: input
DELIMITER: (
LITERAL (string): "Enter the length: "
DELIMITER: )
DELIMITER: )
IDENTIFIER: width
OPERATOR (assignment): =
IDENTIFIER: float
DELIMITER: (
IDENTIFIER: input
DELIMITER: (
LITERAL (string): "Enter the width: "
DELIMITER: )
DELIMITER: )
IDENTIFIER: result
OPERATOR (assignment): =
IDENTIFIER: calculate_area
DELIMITER: (
IDENTIFIER: length
DELIMITER: ,
IDENTIFIER: width
DELIMITER: )
IDENTIFIER: print
DELIMITER: (
LITERAL (string): "The area is:"
DELIMITER: ,
IDENTIFIER: result
DELIMITER: )

Press Enter to continue...
```

Output

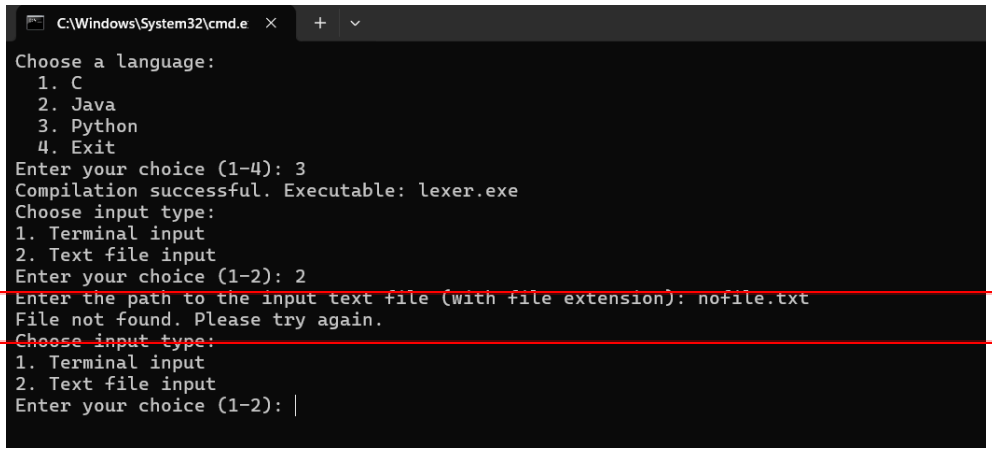
2. Error Handling Tests

To evaluate the robustness of the program, inputs with invalid characters and file-related issues were tested.

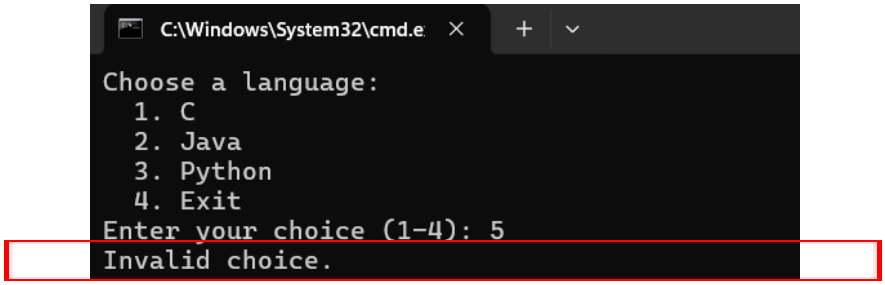
Unrecognized Character Test



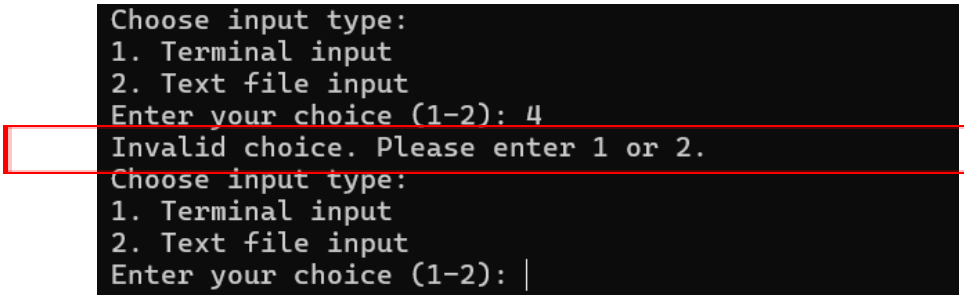
Invalid File Name Test



Invalid Menu Choice



Invalid Input Type



V. Demo Instructions

To run the Multi-language Lexical Analyzer, follow these steps:

1. Clone the repository:

git clone
https://github.com/jaypbazar/Multi-language_Lexical_Analyzer.git

cd Multi-language_Lexical_Analyzer

2. Ensure Python 3.10+, GCC, and Flex are installed and added to the system PATH.

3. Open a terminal in the project directory.

4. Run the main program:

python main.py

5. Follow the prompts to:

- Select a language (1 for C, 2 for Java, 3 for Python, 4 to exit).

- Choose input type (1 for terminal, 2 for text file).
- Provide input (e.g., type code for terminal input or specify a file like input.txt).

6. View the lexer output in the terminal.

A video demonstration is available at: https://youtu.be/auXbGWaaHXU

VI. Error Handling and Edge Cases

The program is designed to handle the following errors and edge cases:

Invalid Language Choice: If the user enters a number outside 1–4, 'main.py' displays ***"Invalid choice"*** and prompts again.

```
Choose a language:
  1. C
  2. Java
  3. Python
  4. Exit
Enter your choice (1-4): Python
Invalid choice.
```

Invalid Input Type: If the user selects an input type other than 1 or 2, 'main.py' displays, ***"Invalid choice. Please enter 1 or 2"*** and prompt again.

```
Choose input type:
1. Terminal input
2. Text file input
Enter your choice (1-2): 3
Invalid choice. Please enter 1 or 2.
```

File Not Found: If the specified text file does not exist, main.py catches the FileNotFoundError and displays ***"File not found. Please try again."***

```
Choose input type:
1. Terminal input
2. Text file input
Enter your choice (1-2): 2
Enter the path to the input text file: qw
File not found. Please try again.
```

Unrecognized Tokens: The Flex files (c_lang.l, java_lang.l, python_lang.l) include a catch-all rule (.) to report unrecognized tokens with an error message, e.g., ERROR: Unrecognized token '@'.

Compilation Errors: If Flex or GCC fails (e.g., due to missing dependencies), the program will not proceed.

VII. Conclusion

This project successfully demonstrates the use of Flex in implementing a multi-language lexical analyzer that supports C, Java, and Python. Through its structured design and error-handling capabilities, it illustrates the principles of the lexical analysis phase of compiler design.

The key accomplishments of this project include:

- Correct implementation of token patterns for three major programming languages
- Dynamic language selection and compilation through Python
- Handling of both terminal and file-based inputs
- Clear and categorized output display
- Robust error handling and user guidance

Challenges Encountered:

- Managing different language syntax rules in Flex
- Handling file system issues and user errors gracefully

Suggestions for Future Improvements:

- Add support for additional languages such as JavaScript or C++
- Integrate a graphical user interface for better usability
- Implement syntax highlighting in the output

VIII. *Recommendations*

To improve the tool and expand its capability, we propose the following recommendations:

- **Add GUI Support**: Implement a graphical interface to make the tool more accessible to non-technical users.
- **Introduce Token Highlighting**: Enhance output readability by applying syntax color schemes.
- **Build Cross-Platform Support**: Refactor commands and dependencies to support both Windows and Linux systems natively.

IX. References

[1] Phases of a compiler: 2025. Retrieved from *https://www.geeksforgeeks.org/phases-of-a-compiler/*.

[2] Introduction of lexical analysis: 2025. Retrieved from *https://www.geeksforgeeks.org/introduction-of-lexical-analysis/*.