A

Compiler Design Lab Project

on

Code Language Detector

Submitted to Manipal University, Jaipur

Towards the partial fulfillment of the Award of the Degree of

BACHELOR OF TECHNOLOGY

In Information Technology

2024-2025

By

Lakshya Pawar

229302177

Ayonija

229302173

Navodit Kapoor

229302204

MANIPAL UNIVERSITY JAIPUR

Under the guidance of

Mr. Virender Dehru

Department of Information Technology

School of Computer Science

Manipal University Jaipur

Jaipur, Rajasthan

1. Introduction

With the proliferation of programming languages and open-source code repositories, identifying the programming language of a code snippet is increasingly vital for tasks like syntax highlighting, code analysis, and automated documentation. The Code Language Detector addresses this need by automatically classifying code as languages such as Python, C, C++, or Java using compiler design principles like lexical analysis and pattern recognition. This project bridges theoretical compiler concepts with practical applications, enabling efficient code management in real-world scenarios like IDEs and code-sharing platforms.

---

2. Problem Definition and Methodology as Proposed Solution

Problem Definition

The diversity of programming languages and the absence of explicit language indicators in code snippets pose challenges for manual identification, which is time-consuming and error-prone, especially when languages share similar syntax. This complicates tasks like syntax highlighting, automated compiling, and intelligent code analysis. The objective is to develop a Code Language Detector that accurately identifies the programming language of a given code snippet using compiler front-end techniques.

Methodology

The proposed solution simulates a compiler's lexical analysis phase to tokenize code and classify it based on language-specific features. The methodology includes the following steps:

1. **Input Acquisition**
   - Accept raw code snippets via user input or a file.
   - Restrict detection to Python, C, C++, and Java for focused analysis.
2. **Pre-processing**
   - Remove comments, extra whitespaces, and blank lines.
   - Normalize indentation to ensure uniform formatting.
3. **Lexical Analysis (Tokenization)**
   - Use Lex to define regular expressions for tokenizing keywords (e.g., `def`, `#include`), identifiers, operators, and special characters.
   - Break code into atomic tokens for further analysis.

4. **Feature Extraction**

   - Identify language-specific patterns, such as:
     - File headers (`#include` for C, `import` for Python).
     - Keywords (`public` for Java, `std::` for C++).
     - Structural conventions (indentation for Python, semicolons for C/Java).
   - These features differentiate similar languages.

5. **Language Classification**

   - Use a rule-based approach in Yacc to count language-specific keywords.
   - Assign the language with the highest keyword count, prioritizing Python, Java, C++, then C.

6. **Output Generation**

   - Display the detected language or indicate if detection fails.

7. **Testing and Evaluation**

   - Test with diverse code snippets across supported languages.
   - Evaluate accuracy and refine rules for ambiguous cases.

---

3. Implementation

File Name: lang_detector.l

```
%{
#include "y.tab.h"
%}

%%
"#include"          { return C_KEYWORD; }
"printf"            { return C_KEYWORD; }
"scanf"             { return C_KEYWORD; }

"iostream"          { return CPP_KEYWORD; }
"std::"             { return CPP_KEYWORD; }
"cin"|"cout"        { return CPP_KEYWORD; }

"public"|"class"|"static"|"void"|"main" { return JAVA_KEYWORD; }
"System.out.println"    { return JAVA_KEYWORD; }

"def"|"import"|"print"|"self"|"elif" { return PYTHON_KEYWORD; }

[a-zA-Z_][a-zA-Z0-9_]*    ;
[ \t\n\r]+                ;
```

```
.                       ;
%%

int yywrap() {
   return 1;
}
```

File Name: lang_detector.y

```
%{
#include <stdio.h>
#include <string.h>

int c_keywords = 0, cpp_keywords = 0, java_keywords = 0, python_keywords = 0;

void yyerror(const char *s) {}
int yylex(void);
%}

%token C_KEYWORD CPP_KEYWORD JAVA_KEYWORD PYTHON_KEYWORD

%%

input:
   /* empty */
   | input language_element
   ;

language_element:
     C_KEYWORD        { c_keywords++; }
   | CPP_KEYWORD      { cpp_keywords++; }
   | JAVA_KEYWORD     { java_keywords++; }
   | PYTHON_KEYWORD   { python_keywords++; }
   ;

%%

int main() {
   yyparse();

   if (python_keywords > java_keywords && python_keywords > cpp_keywords &&
python_keywords > c_keywords)
      printf("Detected Language: Python\n");
   else if (java_keywords > cpp_keywords && java_keywords > c_keywords)
      printf("Detected Language: Java\n");
   else if (cpp_keywords > c_keywords)
      printf("Detected Language: C++\n");
   else if (c_keywords > 0)
      printf("Detected Language: C\n");
```

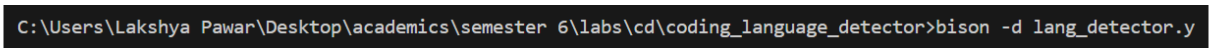
```
    else
        printf("Language could not be determined.\n");

    return 0;
}
```

---

4. Results

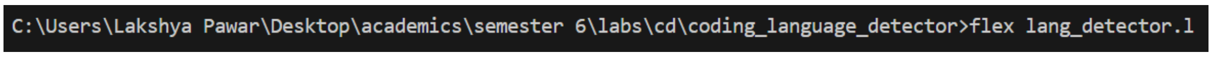
Commands Used for Execution

bison -d lang_detector.y [Generates y.tab.c and y. tab.h]



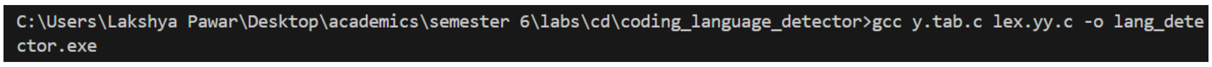
```
C:\Users\Lakshya Pawar\Desktop\academics\semester 6\labs\cd\coding_language_detector>bison -d lang_detector.y
```

flex lang_detector.l [Generates lex.yy.c]



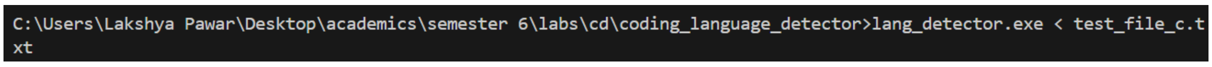
```
C:\Users\Lakshya Pawar\Desktop\academics\semester 6\labs\cd\coding_language_detector>flex lang_detector.l
```

gcc y.tab.c lex.yy.c -o lang_detector.exe [Compile]



```
C:\Users\Lakshya Pawar\Desktop\academics\semester 6\labs\cd\coding_language_detector>gcc y.tab.c lex.yy.c -o lang_dete
ctor.exe
```

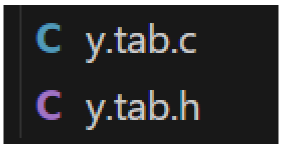
./lang_detector.exe < test_file_c.txt [Run]



```
C:\Users\Lakshya Pawar\Desktop\academics\semester 6\labs\cd\coding_language_detector>lang_detector.exe < test_file_c.t
xt
```

Step 1:

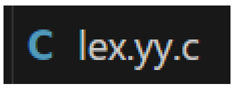
Description: Generated parser files using Bison.
Output: Produced `y.tab.c` and `y.tab.h`



y.tab.c
y.tab.h

Step 2:

Description: Generated lexer file using Flex.
Output: Produced `lex.yy.c`.



lex.yy.c

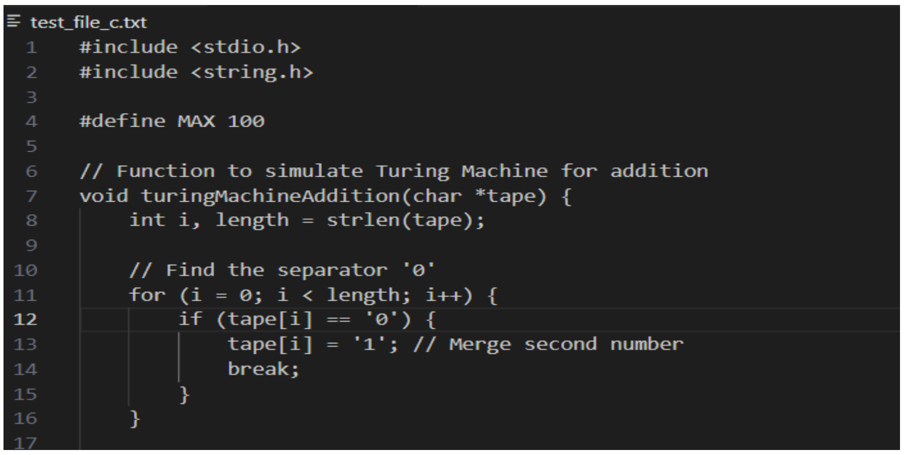
Step 3:

Description: Compiled the parser and lexer files to create the executable.
Output: Produced `lang_detector.exe`.

```
lang_detector.exe
```

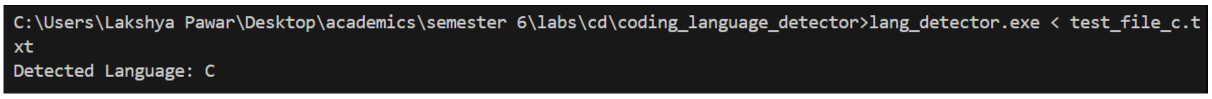
Step 4:

Description: Tested with a C code snippet containing `#include <stdio.h>`, `printf`, and other C-specific keywords.
Output: `Detected Language: C`



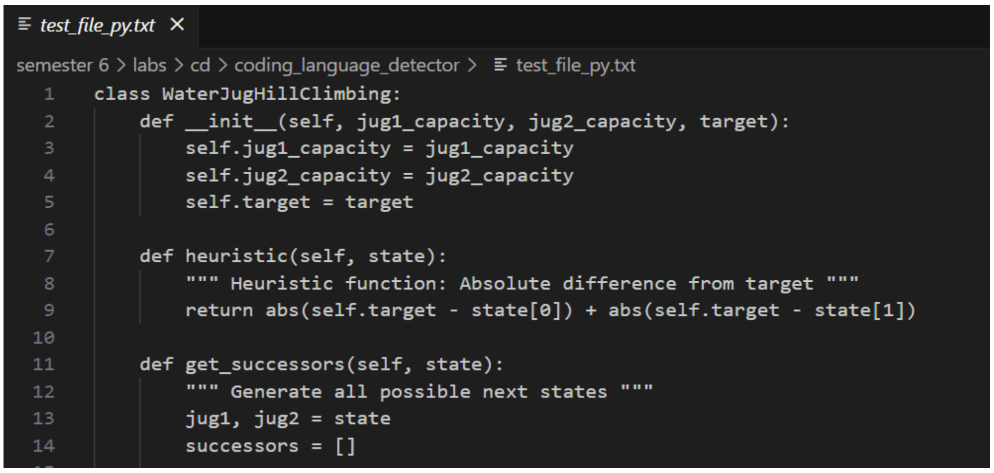
test_file_c.txt

```c
#include <stdio.h>
#include <string.h>

#define MAX 100

// Function to simulate Turing Machine for addition
void turingMachineAddition(char *tape) {
    int i, length = strlen(tape);

    // Find the separator '0'
    for (i = 0; i < length; i++) {
        if (tape[i] == '0') {
            tape[i] = '1'; // Merge second number
            break;
        }
    }
```



```
C:\Users\Lakshya Pawar\Desktop\academics\semester 6\labs\cd\coding_language_detector>lang_detector.exe < test_file_c.t
xt
Detected Language: C
```

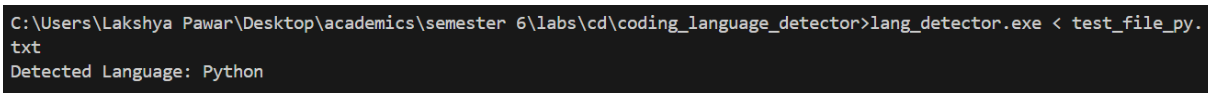
Other Test Cases



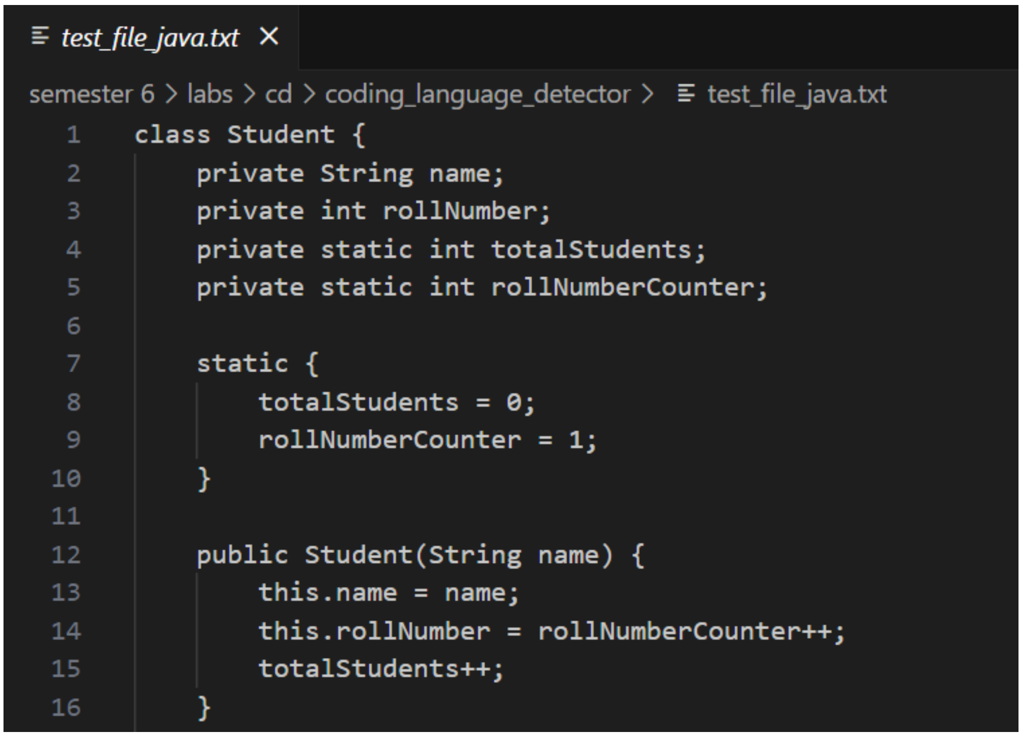
test_file_py.txt ×

semester 6 > labs > cd > coding_language_detector > test_file_py.txt

```python
class WaterJugHillClimbing:
    def __init__(self, jug1_capacity, jug2_capacity, target):
        self.jug1_capacity = jug1_capacity
        self.jug2_capacity = jug2_capacity
        self.target = target

    def heuristic(self, state):
        """ Heuristic function: Absolute difference from target """
        return abs(self.target - state[0]) + abs(self.target - state[1])

    def get_successors(self, state):
        """ Generate all possible next states """
        jug1, jug2 = state
        successors = []
```

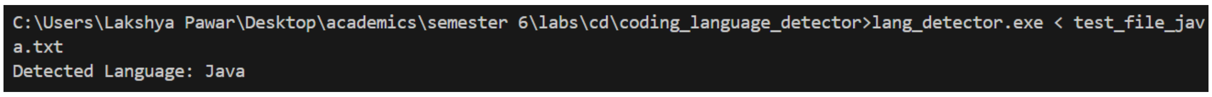


```
C:\Users\Lakshya Pawar\Desktop\academics\semester 6\labs\cd\coding_language_detector>lang_detector.exe < test_file_py.
txt
Detected Language: Python
```



test_file_java.txt
semester 6 > labs > cd > coding_language_detector > test_file_java.txt

```
class Student {
    private String name;
    private int rollNumber;
    private static int totalStudents;
    private static int rollNumberCounter;

    static {
        totalStudents = 0;
        rollNumberCounter = 1;
    }

    public Student(String name) {
        this.name = name;
        this.rollNumber = rollNumberCounter++;
        totalStudents++;
    }
```



```
C:\Users\Lakshya Pawar\Desktop\academics\semester 6\labs\cd\coding_language_detector>lang_detector.exe < test_file_jav
a.txt
Detected Language: Java
```

---

5. Conclusion and Future Work

The Code Language Detector effectively identifies programming languages like Python, C, C++, and Java by leveraging Lex for tokenization and Yacc for rule-based classification, demonstrating the power of compiler design principles in practical applications. The system achieves reliable detection for clear code snippets but may struggle with mixed-language or highly ambiguous inputs. Future enhancements could include support for additional languages, integration of machine learning for improved accuracy, handling edge cases like code fragments, and developing a user-friendly GUI for integration into IDEs or code editors.

Bibliography/References

- GeeksforGeeks - Articles and tutorials on Lex and compiler design concepts.
- Stack Overflow - Community discussions for debugging and implementation support.
- YouTube Tutorials - Lex programming basics and examples.
- Perplexity - Debugging and Execution.