202100377 – Bhavin Baldota
202100381 – Bramha Nimbalkar

Phase 1: Lexical Analysis

Project Overview

The System Programming Project endeavors to create a robust tool using Python with Tkinter for a Graphical User Interface (GUI) that facilitates a thorough analysis of C programming code. Phase 1 of the project focuses on executing a meticulous lexical analysis, ensuring the structural integrity and syntactical correctness of the provided C code.

System Functionalities

1. **Header Validation**: The system conducts a rigorous examination to verify the accuracy and inclusion of header files within the C code. It ensures that specified header files are correctly referenced and utilized within the code structure.

```
# Header Validation
with open('input.c', 'r') as file:
    content = file.read()
    for line in content.split('\n'):
        if line.startswith("#include"):
            header_file = line.split()[1].strip('\"')
            if header_file not in header:
                messagebox.showerror("Syntax Error", "Invalid Header File")
                raise SyntaxError("Invalid Header File")
```

2. **Function Verification**: The system rigorously checks for the presence and correctness of the **main()** function, crucial in C programming. It ensures the appropriate use of function types to adhere to programming standards.

```
# Function Verification
ad_c_code = [line.split() for line in c_code if not line.startswith('#')]
if ad_c_code[0][0] != 'int':
    messagebox.showerror("Syntax Error", "Invalid Function Type")
    raise SyntaxError("Invalid Function Type")
if ad_c_code[0][1] != 'main()':
    messagebox.showerror("Syntax Error", "No main() function present")
    raise SyntaxError("No main() function present")
```

3. **Bracket Balance Check**: An essential part of the analysis, it ensures a balanced count of opening and closing brackets within the code.

```
# Bracket Balance Check
if (content.count('(') != content.count(')')) or (content.count('{') !=
content.count('}')):
    messagebox.showerror("Syntax Error", "Unbalanced Brackets")
    raise SyntaxError("Unbalanced Brackets")
```

4. **Symbol Table Creation**: The tool generates a comprehensive symbol table, capturing declared variables and their respective data types

```
# Symbol Table Creation
symbol_table = {}
for line in ad_c_code:
    if line[0] in types:
        current_type = line[0]
        for sym in line[1:]:
            symbol_table[sym] = current_type
```

5. **Printf Statement Processing**: The system meticulously inspects **printf** statements, validating their syntax and content, ensuring the correct format and data type usage.

```
# Printf Statement Processing
for line in ad_c_code:
    if line[0].startswith('printf'):
        opening_index = line[0].find('(')
        closing_index = line[0].find(')')
        if opening_index != -1 and closing_index != -1:
            content_within_parentheses = line[0][opening_index + 1: closing_index]
            # Additional processing logic for content within parentheses
            # Check and validate syntax within printf statements
        else:
            messagebox.showerror("Syntax Error", "Invalid Printf Statement")
            raise SyntaxError("Invalid Printf Statement")
```

Execution and Conclusion

The system executes a comprehensive analysis of the provided C code, meticulously verifying headers, function types, bracket balance, symbol declarations, and **printf** statement syntax. The output provides a detailed report displaying the analyzed code, potential errors detected, and processed **printf** statement results. This analysis not only aids in error identification but also enhances developers' understanding of code structure and adherence to coding standards.

The comprehensive tool serves as an essential asset for developers, aiding in error identification, providing a detailed report for debugging, and ensuring adherence to best coding practices.