202100377 – Bhavin Baldota
202100381 – Bramha Nimbalkar

Phase 3: Execution

Project Overview

Phase 3 of the System Programming Project encompasses the execution process, integrating the Python-based tool with Tkinter to perform a comprehensive analysis and execution of the provided C programming code. This phase executes the prepared code, performs operations, and generates outputs to provide users with a comprehensive overview of the analyzed code.

Functionality and Execution Process

1. **Code Execution and Function Operations**: The system processes the C code, running operations to validate headers, function types, bracket balance, and symbol declarations.

```
# 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")


# 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")


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

2. **Symbol Table Generation and Printf Statement Analysis**: The tool generates a symbol table, recording declared variables and their respective data types. It analyzes **printf** statements, ensuring correct syntax and 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


# 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")
```

3. **Output Generation and Error Identification**: The system produces outputs displaying the analyzed code, potential errors, and processed **printf** statement results for user reference.

```
    split_printf_contents = []

    for line in printf_contents:
        lin = line[0].split()
        split_printf_contents.append(lin)

    split_printf_contents
    opreations = ['+', '-' , '*', '/']
    store_val = []
    flag_check1 = 0
    flag_check2 = 0

    def get_symdol_value(sym):
        for line in final_symbol:
            for i in range(len(line)):
                if line[i] == sym and len(line) == 3:
                    return line[i+1]


    for line in split_printf_contents:
        flag_check1 = 0
```

```
        flag_check2 = 0
        store_val.clear()
        for i in range(len(line)):
            if line[i] in only_symbol:
                sym = line[i]
                val = get_symdol_value(sym)
                store_val.append(val)
        for j in range(len(line)):
            if line[j] in datatype_check:
                flag_check1 = 1
                for k in range(len(line)):
                    if line[k] in opreations:
                        flag_check2 = 1
                        if line[k] == '+':
                            print(int(store_val[0]) + int(store_val[1]))
                            output_text.append([str(int(store_val[0]) +
int(store_val[1]))])
                        if line[k] == '-':
                            print(int(store_val[0]) - int(store_val[1]))
                            output_text.append([str(int(store_val[0]) -
int(store_val[1]))])
                        if line[k] == '*':
                            print(int(store_val[0]) * int(store_val[1]))
                            output_text.append([str(int(store_val[0]) *
int(store_val[1]))])
                        if line[k] == '/':
                            print(int(store_val[0]) / int(store_val[1]))
                            output_text.append([str(int(store_val[0]) /
int(store_val[1]))])
                if flag_check2 == 0:
                    print(val)
                    output_text.append([val])
        if flag_check1 == 0:
            messagebox.showerror("Syntax Error", "Data type is Missing")
            raise SyntaxError("Data type is Missing")
```

Conclusion and Output Details

Phase 3's execution process offers a comprehensive analysis of the provided C code. It executes operations to verify headers, function types, bracket balance, and symbol declarations. The generated output provides users with a detailed report displaying the analyzed code, identified errors, and processed **printf** statement results.

The phase not only aids in the identification of errors but also enhances the users' understanding of code structure and adherence to coding standards. It provides an essential tool for developers, supporting the identification and rectification of potential errors while ensuring compliance with best coding practices.

C Compiler

```c
#include <stdio.h>
#include <string.h>
#define Max 10
void main()
{
    int a, b;
    float c, d;
    a = 10;
    printf ("%d" , a);
    b = 20;
    printf ("%d" , a + b);
}
```

Lexical Analysis

```
' <stdio.h> ' is the header file used in the code
' <string.h> ' is the header file used in the code
' Max ' is the constant used in the code with ' 10 ' as its value

Type of function used is ' void '
Name of the function is ' main() '

' int ' type variable named ' a ' with value ' 10 ' is declared int the code
' int ' type variable named ' b ' with value ' 20 ' is declared int the code
' float ' type variable named ' c '
' float ' type variable named ' d '
```

Output

```
10
30
```