Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi

CS 424 (Compiler Construction Lab)

<u>Lab Project</u>

Submitted by

Zartaj Asim 2020526

C-Like Compiler Implementation Overview

Introduction:

The C-Like Compiler project aims to implement a simplified compiler for a C-like language using Python. The compiler consists of a lexer, a parser, and a graphical user interface (GUI) to facilitate code input, syntax checking, and execution.

1. Lexer Implementation:

- The lexer is responsible for converting input code into tokens, which are the smallest units of meaningful language constructs.
- We defined lexical elements such as keywords, identifiers, literals, operators, and punctuation symbols.
- Regular expressions were used to match and tokenize different elements of the input code based on predefined token types.
- The lexer iterates over the input code and identifies tokens according to the specified grammar rules.

2. Grammar Defined:

- program -> declaration_list
- declaration_list -> declaration | declaration_list declaration
- declaration -> variable_declaration | function_declaration
- variable_declaration -> type_specifier ID ;
- function_declaration -> type_specifier ID (parameters) compound_statement
- parameters -> parameter_list | VOID
- parameter_list -> parameter | parameter_list , parameter
- parameter -> type_specifier ID
- compound_statement -> { statement_list }
- statement_list -> statement | statement_list statement
- statement -> expression_statement | compound_statement | selection_statement | iteration_statement | return_statement
- expression_statement -> expression ;
- selection_statement -> IF (expression) statement | IF (expression) statement ELSE statement

- iteration_statement -> WHILE (expression) statement
- return_statement -> RETURN expression ;
- expression -> variable = expression | simple_expression
- variable -> ID
- simple_expression -> additive_expression relop additive_expression | additive_expression
- additive_expression -> additive_expression addop term | term
- term -> term mulop factor | factor
- factor -> (expression) | variable | call | NUM
- call -> ID (args)
- args -> arg_list | ε
- arg_list -> expression | arg_list , expression

3. Parser Implementation:

- The parser processes the tokens generated by the lexer and constructs a parse tree based on the grammar rules.
- We defined grammar rules for declarations, statements, expressions, and control flow constructs.
- Recursive descent parsing technique was employed to recursively parse the input code and generate a hierarchical representation of its syntactic structure.
- Error handling mechanisms were implemented to detect syntax errors and provide meaningful error messages to the user.

4. Sample Input Code:

```
int main() {

int x = 10;

float y = 3.14;

if (x > 0) {

y = y + 1.0;

}
return 0;

}
```

Generated Tokens:

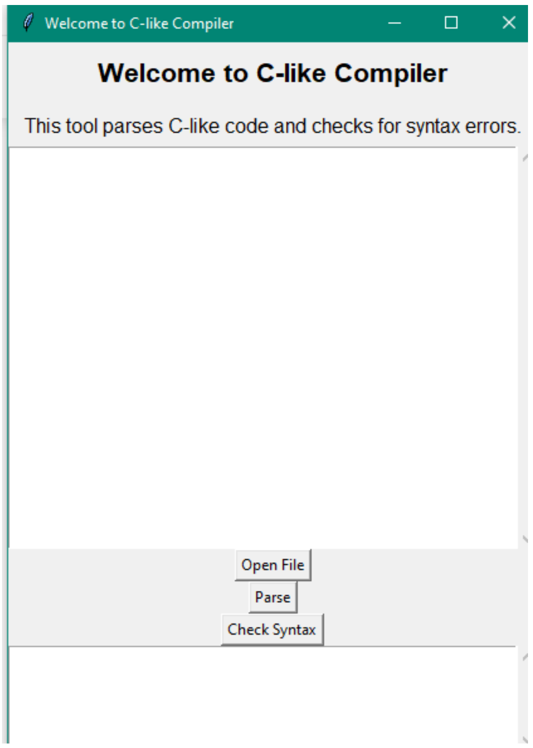
```
['int', 'main', '(', ')', '{', 'int', 'x', '=', '10', ';', 'float', 'y', '=', '3.14', ';', 'if', '(', 'x', '>', '0', ')', '{',
'y', '=', 'y', '+', '1.0', ';', '}', 'return', '0', ';', '}']
```

Parse Tree:

```
('program', [('declaration_list', [('declaration', ('function_declaration', ('type_specifier', 'int'),
'main', ('parameters', 'VOID'), ('compound_statement', [('statement_list', [('statement',
('expression_statement', ('expression', [('variable', 'int'), '=', ('expression', [('variable', 'x'),
'10'])]))), ('statement', ('expression_statement', ('expression', [('variable', 'float'), '=',
('expression', [('variable', 'y'), '3.14'])]))), ('statement', ('selection_statement', 'if',
('expression', [('variable', 'x'), '>', '0']), ('compound_statement', [('statement_list',
[('statement', ('expression_statement', ('expression', [('variable', 'y'), ('expression',
[('variable', 'y'), '+', '1.0'])])))])])), None)])), ('return_statement', ('expression', '0'))])]))))])]
```

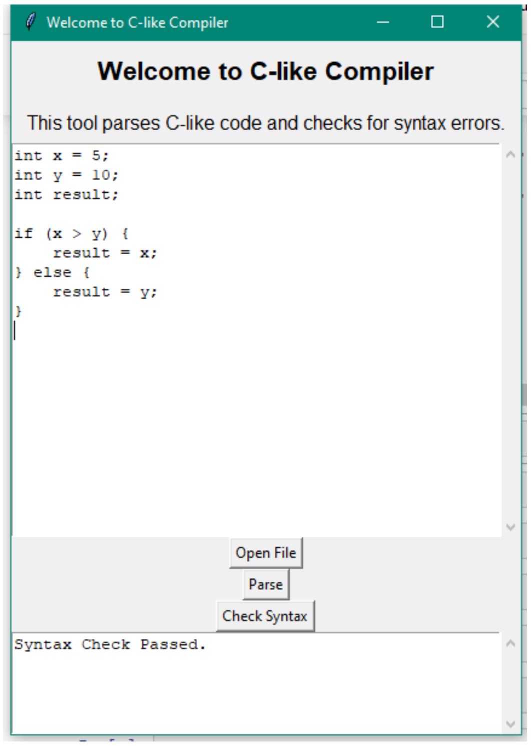
5. Graphical User Interface (GUI):

- The GUI provides a user-friendly interface for interacting with the C-Like Compiler.
- It includes features such as a text area for code input, buttons for file handling and functionality execution, and an error logger.
- Users can write code directly in the text area or upload code files using the "Open File" button.
- The "Check Syntax" button triggers the lexer and parser to tokenize and parse the input code, respectively. Syntax errors, if any, are displayed in the error logger.
- The "Execute" button executes the parsed code according to the defined grammar rules and displays the output or performs the specified action.



Welcome to C-like Compiler

This tool parses C-like code and checks for syntax errors.



Welcome to C-like Compiler

This tool parses C-like code and checks for syntax errors.

```
int x = 5;
int y = 10;
int result;

if (x > y) {
    result = x;
} else {
    result = y;
}
```

```
Syntax Check Passed.
```