***Lab Terminal***



***Compiler Construction***

# ***Submitted to:***

**Dr Bilal Bukhari**

# ***Submitted by:***

**Syed Afzal Hussain**

**(CIIT/Fa21-BCS-055/ATK)**

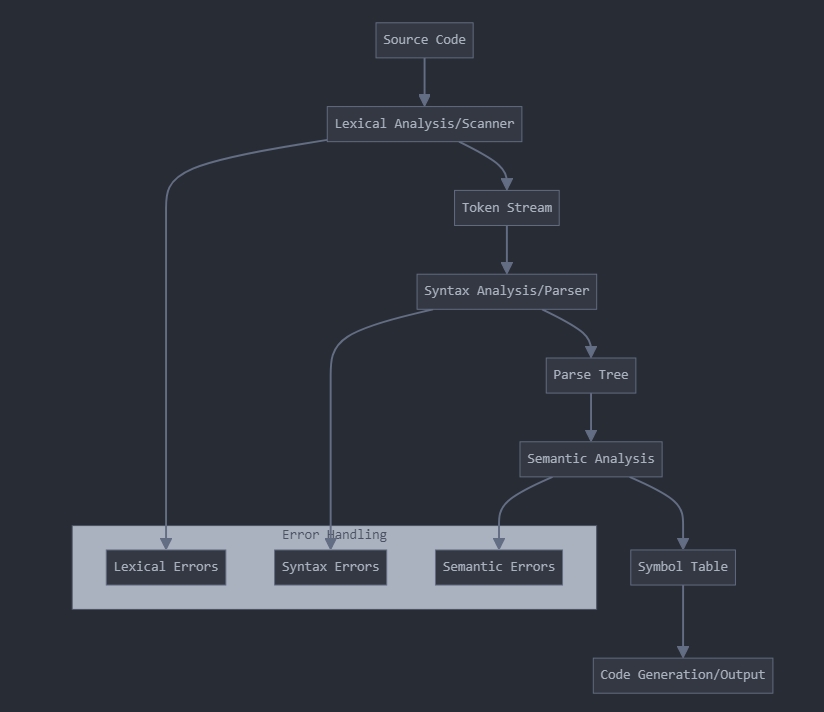
***Jan 3, 2025***

***Department of Computer Sciences***

***COMSATS UNIVERSITY ISLAMABAD***

***ATTOCK CAMPUS***

1.Class of Sequence :



2.Code function of mini compiler:

public Token GetNextToken() {

while (currentChar != EOF) {

// Skip whitespace

if (char.IsWhitespace(currentChar)) {

SkipWhitespace();

continue;

}

// Identify numbers

if (char.IsDigit(currentChar)) {

return ExtractNumber();

}

// Identify identifiers/keywords

if (char.IsLetter(currentChar)) {

return ExtractIdentifier();

}

// Handle operators

if (IsOperator(currentChar)) {

return ExtractOperator();

}

}

return new Token(TokenType.EOF, "");

}

// Parser function example - Expression parsing

public Node ParseExpression() {

Node left = ParseTerm();

while (currentToken.Type == TokenType.PLUS ||

currentToken.Type == TokenType.MINUS) {

Token op = currentToken;

Eat(currentToken.Type);

Node right = ParseTerm();

left = new BinaryOperatorNode(left, op, right);

}

return left;

}

3.Optimization Used:

Constant Folding: Evaluating constant expressions at compile time

Dead Code Elimination: Removing unreachable code

Common Subexpression Elimination: Avoiding redundant computations

Strength Reduction: Replacing expensive operations with simpler ones

Identifier Length Optimization: Using minimum length for identifiers while maintaining uniqueness

4.Test Cases:

int main(){

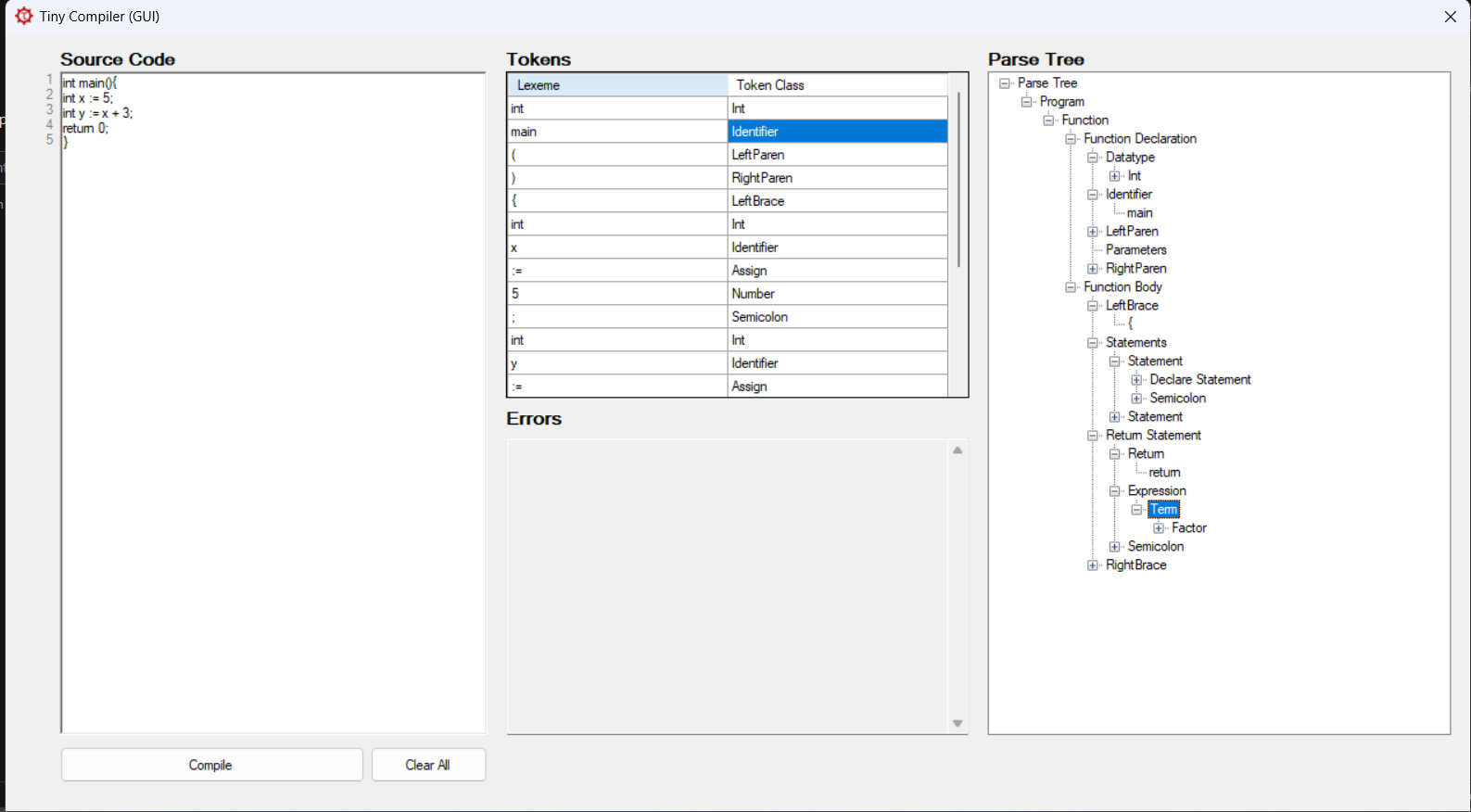
int x := 5;

int y := x + 3;

return 0;

}

Output:



// Test Case 2: Conditional Statements

int main(){

int x := 5;

int y := x + 3;

if (x > 0) {

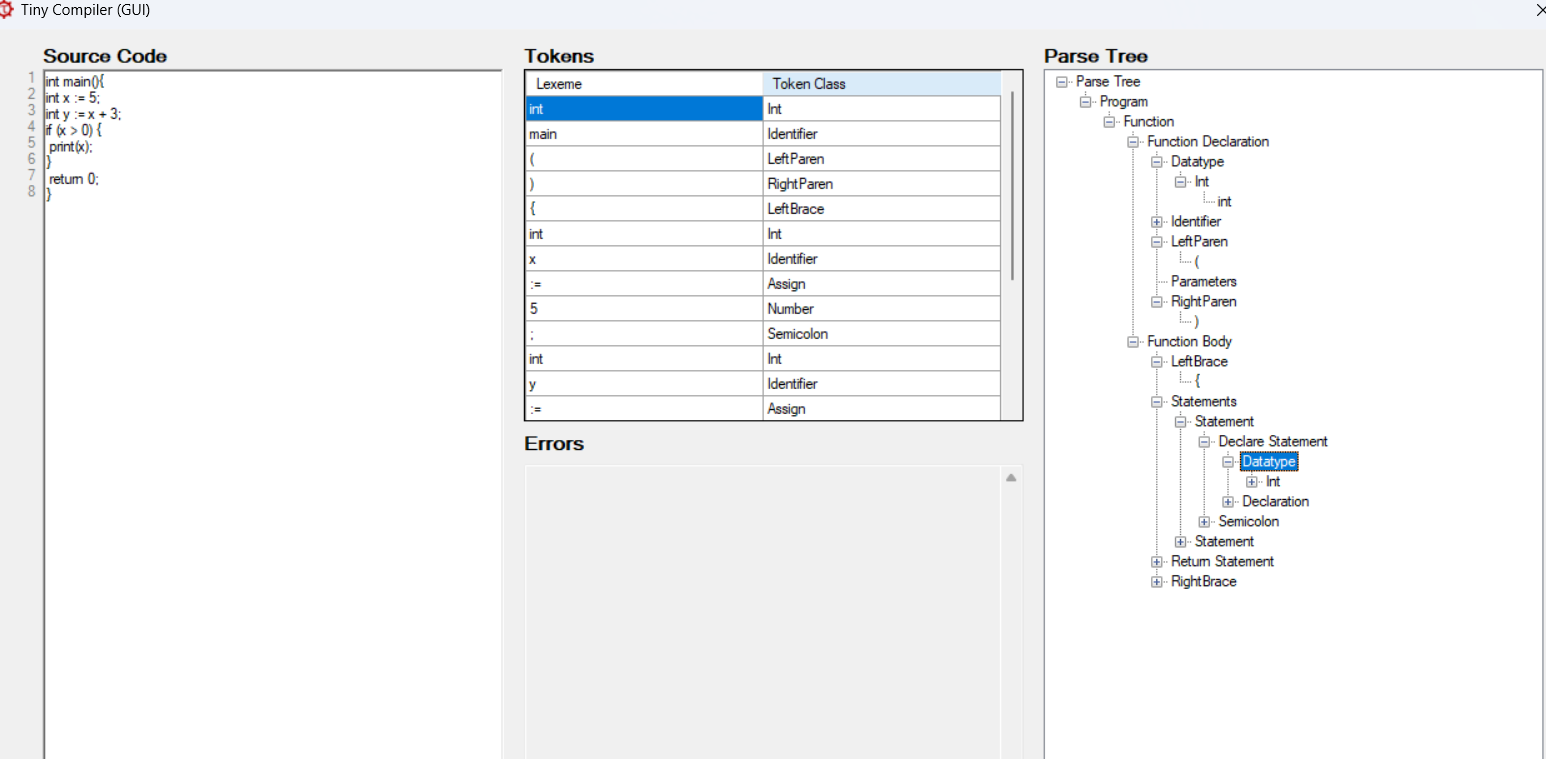
print(x);

}

return 0;

}

Output:



// Test Case 3: Loop Construct

int main(){

int x := 5;

int y := x + 3;

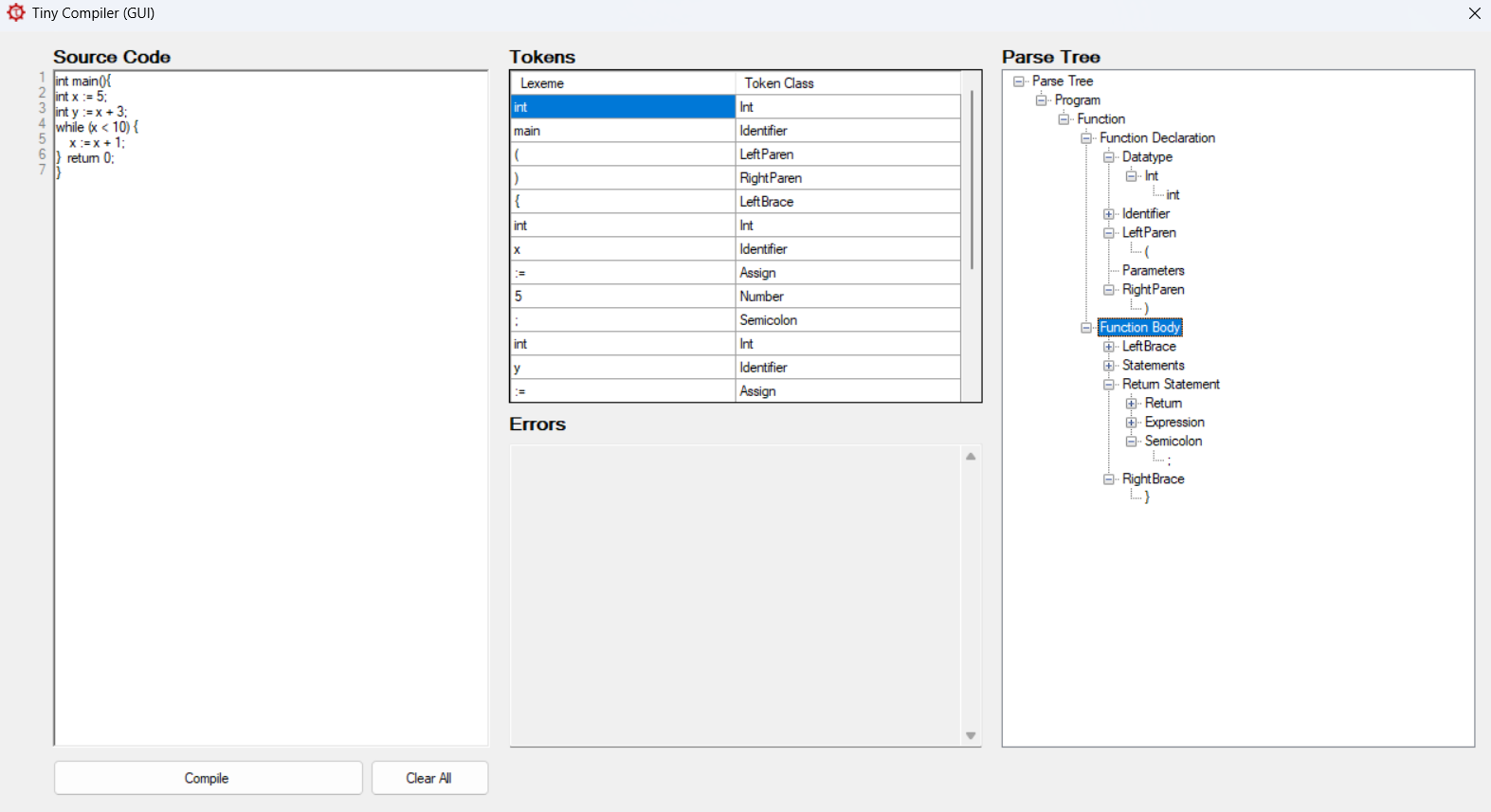
while (x < 10) {

x := x + 1;

} return 0;

}

Output:



// Test Case 4: Function Definition and Call

int main(){

int x := 5;

int y := x + 3;

function add(a, b) {

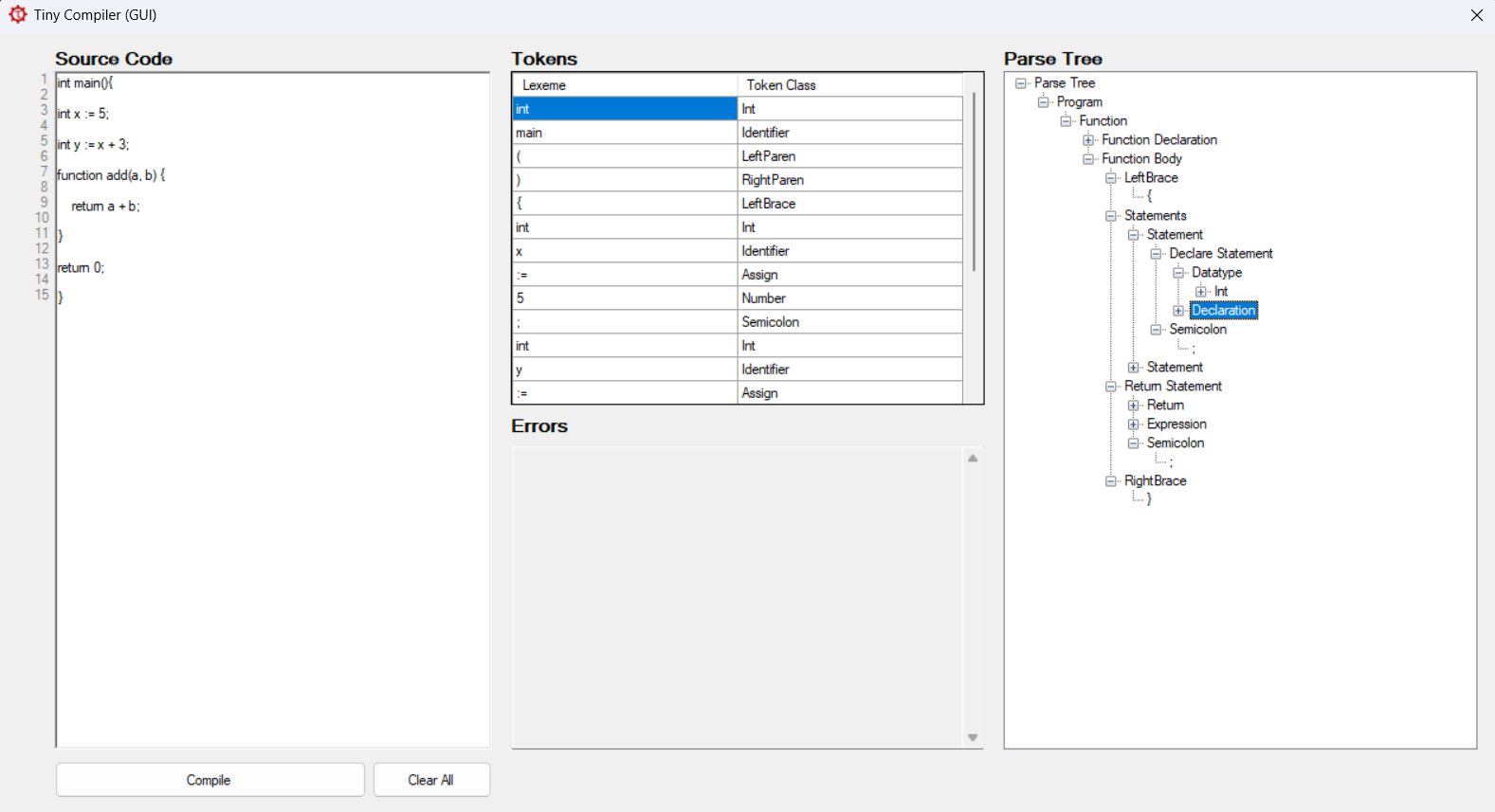
return a + b;

}

return 0;

}

Output:



5.Semantic Analysis:

Parse Tree Traversal

Traverse the parse tree generated by the parser.

For each node:

Check its type (e.g., assignment, operation, variable declaration).

Validate semantic rules.

Example for handling variable declarations and usage:

public void Analyze(ParseTreeNode root) {

if (root.Type == NodeType.VariableDeclaration) {

string varName = root.Children[0].Value;

string varType = root.Children[1].Value;

symbolTable.AddSymbol(varName, varType);

}

if (root.Type == NodeType.Assignment) {

string varName = root.Children[0].Value;

var symbol = symbolTable.GetSymbol(varName);

string assignedType = EvaluateExpression(root.Children[1]);

if (symbol.Type != assignedType) {

throw new Exception($"Type Mismatch: Cannot assign {assignedType} to {symbol.Type}.");

}

}

foreach (var child in root.Children) {

Analyze(child); // Recursively process child nodes.

}

}

***The End***