COMSATS University Islamabad,

Attock Campus

**Department of Computer Science**



Submitted by:

**Muhammad Hamza (FA21-BCS-058)**

Submitted to:

**DR. Bilal Bukhari**

**Subject:** Compiler Construction

**Lab Terminal**

**Q5 :- Explain the functions which performs the semantic analysis in my mini compiler.**

**1. Symbol Table Management**

Maintains a record of identifiers (variables, functions, etc.) and their associated attributes (type, scope, etc.).

**Function:** AddSymbol

Registers a new symbol in the table.

public void AddSymbol(string name, string type, int scope)

{

if (symbolTable.ContainsKey(name))

{

throw new Exception($"Error: Symbol '{name}' already defined.");

}

symbolTable[name] = new SymbolInfo(type, scope);

}

**Function:** LookupSymbol

Checks if a symbol is defined in the current or parent scope.

public SymbolInfo LookupSymbol(string name)

{

if (symbolTable.TryGetValue(name, out var symbol))

{

return symbol;

}

throw new Exception($"Error: Symbol '{name}' is not defined.");

}

**2. Type Checking**

Ensures that operations and expressions use compatible data types.

**Function**: CheckTypes

Validates compatibility between operands.

public void CheckTypes(string type1, string type2, string operation)

{

if (type1 != type2)

{

throw new Exception($"Type mismatch: Cannot apply '{operation}' between '{type1}' and '{type2}'.");

}

}

**3. Scope Management**

Manages nested scopes, ensuring variables are used in the correct context.

**Function:** EnterScope

Pushes a new scope onto the stack.

public void EnterScope()

{

currentScope++;

}

**Function**: ExitScope

Pops the current scope, removing symbols that go out of scope.

public void ExitScope()

{

currentScope--;

// Remove symbols defined in the current scope

symbolTable = symbolTable

.Where(entry => entry.Value.Scope < currentScope)

.ToDictionary(entry => entry.Key, entry => entry.Value);

}

**4. Semantic Rule Validation**

Checks adherence to language-specific rules, such as function argument counts or control flow restrictions.

**Function:** ValidateFunctionCall

Ensures the correct number and type of arguments are passed to a function.

public void ValidateFunctionCall(string functionName, List<string> argumentTypes)

{

var functionInfo = LookupSymbol(functionName);

if (functionInfo.Arguments.Count != argumentTypes.Count)

{

throw new Exception($"Error: Function '{functionName}' expects {functionInfo.Arguments.Count} arguments but got {argumentTypes.Count}.");

}

for (int i = 0; i < argumentTypes.Count; i++)

{

CheckTypes(functionInfo.Arguments[i], argumentTypes[i], "argument");

}

}

**5. Control Flow Validation**

Ensures constructs like loops and conditionals are used correctly.

**Function:** ValidateReturnStatement

Checks if a function returns the correct type.

public void ValidateReturnStatement(string expectedType, string actualType)

{

if (expectedType != actualType)

{

throw new Exception($"Error: Return type mismatch. Expected '{expectedType}', got '{actualType}'.");

}

}

**6. Constant Folding and Evaluation**

Evaluates constant expressions at compile time.

**Function:** EvaluateConstantExpression

Performs constant expression evaluation.

public int EvaluateConstantExpression(string expression)

{

// Example: Simple integer arithmetic evaluation

return (int)new DataTable().Compute(expression, string.Empty);

}