**Name:Mohammad Asim Khan  
Reg no:FA21-BCS-037  
Compiler Construction Lab Terminal**

**Question 5: Semantic Analysis in code**

**Answer:**

#### Semantic Analysis in the Code

The SymbolTable class plays a pivotal role in the semantic analysis phase. Here's how it functions and integrates into the compiler:

#### ****Managing Variable Declarations****

**Code:**

public void DeclareVariable(string name, Type value)

{ if(\_variables.ContainsKey(name))

throw new SemanticException("Variable " + name + " already exist");

\_variables[name] = value;

\_values[name] = value.GetDefaultValue();}

**Explanation**:

* When a variable is declared in the source code, this method ensures:
  + The variable is not already declared.
  + If valid, the variable is added to the \_variables dictionary along with its type.
  + An initial default value is assigned using GetDefaultValue.

#### ****2. Ensuring Variable Usage Validity****

**Code**:

public Type GetVariableType(string name)

{ if (!\_variables.ContainsKey(name))

throw new SemanticException("Variable " + name + " doesn't exist");

return \_variables[name];

}

**Explanation**:

* Before a variable is used in an expression or statement, its type is checked.
* If the variable has not been declared, a SemanticException is thrown, signaling an error in the source program.

#### ****3. Managing Variable Values****

**Code**:

public InterpreteValue GetVariableValue(string name)

{ return \_values[name];}

public void SetVariableValue(string name, InterpreteValue value)

{ \_values[name] = value;}

**Explanation**:

* These methods handle retrieving and updating the values of variables during interpretation.
* They ensure that variables are used only after being assigned valid values.

1. **Singleton Pattern for Centralized Access**

**Code:**

**public static SymbolTable Instance {**

**get{**

**if(\_instance==null)**

**\_instance=new SymbolTable();**

**return \_instance;} }**

**Explanation**:

* The SymbolTable class is implemented as a singleton, ensuring that there is a single shared instance throughout the compiler.
* This design ensures consistency and allows all semantic analysis components to access the same symbol table.

#### ****Integration with Other Components****

**Semantic Nodes**: Each node in the abstract syntax tree (e.g., AssignmentNode, ForStatementNode) interacts with the SymbolTable for type checking, variable lookup, and value assignment.

**Example in** AssignmentNode:

**Code:**

var variableType = SymbolTable.Instance.GetVariableType(variableName);

if (variableType != expressionType)

throw new SemanticException("Type mismatch in assignment");

This code validates that the type of the variable matches the type of the expression being assigned.