**COMPILER CONSTRUCTION**

**LAB TERMINAL**

**Student Name:** Hafza Nawaz

**Registration no:** FA21-BCS-018

**Teacher Name:** Sir Bilal Ahmad

**Question 3:**

**Explain function that perform semantic analysis in your mini compiler ?**

**Answer:**

**Key Semantic Analysis Functions and Their Roles**

Semantic analysis in compiler is primarily handled during the parsing phase (Parser.py) and involves interactions with the symbol table (SymbolTable.py). There isn't a dedicated "semantic analysis" pass. It is intermixed with syntax analysis.

**1.Parser.parse\_declaration():**

Purpose: Handles the parsing of variable declarations (int x;, int a, b;).

**Semantic Actions:**

Type Verification: Checks if the declared type is valid (int).

Symbol Table Insertion: Creates a new entry in the symbol table for each declared variable using SymbolTable.insert(), storing the variable's name and type.

Redeclaration Check: It implicitly checks for duplicate declarations of same variable in the same scope while inserting symbol.

def parse\_declaration(self):

type\_ = self.current\_token.value

self.consume\_token("keyword", type\_) # e.g. int

while True:

var\_name = self.current\_token.value

self.consume\_token("identifier")

if not self.symbol\_table.check(var\_name):

self.symbol\_table.insert(var\_name, type\_, self.current\_scope, 0) # insert

else:

self.error(f"Variable {var\_name} already declared in current scope")

if self.current\_token.type == "comma":

self.consume\_token("comma")

else:

break

self.consume\_token("semicolon") # ;

Explanation: First it consumes keyword int. It then keeps consuming the variable name and insert that in the symbol table along with the type and current scope. If variable is already declared then error is reported. After that it check the end of the statement by consuming ;

**2.Parser.parse\_assignment():**

Purpose: Parses assignment statements (x = 10;, y = a + 5;).

Semantic Actions:

Variable Existence Check: Before creating assignment statement AST, it checks if the variable exists in the symbol table using SymbolTable.lookup(). If not, it reports an error because a variable has to be declared before being used.

Type Checking (Implicit): Type check will not be fully performed at parser but it does not allow assignment of variable on non-variable entities. Further type checking is not done at the parser level and during code generation.

def parse\_assignment(self):

var\_name = self.current\_token.value

self.consume\_token("identifier")

if not self.symbol\_table.check(var\_name):

self.error(f"Variable {var\_name} not declared in current scope")

self.consume\_token("equal")

expr\_node = self.parse\_expression()

self.consume\_token("semicolon") # ;

return ast.Assign(var\_name, expr\_node) # AST node creation

**3.Parser.parse\_expression() (and related functions like parse\_term, parse\_factor):**

Purpose: Parses expressions (e.g., a + b, x \* 5, (a + 2) \* b).

Semantic Actions:

Variable Existence Check: During evaluation of factors, it checks variable existence in symbol table using SymbolTable.check().

Type Checking (Implicit): During code generation phase it will perform implicit type checks during code generation where type mismatches will be caught.

def parse\_factor(self):

if self.current\_token.type == 'identifier':

var\_name = self.current\_token.value

if not self.symbol\_table.check(var\_name):

self.error(f"Variable {var\_name} not declared in current scope")

self.consume\_token('identifier')

return ast.Variable(var\_name)

elif self.current\_token.type == 'number':

num = int(self.current\_token.value)

self.consume\_token('number')

return ast.Number(num)

elif self.current\_token.type == 'left\_paren':

self.consume\_token('left\_paren')

expr\_node = self.parse\_expression()

self.consume\_token('right\_paren')

return expr\_node

else:

self.error("Expected number, variable, or left parenthesis")

**Key Points**

Symbol Table Usage: The symbol table (SymbolTable.py) is central to semantic analysis. It's used to store and look up variable and function information.

Implicit Type Checking: Type checking is mostly implicit, i.e not performed explicitly in the parser or any dedicated pass, but during code generation mismatched type are caught.

Intermixed with Parsing: Semantic analysis is not a separate phase. Instead, it is integrated with the parsing process.

**Screenshot of code semantic analysis using symbol table etc :**

