Symbol Table Implementation

The symbol table is implemented using a linked list, where each symbol (variable or function) is represented by a Symbol struct, and the list is managed by the SymbolTable struct that contains details such as name, type, scope level and initialization status.

Components:

Symbols (Symbol struct): Each symbol stores information about a declared variable such as its name, type, scope level, and whether it has been initialized

Symbol Table (SymbolTable struct): This structure holds the linked list of symbols and tracks the current scope level

Operations:

1. **Initialization**

The symbol table is initialized with an empty list of symbols and the scope level set to zero. This ensures that symbols can be added dynamically as they are found in the code.

2. **Adding symbols**

When a new variable or function is declared, it is added to the symbol table. The process is to first create a new symbol entry, assign it to the current scope level, marking it as uninitialized by default and linking it to the symbol list

3. **Symbol lookup**

To verify whether a symbol has been declared before its usage. The symbol table supports both global and current scope lookup which either searches for a symbol throughout all scopes or restricts the search to the most recent scope.a

4. **Scope management**

Symbol table uses scope tracking functions to enter a new scope (increments the scope level) and exiting a scope (removes all symbols declared in the current scope before decrementing the scope level)

5. **Removing symbols**

Upon exiting a scope, variables declared at that level are removed. This is handled by scanning the list and deallocating any symbols belonging to the current scope

6. **Freeing the symbol table**

To prevent memory leaks, the symbol table is freed at the end of semantic analysis. This involves deallocating all stored symbols and the table structure itself.

Semantic Checking Rules

Semantic checking ensures that the program follows correct variable usage, type safety and function rules.

1. Variable Declaration rules
    a. Variables must be declared before use
    b. Redeclaration of a variable in the same scope is not allowed
    c. Variables must be initialized before they are used

2. Type Checking rules
    a. Expressions must have matching operand types
    b. Assignments must follow type compatibility between the assigned value and variable
    c. Function arguments must match expected types

3. Function Call rules
    a. Functions must be called with correct number of arguments
    b. Arguments must be of valid types

4. Scope rules
    a. Variables declared inside a block are not accessible outside that block
    b. When exiting a block, variables declared within it are removed from the symbol table

5. Control Flow Validation
    a. Conditions in if, while and repeat-until statements are supposed to be boolean expressions
    b. Functions must return values of the correct type

Semantic Error Handling

SEM_ERROR_UNDECLARED_VARIABLE

- This error occurs when a variable is used before it has been declared in the symbol table
  Eg: x = 5; //Error: ‘x’ is not declared
- Detection: The function lookup_symbol() searches for the variable in the symbol table. If the variable is not found, the above error is triggered.

SEM_ERROR_REDECLARED_VARIABLE

- This error occurs when a variable is declared more than once within the same scope
  Eg: int x;
  int x; //Error: ‘x’ already declared in this scope
- Detection: The function check_declaration() checks if a variable already exists in the current scope using lookup_symbol_current_scope(). If it does, the above error is reported.

SEM_ERROR_TYPE_MISMATCH

- This error occurs when an operation or assignment involves incompatible types
  Eg: int x = “Hello”; //Error: Type mismatch
- Detection: The function check_expression() verifies the types of operands. If the types are incompatible, the above error is raised

SEM_ERROR_UNINITIALIZED_VARIABLE

- This error occurs when a variable is used before being assigned a value
  Eg: int x;
  print (“%d”, x); //uninitialized x value
- Detection: During a lookup in check_expression(), if a variable is found but has not been initialized (is_initialized == 0), the above error is triggered

SEM_ERROR_INVALID_ARGUMENT

- This error occurs when an invalid argument is passed to a function
  Eg: factorial (-5); //Error: Invalid argument for function ‘factorial’
- Detection: In check_function_call(), if an argument does not meet the expected conditions (eg: non-negative for factorial function), the above error is raised

SEM_ERROR_FUNCTION_CALL_NO_ARGUMENTS

- This error occurs when a function that requires arguments is called without them.
  Eg: factorial (); //Error: Function
- Detection: In check_function_call(), if node -> args == NULL, the above error is raised

SEM_ERROR_FUNCTION_CALL_TOO_MANY_ARGUMENTS

- This error occurs when a function is called with more arguments than it can support
  Eg: factorial (5,3); //Error: Function has too many arguments
- Detection: In check_function_call(), if more than one argument is passed (node -> args -> right != NULL), then above error is raised

Testing:

While testing the first few lines (both valid and invalid cases), the semantic analyzer runs perfectly. However, there were some issues with the parser that we weren’t able to completely fix that caused an error in "if (x > 0) {\n"

```
PS C:\Users\jayan\Desktop\phase3-w25> gcc src/lexer/lexer.c src/parser/parser.c src/semantic/semantic.c -o semantic.exe
PS C:\Users\jayan\Desktop\phase3-w25> .\semantic.exe
Analyzing input:
int x;
x = 42;
z = 10;
x = test + 1;



int
x
;
x
=
42
;
z
=
10
;
x
=
test
+
1
;
AST created. Performing semantic analysis...

Semantic Error at line 2: Undeclared variable 'z'
Semantic Error at line 4: Undeclared variable 'test'
Semantic analysis failed. Errors detected.
PS C:\Users\jayan\Desktop\phase3-w25>
```

```
PS C:\Users\jayan\Desktop\phase3-w25> gcc src/lexer/lexer.c src/parser/parser.c src/semantic/semantic.c -o semantic.exe
PS C:\Users\jayan\Desktop\phase3-w25> .\semantic.exe
Analyzing input:
int x;
x = 42;



int
x
;
x
=
42
;
AST created. Performing semantic analysis...

Semantic analysis successful. No errors found.
```

```
PS C:\Users\jayan\Desktop\phase3-w25> gcc src/lexer/lexer.c src/parser/parser.c src/semantic/semantic.c -o semantic.exe
PS C:\Users\jayan\Desktop\phase3-w25> .\semantic.exe
Analyzing input:
int x;
int x;



int
x
;
int
x
;
AST created. Performing semantic analysis...

Semantic Error at line 2: Variable 'x' already declared in this scope
Semantic analysis failed. Errors detected.
```