Semantic AnalyzerCurrent

Project Report: JAM Language Implementation

---

1. Lexer

Functionality:

- Tokenizes the input source code.
- Handles keywords, identifiers, literals, operators, and delimiters.
- Skips whitespace and comments.

Key Functions:

- `get_next_token()`
- `create_token()`
- `identifier()`
- `number()`
- `string_lit()`
- `skip_whitespaces()`
- `skip_comments()`
- `advance()`
- `curr_char()`
- `next()`

---

2. Parser

Functionality:

- Parses tokens into an Abstract Syntax Tree (AST).
- Handles function declarations, variable declarations, return statements, and expressions.
- Includes error handling for syntax errors.

Key Functions:

- `parse_program()`
- `parse_statement()`
- `parse_function_decl()`
- `parse_var_decl()`
- `parse_return_stmt()`
- `parse_expression()`
- `create_ast_node()`
- `expect_token()`
- `advance_token()`

---

3. Semantic Analyzer

Current Functionality:

- **Symbol Table Management**: Adds and checks symbols (variables and functions).
- **Type Checking**: Ensures type compatibility in expressions and assignments.
- **Function Validation**: Checks function declarations and calls, including argument counts.
- **Variable Usage**: Ensures variables are declared before use.

Potential Improvements:

- **Scope Management**: Implement scope handling to differentiate between global and local variables.

Key Functions:

- `addSymbol()`
- `isDeclared()`
- `getType()`
- `getFunctionArgCount()`
- `getASTArgCount()`
- `checkBinaryExpression()`
- `checkVariableDeclaration()`
- `checkVariableUsage()`
- `checkFunctionDeclaration()`
- `checkFunctionCall()`
- `traverse()`

4. Execution Engine

Current Functionality:

- **Expression Evaluation**: Evaluates expressions, including handling type conversion.
- **Statement Execution**: Executes variable declarations, assignments, and return statements.
- **Function Execution**: Manages function calls using a call stack and handles variable environments.

Potential Improvements:

1. **Memory Management**: Ensure all dynamically allocated memory is properly freed to avoid memory leaks.
2. **Return Value Management**: Instead of printing return values, consider storing them in a specific variable or returning them to the caller.

Key Functions:

- `freeSymbolTable()`
- `freeEnv()`
- `cleanup()`
- `addSymbol()`
- `isDeclared()`
- `addEnvEntry()`
- `getEnvEntry()`
- `pushCallStack()`
- `popCallStack()`
- `evaluateExpression()`
- `executeStatement()`
- `executeFunction()`
- `execute()`