Report
Compiler Design
comp-6421

Assignment 4
Symbol Table and Semantic Analyzer

Professor: Dr. Joey Paquet

Neona Sheetal Pinto
40172626

List of Semantic Rules Implemented

Symbol table creation phase
1. A new table is created at the beginning of the AST traversal for the global scope.
2. A new entry is created in the global table for each class declared in the program. These entries should contain links to local tables for these classes.
3. An entry in the appropriate table is created for each variable defined in the program, i.e. a class' data members or a function's local variables.
4. An entry in the appropriate table is created for each function definition (free functions and member functions). These entries should be links to local tables for these functions.
5. During symbol table creation, there are some semantic errors that are detected and reported, such as multiple declared identifiers in the same scope, as well as warnings such as for shadowed inherited members.
6. All declared member functions should have a corresponding function definition, and inversely. A member function that is declared but not defined constitutes a "no definition for declared member function" semantic error. If a member function is defined but not declared, it constitutes a "definition provided for undeclared member function" semantic error.
7. The content of the symbol tables should be output into a file in order to demonstrate their correctness/completeness.
8. Class and variable identifiers cannot be declared twice in the same scope. In such a case, a "multiply declared class", "multiply declared data member", or "multiply declared local variable" semantic error message is issued.
9. Function overloading (i.e. two functions with the same name but with different parameter lists) should be allowed and reported as a semantic warning. This applies to member functions and free functions.

• Semantic checking phase – binding and type checking
10. Type checking is applied on expressions (i.e. the type of sub-expressions should be inferred). Type checking should also be done for assignment (the type of the left and right-hand side of the assignment operator must be the same) and return statements (the type of the returned value must be the same as the return type of the function, as declared in its function header).
11. Any identifier referred to must be defined in the scope where it is used (failure should result in the following error messages: "use of the undeclared variable", "use of undeclared member function", "use of the undeclared free function", "use of undeclared class").
12. Function calls are made with the right number and type of parameters. Expressions passed as parameters in a function call must be of the same type as declared in the function declaration.
13. Referring to an array variable should be made using the same number of dimensions as declared in the variable declaration. Expressions used as an index must be of integer type. When passing an array as a parameter, the passed array must be of compatible dimensionality compared to the parameter declaration.
14. Circular class dependencies (through data members\inheritance) should be reported as semantic errors.
15. The "." operator should be used only on variables of a class type. If so, its right operand must be a member of that class. If not, an "undeclared data member" or "undeclared member function" semantic error should be issued.
16. In the Impl block, the name of the class should match the struct class name.

- Symbol table creation
  The phase is to create symbol tables and their entries. Almost all AST nodes are used to transmit symbol table information. Some nodes represent identifier declarations/definitions in the form of a symbol table record to be inserted in a symbol table.

VarDeclNode	variable entry
StructDeclNode	struct entry
FuncDeclNode	function entry
FuncDefNode	free function entry

  Some nodes represent a scope, in which case they need to create a new symbol table, and then insert the symbol table entries they get from their children.

ProgNode	global symbol table
Struct DeclNode	Struct table
FuncDefNode	function table

- Binding and type checking
  This phase computes the type of subtree expressions, assignment statements, return statements, etc., and do some semantic checking like array dimension consistency, function calls parameters, etc.

StructDeclNode	Circular struct class dependency
DimListNode	Array dimension
ReturnStatNode	Type error in return statement
AssignStatNode	Type error in assignment statement
AddOpNode	Type error in addition operation
MultOpNode	Type error in multiplication operation
ExprNode	Type error in expression
VariableNode	Use of the array with a wrong number of dimensions
TypeNode	Use of undeclared class

IdNode	Undeclared local variable
DotNode	Use of the array with a wrong number of dimensions/ Undeclared member function/ Undeclared data member
FuncCallNode	Undeclared free function/ Function call with the wrong type(number) of parameters/ Array parameter using a wrong number of dimensions
IndiceNode	Array index is not an integer

Design

The visitor pattern is used to do abstract syntax tree traversal for several phases: constructing subtree representation, generating symbol tables, and checking types.

Three concrete visitor classes, as well as the parent base visitor class, are created.

- The ***ReconstructSourceProgramVisitor*** class is to construct a string that represents the subtrees.
    - Constructing subtree expression: The semantic action is useful for debugging and informative error reporting, exposing the location and expression of the errors. It mainly aims represent to nodes that have expression meaning, but many nodes need to be traversed to transmit the information. In fact, the progNode, for example, has a m_subtreeString which contains all source tokens.
- The ***SymTabCreationVisitor*** class is to create symbol tables and symbol entries.
- The TypeCheckingVisitor class is to compute the type of subtrees and check type bindings.
- The ***SymTab*** class and ***SymTabEntry*** class represent the symbol table and symbol entry. Concrete SymTabEntry classes, including *StructEntry*, *FuncEntry*, *VarEntry* are created to represent symbol entries for classes, functions, and variables.

The class of ***SematicAnalyzer*** is responsible for dealing with every file. Having three visitor objects to do tree traversal starting from the root node(progNode) in three phases.

The ***SemanticAnalyzerDriver*** opens every file(.src) in a folder or a single file(.src), first creates a SyntacticAnalyzer object and then a SemanticAnalyzer object to handle the task of every visitor and output reports to files.

Tools

1. ***IntelliJ IDEA***: IDE platform for writing the source code. The IDE has smooth, efficient and clever code suggestions which makes it easy to type. Debugging is much faster compared to other IDE's.
2. ***Java.util***: Used util files in the source code for creating tables, hashmaps,. Etc.

3. ***Visitor pattern***: Used for following the codes example the professor provided, which gives a general idea to implement the pattern in this assignment. Since different semantic actions are needed to do semantic checking and the next assignment also contains semantic actions, this pattern separated different groups of operations, making the implementation much easier.