COMP 442: Compiler Design

Assignment 3: Semantic Analyzer

Done by:

Abdulla Alhaj Zin, 40013496

Date:

07/04/2020

To:

Dr. Joey Paquet

Section 1: List of Semantic Rules Implemented

Symbol table creation:

1. A new table is created at the beginning of the program for the global scope.
   Implemented and working perfectly.
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.
   Implemented and working perfectly.
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.
   Implemented and working perfectly.
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.
   Implemented and working perfectly.

5. During symbol table creation, there are some semantic errors that are detected and reported, such as multiply declared identifiers in the same scope, as well warnings such as for shadowed inherited members.
   Implemented and working perfectly
6. All declared member functions should have a corresponding function definition, and inversely. A member function that is declared but not defined constitutes an "no definition for declared member function" semantic error. If a member function is defined but not declared, it constitutes an "definition provided for undeclared member function" semantic error.
   Implemented and working perfectly

7. The content of the symbol tables should be output into a file in order to demonstrate their correctness/completeness.
   Implemented and working perfectly
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.
   Implemented and working perfectly

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.
   Implemented and working perfectly

•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).
    Implemented and but not working properly all the time.
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 undeclared local variable”, “use of undeclared free function”, “use of undeclared class”).
    Implemented and working perfectly

12. Functioncalls are made with the right number and type of parameters. Expressions passed as parameters in a functioncall must be of the same type as declared in the function declaration.
    Implemented except for checking the number of parameters in the function call.
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.
    Not implemented

14. Circular class dependencies (through data members\inheritance) should be reported as semantic errors.
    Implemented and working semi-perfectly
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, a “undeclared data member” or “undeclared member function” semantic error should be issued.
    Implemented and working perfectly.

Section 2: Design

The desing of the semantic analyzer was implemented by generating the symbol tables through the AST. The previous AST had to be redesigned to have designated node classes for each type to allow the use of the Visitor design pattern. Three visitors were implemented on the visitor design pattern, SymbolTableVisitor.cs, TypeCheckingVisitor.cs, and ASTVisitor.cs. Each visitor allows the compiler to add dynamic functionalities to each node when visiting them in the different traversal techniques. Unit tests as well as integration tests were implemented as well to test the completeness of the semantic checks. Finally, two files are generated, .outsymboltable and .outsemanticerrors which include the generated symbol table, and the semantic errors and warnings, respectively.

Section 4: Use of Tools

No external libraries were used outside of the built-in .Net core C# libraries. No special classes were used to handle any of the analysis.