Comp 442: Compiler Design
Concordia University, Montreal
Winter 2024
William Zijie Zhang
Concordia ID: 40176870

Assignment 4

William Zijie Zhang

1 Implemented Semantic Rules

The checklist below demonstrate all the semantic rules I have implemented.

1.1 Symbol Table Creation Phase

- ☑ New table is created at the beginning for the global scope.
- ☑ A new entry in the global table is created for every class.
- ☑ An entry is created for each variable defined in the program.
- ☑ An entry is created for each function definition.
- ☐ Semantic errors are detected and reported during symbol table creation.
- ☑ All declared member functions should have a corresponding function definition.
- ☑ The content of the symbol table should be output to a file.
- ☐ Class and variable identifiers cannot be declared twice in the same scope.
- ☐ Function overloading should be allowed and reported as a semantic warning.

1.2 Semantic Checking Phase

- ☑ Type checking is applied on expressions and assignment.
- ☑ Any identifier referred to must be defined in the scope where it is.
- ☐ Function calls are made with the right number and type of parameters.
- ☑ Referring to an array variable should be made using the same number of dimensions as declared in the variable.
- ☒ Circular class dependencies should be reported as semantic errors.
- ☑ The DOT operator should be used only on variables of a class type.

2 Design

In this section I will describe the structure of my solution for the semantic analysis portion of the compiler. Throughout the assignment, I have made significant usage of the Visitor pattern, which allows decoupling between subclass definitions and functionalities that act on those subclasses. I began by adjusting my AST tree generation process to generate corresponding subclasses of $Node$. Then I followed with the implementation of the $Visitor$ class, in Python, I had to import external decorators which allows for dynamic dispatching. I also implemented a generic accept method which first loops through all the children of a given node and finally also accepts the visitor.
Then I continued the implementation by completing the subclasses $SymbolTableVisitor$ and $TypeCheckingVisitor$ which perform different operations depending on the type of the node encountered. For the table generation, I separated the nodes into two categories, nodes with table entries and nodes with their own scopes. This way, given the bottom-order of visitor calls, we could easily fill out a symbol table by appending the table entries of its children. As for the type checking, I reused the information gathered in the symbol table to get the type of previously declared variables and propagated that information to the parent nodes. Finally, before the demonstration, if time allows it, I will give a try to other semantic checks such as valid accesses of struct members and verifying the parameters of function calls.

3 Use of Tools

I have made use of two external Python resources for the completion of this assignment: PyVisitor and PrettyTable. The former came in clutch since Python's structure didn't allow for multiple dispatching (for some odd reason I really cannot explain). Hence I had to resort to using these visitor decorators to be able to have the same result as other languages (but in this case the dispatch is performed dynamically). I also used an external table library which helped me easily display the program information in an orderly manner. However, for the type checking visitor, I wanted to query the tables and did not get satisfying results. Finally, as per usual, I made extensive usage of code examples and ideas from the lecture slides of Professor Joey Paquet.