CSE4100: Compilers **Fall 2015**

Homework 5: Semantic Analysis

Out Date: 11/05/2015
Due Date: 11/12/2015

Objectives

The purpose of this homework is to implement the key parts of the semantic analysis for `C--`.

Handout

The handout is available on the website. You will have to implement two new collection of classes, namely: *types* and *semantic descriptors*. Additionally, you will have to implement the virtual methods on the AST to carry out all the passes of the semantic analysis. To give some guidance, the skeleton of the descriptors and types are provided.

These are mere skeletons, you have to implement the body of the classes. The supporting code includes a few minor classes to represent scopes and the semantic environment. Once again, this code is meant to offer high-level guidance. Feel free to adapt it as you see fit.

Finally, the handout includes the solution to the previous homework. That is, `grammar.y`, `ast.H` and `ast.C` contain a correct attributed grammar and the definition of classes for AST nodes. I encourage all of you to keep on using your own solution as you are already familiar with it. It would take time to *acclimate* yourself with my implementation. Feel free to look it over to see where you have mistakes that should be fixed of course. If all else fails and you are desperate, you are welcome to use my solution as a starting point.

The analysis is broken down in several passes and this writeup discusses each pass in turn.

1 Pass 0

The very first pass is responsible for collecting the names of all the classes appearing in the program. It should not attempt to analyze relationships between classes (inheritance) expresses throught the `extends` annotation. These types are *hollow* at the beginning and will be filled in subsequent passes. The pass is nothing but a recursive traveral of the AST looking at top-level class names. Check the virtual method defined in the most abstract `Node` class. The method takes a single argument as input: the semantic environment (i.e., Γ in the lecture notes) that it modifies and uses during the traversal.

Naturally, you must insert code (in your main program) to

- instantiate the semantic environment
- call the first pass

2 Pass 1

Pass 1 is responsible for connecting all the classes collected in Pass 0 to properly reflect the inheritance relationships. Once pass 1 is complete, all the classes relationships are known.

3 Pass 2

The second pass is responsible for computing the types that appear within the classes themselves, i.e., it fills the class types with the attribute types and the method/constructor types. This needs to be done in pass 2 before analyzing the body of methods to be able to deal with `C--` program containing mutually recursive methods and classes. If the pass is successful, the next pass can proceed and verify all the method bodies. The second traversal should, of course, re-open scopes that were created during the previous pass (pass 1) so that the semantic environment state is identical to what it was at the same node of the AST during the first pass. To do this, simply preserve the scopes when you close them when leaving classes during pass 1 and store the scope in the AST itself so that you can reopen it when you re-enter the scope. Once again, go back to your main program and add a suitable call to the virtual method implementing the pass#2 of the analysis.

4 Pass 3

The third pass is responsible for analyzing the body of methods. This is deferred given that all the declarations must be known ahead of time (`C--`, like Java, allows forward declarations). This analysis build upon the Pass #2 analysis and must, once again, restore the environment to the state it was in at each node of the AST. The visit actually performs new work on the AST nodes corresponding to syntactic constructions that appear in the body of methods/constructors (i.e., all the statements and expressions). At the end of the third pass, a final verification must be carried out to make sure that

- all the methods have been defined
- all the classes have at least one constructor
- the 'Main' class exists (The entry point of a `C--` is simply the instantiation of a class named `Main` and the call of its constructor).

Edit your main program to call the third pass. Once that call returns, a simple query on the semantic environment (or on the root of the AST) should be sufficient to establish whether the program passed the semantic analysis and whether the compiler can proceed to code generation!

Recommendation

As you build the passes, you will realize that you need to "fill in" the implementation of the types and the descriptors. Do them as you go and test each little piece at a time. Do not attempt to write everything all at once and only test at the end. Do this piecemeal.

Handin

Once you are ready to handin follow the same procedure as before and submit to moodle.