CSE 425 Studio: Basic Semantics I

These studio exercises and the next ones are intended to acquaint you with basic ideas and techniques for managing programming language semantics, which you will do in a multi-paradigm programming language (C++).

In this studio you will again work in self-selected groups of ~3 people, and will report the results of your work on the studio exercises in an e-mail to the course account. Students who are more familiar with the material are encouraged to partner with and help those who are less familiar with it, and asking questions of your classmates and professor during the studio sessions is highly encouraged as well. The links on the main course web page also may be helpful for these exercises, as resources.

Please record your answers as you work through the following exercises. After you have finished please email your answers to the required exercises, and to any of the enrichment exercises you completed, to the cse425@seas.wustl.edu course email account, with "Basic Semantics Studio I" in the subject line. The enrichment exercises let you dig deeper into the material, especially if you finish the required ones.

PART I: REQUIRED EXERCISES

1. Form a team of ~3 people and write down your names as the answer to this exercise.

2. Open up Visual Studio 2013 and create a new project for this studio (for example, named **`BasicSemanticsStudioI`** or something similar that identifies which studio this is for). Before making any changes, notice the signature of the main function that Windows provides for you in your project's main source file (e.g., **`BasicSemanticsStudioI.cpp`**) and write it down. Then, modify the main function signature so that it matches the standard portable (e.g., between Windows and Linux) main function signature (and variable names) for a C++ program: **`int main (int argc, char * argv[])`**. As the answer to this exercise, describe for both the generated and modified versions of the main function: (1) which identifiers are part of the global scope and what their attributes are; and (2) which identifiers are part of the main function scope, and what their attributes are.

3. Above your main function, declare a simple struct (with no public, private, or protected access specifiers) that has two integer member variables and a default constructor that sets their values both to 0. In your main function write a statement that accesses and prints out the values of those member variables. Build and run your program and observe what happens. Then, change the type of declaration from struct to class and try to build your program again. As the answer to this exercise please say what your observations tell you about how class vs. struct scopes differ, and specifically which identifers and attributes were affected by that difference in this exercise.

4. Comment out any code from the previous exercise that won't allow your program to compile. Declare (and define if you'd like to give it constructors) a basic C++ struct to represent an attribute, which has two strings as member variables (one for the name of the attribute and one for the information associated with it). Declare (and define if you'd like to give it constructors) a basic C++ struct to represent an identifier in a symbol table, which has a string for the name of the identifier and a vector of attribute objects (of the struct type you just wrote in the previous part of this exercise). In your main function, declare identifier objects for the different kinds of attributes that you identified in the second exercise, and add appropriate attributes to them according to your answer to that exercise as well. After that, your main function should print out the contents of those identifier objects (including their attributes). Build and run your program, and as the answer to this exercise, show your code and the output your program produced.

5. Develop a symbol table class that has a vector of identifier objects (as declared in the previous exercise) as its only member variable. Declare and define an insertion operator (C++ **`operator<<`**) as a public member of that that class, which takes a single reference to a const identifier object (from the previous exercise) and pushes it back into the vector member variable. Also declare and define a stand-alone insertion operator (C++ **`operator<<`**) that is outside any class, which takes a reference to a non-const **`ostream`** and a reference to a const symbol table object and uses the ostream's << operator to output the contents of the symbol table object's vector member variable. In the symbol table class, declare that external operator to be a friend of the class (giving it access to private elements).

In your main function, declare a symbol table object and insert the identifiers from the previous exercise into it using its insertion operator. Then, using the cout standard output stream object and the external ostream insertion operator you just wrote, output the symbol table's contents. Compile and run your program and as the answer to this exercise show your code and the program's output.

Important: when you have finished these exercises, please make sure to keep your code for use in the next studio, where you will continue to extend the symbol table you have developed in these exercises.

PART II: ENRICHMENT EXERCISES (Optional, feel free to skip or try variations that interest you)

6. Extend your symbol table class from the previous exercise with an additional string member variable for the name of the symbol table. Also add a constructor that takes a string as its only argument and initializes its string member variable using the passed string parameter. Modify the external insertion operator that takes references to an ostream and a symbol table, so that it prints out the name of the symbol table before printing out its contents.

Modify your main function so that it constructs two separate symbol table objects, one for the global scope and one for the main function scope (giving them appropriate names like "global" and "main"), and inserts the same set of identifiers as before into one or the other of them (but not both – for example the identifier and attributes for the main function would go into the global symbol table, while the argc and argv identifiers and their attributes would go into the symbol table corresponding to the main function) and then uses cout and the overloaded insertion operator you just modified to print them out. Build and run your code and as the answer to this exercise show your code and the output the program produced.

7. Extend your symbol table class from the previous exercise with another member variable that is a map of identifiers and pointers to other symbol table objects that is initially empty. Also add a method that takes a reference to a symbol table as its only argument, and if and only if the name of the passed symbol table is not the same as another identifier already in the symbol table on which the method is being called, adds a new entry to the map with an identifier for the passed symbol table's name, and the address of the passed symbol table object. In your main function, invoke that method on the "global" symbol table object, passing in the "main" symbol table object.

Modify the external insertion operator that takes an ostream and a symbol table so that it recursively prints out symbol tables using indentation to show each level, and modify your main function so that it only calls that operator with the "global" symbol table object (since the "main" symbol table object now should be linked under the global symbol table object, and should be printed out as a recursive sub-table). Build and run your program, and as the answer to this exercise please show your code and the output your program produced.