CSE 425 Studio: Basic Semantics II

These studio exercises are intended to acquaint you with additional ideas and techniques for managing programming language semantics, which you will again do in 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. There are also links on the main course web page, which you are also encouraged to use as resources.

Please record your answers as you work through the following exercises. After you have finished please e-mail your answers to the required exercises, and to any of the enrichment exercises you completed, to the cse425@seas.wustl.edu course account, with "Basic Semantics Studio II" in the subject. 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 Win32 Console Application (for example, named **`BasicSemanticsStudioII`** or something similar that identifies which studio this is for). 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[])`**. Outside your main function, declare a global integer variable and initialize it to an explicit value. Inside your main function declare a local integer variable and initialize it to a different explicit value. Also in your main function, declare a pointer to an integer variable, and initialize it with the result of calling new to dynamically allocate (on the memory heap) an integer variable initialized with yet another different explicit value (**hint:** the syntax for initializing a dynamically allocated variable is to give its type and then an initialization value surrounded by parentheses – as though you were explicitly calling a constructor even for built-in types like integers in C++). In your main function print out the values of all three integer variables and the value of the pointer variable, and build and run your program. Show your code and the output your program produced, as the answer to this exercise.

3. In your main function code, use the address-of operator (the C++ **`&`** operator) to obtain and print out the addresses of the three integer variables, and the address of the pointer variable, all from the previous exercise. In your code, subtract the addresses of the three integer variables from each other and then multiply the result of each subtraction by the size of an integer (which can be obtained using the C++ **`sizeof`** operator) to obtain and print out the distances between them in bytes (subtraction of pointers or addresses gives the offset in the number of objects of that size that are between them, e.g., in a hypothetical contiguous array, which is why you have to multiply by the size of an object to get the actual distance in bytes). As the answer to this exercise please show the output your program produced and then say what your observations tell you about the relative ordering of global variables, stack variables, and heap variables in the memory used by your program, and also some sense of how big is each of the areas for which the observations clearly indicate such a conclusion.

4. Declare a number of additional pointers to integers, and initialize or assign them to point to different ones of the three integer variables from the previous exercises. By using assignment (the `=` operator), dereferencing (the `*` operator), and the address-of operator (operator `&`), and printing out the values of the integer variables and the pointers that point to them, examine what happens when you change (1) to which integer variable a pointer points, (2) the value of the integer variable being pointed to, or (3) both, from the perspective of the other pointers (and from the perspective of the integer variables themselves). As the answer to this exercise, briefly describe what you observed, and what those observations indicate about the semantics (especially regarding pointer/reference semantics versus storage/value semantics) of pointers and the variables to which they point in C++.

5. Extend the symbol table class that you developed in the previous studio, so that each identifier has as many of the following attributes that are well-defined for it in C++: its name, its location, its value, its (data) type, and its storage size. Using the structs and classes from the previous set of studio exercises, the C++ STL **`ostringstream`** class that can easily convert non-string values into strings that represent them, and the address-of, dereference, and size-of operators in C++, fill in those attributes so that they accurately reflect the contents of a basic main function (as of exercise 2 in the previous set of studio exercises, for example). Also modify the external insertion operator that takes an ostream and a symbol table, so that it prints out theses attributes fully including their names and values. Build and run your program, and as the answer to this exercise please show the code you added for this exercise and also the output of your program when you ran it.

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

6. Extend your structs, classes, and/or objects from the previous exercise so that your program also can deal with pointer variables, and design and within your symbol table object(s) implement a suitable set of attributes for each of the integer variables and pointer to integer variables in exercise 4 above. If necessary, again modify the external insertion operator that takes references to an ostream and a symbol table, so that it also prints out the identifiers and attributes you added in this exercise. Build and run your program, and as the answer to this exercise show the code you wrote and your program's output.