CSE 425 Studio: Data Types II

These studio exercises are intended to extend your familiarity with ideas and techniques for managing programming language data types, again 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. There are also links on the main course web page to helpful web sites for these exercises, 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 send an email containing your answers to the required exercises, and to any of the enrichment exercises you completed, with "Data Types Studio II" in the subject line to the course email account (cse425@seas.wustl.edu). The enrichment exercise is a chance to dig deeper into the material, especially if you breeze through 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 (for example, **`DataTypesStudioII`** 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[])`**.

In your main function, declare fixed sized arrays with all three basic types (enumerated, **`int`**, and **`char`** types) from the previous studio, both with and without initializing the values that those arrays contain, and print out the values that each array contains. As the answer to this exercise, describe how the array types are constructed in C++, for both cases (i.e., when values for initialization are or are not provided).

3. In your main function, declare pointers to the types contained in the arrays from the previous exercise, and initialize those pointers using the names of the arrays. Also declare references to the types contained in the previous exercise, and initialize them using the 0^{th} element of each array. Compare and contrast which of the different operators that can be used for arrays, pointers, and/or references can or cannot be used with the others (try prefix and postfix increment and decrement, dereference, size of, and address of operators plus any others you can think of) and for those that are legal to use for more than one of them, compare and contrast how those operators behave. As the answer to this exercise please summarize your observations about which operators are supported for arrays, pointers, and/or references, and any differences in how they behave for each type.

4. Declare and define a **`struct`** type that has a single **`int`** member variable and a default constructor that initializes it to **`0`**. Declare and define another **`struct`** type that inherits from the first one via public inheritance, has no member variables besides the one it inherited, and has a default constructor that assigns a non-zero value (e.g., **`-1`**) to its inherited member variable. Use typedefs to declare another name for each of these struct types, and in your main function declare variables of each of the struct types and of each of the type names introduced by the typedefs. Print out the value of each of those variables as soon as it is initialized, and also try all different possible combinations of assigning one of the variables to one of the others, and for those that are legal print out the values of the variables before and after each assignment. As the answer to this exercise please describe what you observed in terms of (1) the default construction semantics of each of the types, (2) the compatibility of the different types with respect to the assignment operator, and (3) whether or not any of the assignments that were legal was able to change the values of any of the types (and if so which ones).

5. To both the base and derived struct types from the previous exercise, add a virtual method named **`print`** that takes no parameters and prints out the name of the class and the value of the **`int`** member variable to the standard output stream. In your main function declare objects of each of the base and derived types, and also declare four pointer variables, two of which are pointers to the derived type and two of which are pointers to the base type. Try to initialize each type of pointer with the address of each of the objects, and for those that are allowed use the pointer to call the print method (using the arrow operator and/or dereferencing the pointer and then using the dot operator). As the answer to this exercise please say which initializations were allowed and which were not, and for the ones that were allowed what output was produced by the call to the **`print`** method.

6. Repeat the previous exercise, using the C++ **`dynamic_cast`** operator to perform different dynamic type conversions and using the results of those conversions to initialize the different pointer types. Again call the print method for any initializations that you were able to make work, and as the answer to this exercise please say which initializations were allowed and which were not, and for the ones that were allowed also show the output that was produced by the call to the **`print`** method.

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

7. Repeat the previous exercise, but use different combinations of operators (including dereference, reinterpret cast, dynamic cast, and others) to print out both the addresses and values of different objects of different types (including C-style strings, integers, doubles, and objects of base and derived types) through as many different types of pointers as you can think of to try. As the answer to this exercise please describe the different combinations that you tried, which of them involved which types of implicit and/or explicit type conversions, and how those different conversions affected how the standard output stream's insertion operator (**`operator<<`**) behaved.