CSE 425 Studio: Intro to Programming Languages and their Design

These studio exercises are intended to acquaint you with one particular language (C++) and one development environment (Visual Studio 2013) as well as to explore issues that are not confined to that one language, and which set the stage for comparing programming languages based on their features.

In this studio (as in the others this semester) you will work in self-selected groups of ~3 people, and will report the results of your work in an e-mail to the course grading account (cse425@seas.wustl.edu). 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.

Please record your answers as you work through the following exercises. The enrichment exercises are optional but are a good way to dig into the material a little deeper, especially if you breeze through the required ones. After you have finished please send an e-mail with "Intro Studio" in the subject line, containing your answers to the required exercises along with your answers to any of the enrichment exercises you completed, to the course e-mail account (cse425@seas.wustl.edu) and please make sure that you get an e-mail back from the course e-mail account's auto-reply feature, as confirmation your answers were received. **Note that it is your responsibility to keep track of which studio exercises you have or have not submitted, and you should do so throughout the semester.**

PART I: REQUIRED EXERCISES

1. Form a team of ~3 people of your choice (preferably a mix of people familiar and unfamiliar with C++) and write down the names of the team members as the answer to this first exercise.

2. Open up Microsoft Visual Studio 2013, and make sure that your programming environment settings are for C++. Create a new **Win32 Console Application** Visual C++ project named **`IntroStudio`** or something similar that identifies which studio this is for. As the answer to this second exercise give (a) the name you chose for the project, (b) the directory where Visual Studio 2013 created the project, and (c) the names of all of the files that appear in the Solution Explorer window that should appear once you have created the project.

3. Modify the signature of the main function in your project's main source file (e.g., **`IntroStudio.cpp`**) so that it matches the standard portable (e.g., between Windows and Linux) main function signature for a C++ program: **`int main (int, char *[])`**. Add a single line to the body of the main function that outputs a message string (e.g., K&R's famous **`"hello, world!"`**) using the **`cout`** standard output stream object. Below the line that says **`#include "stdafx.h"`** (a required inclusion for the default C++ configuration in Visual Studio 2013 on the Windows lab machines, which must be the first non-comment line of the program) add the necessary pre-compiler and compiler directives to the top of your project's main source file to allow the symbols in the line of code you wrote to be recognized, when you try to build the program.

H**int:** you'll need to include some libraries and also open up the **`std`** namespace. After successfully building the program, open up a terminal window (you can run the **`cmd`** program from the Windows start menu to get one), **`cd`** into the directory where the executable program was built, and run it (by typing the name of the executable file at the command prompt). Say what output is produced, as the answer to this exercise.

4. In the main function before the output statement you wrote in the previous exercise, declare three integer variables and in those declaration statements initialize two of them to known constant values (e.g., 2 and 3) and initialize the third one to 0. Following those declarations, write a single statement that assigns the third variable the sum of the values of the first two. Modify the output statement so that it outputs the values of the three variables, with the name of each variable printed out ahead of its value. Build and run your updated program and as the answer to this exercise show both the new definition of the main function and the output the program produced.

5. Declare and define a **`struct`** type named **`Adder`**, with a single public member variable that is an integer (e.g., named **`value`**); a public constructor that takes a reference to a const integer as its only argument and initializes the member variable with that reference; a public default constructor that sets the member variable's value to 0; and a public function call operator, **`operator()`**, that takes a single reference to a const integer as a parameter and adds the parameter's value to the member variable (i.e., assigns the member variable the sum of its previous value and the value of the parameter). In your main function, declare a variable (i.e., an object) of type **`Adder`** that takes the third integer variable (from the previous exercise) as an argument to its constructor. Replace the statement that assigned the sum of the values of the other two integer variables to the third variable, with a sequence of invocations of the function call operator on the **`Adder`** object for each of the other integer variables (hint: the syntax is just like calling the **`Adder`** object as though the variable name (not the variable type) were the name of a function, and passing in one of the variables each time you do that). Modify the output statement so that instead of printing out the name and value of the third integer variable from the previous exercise, it prints out the name and value of the **`value`** member variable of the **Adder** object. Build and run your updated program and as the answer to this exercise again show both the new definition of the main function and the output the program produced.

6. Declare a **`vector`** (or other STL container) of integers in your main function, and push back a number of integer values into it. Comment out previous code that used the **`Adder`** object, and instead use the STL **`for_each`** algorithm to apply the **`Adder`** object (the algorithm uses the object's function call operator) to each integer in the vector, which will accumulate the sum of the integer values in the container. Assign the result of the call to **`for_each`** back into the **`Adder`** object (**note:** the library enables C++11 move semantics, for efficiency). Add a statement (e.g., using STL iterators and the STL copy algorithm) or multiple statements (e.g., using a for loop) to output the names of the integers in the container (e.g., the container name and the position in the container) and their values, and the name and value of the member variable of the **`Adder`** object. Build and run your program and as the answer to this exercise again show both the new definition of the main function and the output the program produced.

H**int:** you'll need to include the container's library and also the **`<algorithm>`** library.

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

7 and beyond. Repeat some of the exercises above in another programming language with which you are familiar. As the answer to this exercise please say what other language(s) you tried, describe which parts were readily portable to the other language(s) and which were not, and what those observations say about the different features and/or design criteria on which the different languages you tried were based.