CSE 425 Studio: Control Flow I

These studio exercises are intended to extend your familiarity with ideas and techniques for basic (statement, sequence, and selection based) program control flow, 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 e-mail your answers to the required exercises, and to any of the enrichment exercises you completed, to the cse425@seas.wustl.edu account, with the subject line "Control Flow Studio I". The enrichment exercises offer 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. **Initialization.** Open up Microsoft Visual Studio 2013 and create a new project for this studio (for example, named **`ControlFlowStudioI`** 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[])`**.

Declare (but do not initialize) a number of variables of type **`int`**, some non-static and local to the main function, and some static both within and outside the main function. After the last of the declarations in the main function, use separate output statements to print out the values of all those variables (declared both within and outside of the main function) to the standard output stream.

Build and run your program, and if any statements cannot be compiled due to using uninitialized variables, comment them out so the other statements can be compiled. As the answer to this exercise please describe what you saw in terms of the default initialization semantics of those different groups of variables.

3. **L-Values and R-Values.** Add initialization values to each of the variables you declared in the previous exercise, and then write a number of simple assignment statements (not involving any of the other assignment operators besides just **=**) involving those variables, integer arithmetic operators, and numeric integers like 3 and 7, etc. in different combinations so that each variable appears at least once on the left hand side of a statement, and at least once on the right hand side of a statement. Just after each assignment statement, print out the names and values of the variables that were involved in it.

Build and run your program, and as the answer to this exercise please (1) show your code and the output of your program, and (2) for at least one of the variables please indicate all the places where it was used as an l-value, and all of the places where it was used as an r-value.

4. **Sequencing.** Reorder the lines from the code in the previous exercise so that different assignment statements involving one or more variables in common (including some statements where a variable appears on the left hand side of the assignment and other statements where the same variable appears on the right hand side of the assignment) appear in different orders, but are each still followed immediately by the statement that prints out the values of the variables involved in them.

Build and run your program, and as the answer to this exercise please describe the different data dependences that were seen between the variables in the assignment statements.

5. **Selection.** Reorder the assignment statements (while again making sure that their corresponding print statements immediately follow them) so that all the assignment statements with the same variable on the left hand side are consecutive. That is, they should look something like

```
x = ...

x = ...

y = ...

z = ...

z = ...

z = ...
```

Then, add **`if`**…**`else`** statements so that after each assignment statement if the value of the variable was just assigned is an even number it continues executing statements with that same variable on the left hand side (if there are any left), and otherwise executes the next assignment statement involving another variable on the left hand side. Build and run your program with different initial values for the variables whose values are being tested, and as the answer to this exercise please describe how the program's behavior changed with different initial values for the variables.

6. **Selection Operator.** Repeat the previous exercise, but use the ternary conditional operator (**`? :`**) that is provided by C++ instead of using **`if`**…**`else`** statements. As the answer to this exercise please indicate whether or not you saw any differences in the behavior of the program compared to in the previous exercise, and if so what they were.

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

7. Rewrite the assignment statements from exercise 3 with various other assignment operators such as **`+=`**, **`*=`**, etc. where possible, i.e., where the same variable is on the left and right of the assignment and the statement can be re-written to have the same meaning.

8. Use modulus arithmetic (**`operator%`**) on a single variable to convert the **`if`**…**`else`** logic of exercise 5 that involves it, into switch logic. Build and run your program, confirm that the same output is produced in this exercise as in exercise 5, and as the answer to this exercise please show your code.