# Studio Session on C++ Functions

These studio exercises are intended to give you more experience with basic features of C++ functions (including pass-by-value vs. pass-by-reference vs. pass-by-const-reference, iterative vs. recursive function structure, declarations and definitions, and default function argument values), and to give you experience using those concepts and techniques within the Visual C++ environment.

As before, students who are more familiar with the material are encouraged to help those for whom it is less familiar. Asking questions of your professor and teaching assistant (as well as of each other) during the studio sessions is highly encouraged as well.

Please record your answers you work through the following exercises. After you have finished, please submit your answers to the required exercises and to any of the enrichment exercises you completed to Blackboard or our OJ system (if applicable).

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.

**PART I: REQUIRED EXERCISES**

1. Open Visual Studio and create a new create a new Visual C++ Empty Project for this studio. Throughout these exercises you will replace the code for previous exercises with new code – one way to do this without losing what you've already done (so you can refer back to it later if that's useful) is to comment out the old code. Add a new header file and a new source file to your project. These are where you will declare and define (respectively) the functions you will develop during this studio. Declare (in your new header file) and define (in your new source file) a function that takes an unsigned integer by value as its only argument, uses a loop to compute an unsigned integer that is the factorial (please see http://en.wikipedia.org/wiki/Factorial) of the argument that was passed to it, and returns that result by value. Call this function from your main function with several different input values and print out (and save a copy of) what the function's inputs and outputs were. Rewrite the function so that instead of using a loop the function calls itself recursively to compute the factorial. Run the same set of trials using the new recursive version of the function, and as the answer to this exercise (1) show the inputs and outputs for both versions of the function, and (2) indicate whether or not they were the same.

2. Rewrite the function from the previous exercise so that (1) instead of returning an unsigned integer by value it has a void return type, and (2) in addition to taking an unsigned integer by value (which contains the number whose factorial is being computed) the function also takes a second argument of type unsigned integer by reference, and puts the result of the factorial computation into that second argument. In your main function, declare an unsigned integer variable to hold the result, and call the function as you did in the previous exercises, but passing that variable as the second argument to the function. Please describe how you modified your function, and the value of that variable that was produced by each call to the function, with different inputs as in the previous exercise.

3. Modify your solution to the previous exercise so that for the second argument it takes a pointer to an unsigned integer by value, instead of taking an unsigned integer by reference. As the answer to this exercise, please explain what you had to do to make your solution work correctly.

4. Declare and define another function that takes two unsigned integers as arguments by value and returns the result of raising the first one to the power of the second one (exponentiation). Build and run your program with the main function using different values to call that new function, to verify it works correctly. Then, as the answer to this exercise, answer the following questions and for each one explain why you saw what you saw. (1) What happens (and why) when you declare a default value for the second argument but not the first one, and call the function from main with only one value? (2) What happens (and why) when you declare a default value for the first argument but not the second one, and call the function from main with only one value? (3) What happens (and why) when you provide default values for both arguments, and call the function from main with no values?

**PART II: ENRICHMENT EXERCISES (optional, feel free to skip some and do ones that interest you).**

5. Declare and define a function that takes an argument of type **`size_t`** by value as its first argument, and takes an array of unsigned integers by value as its second argument. In your main function, declare an array of unsigned integers, initialize the values of all positions in the array to some specific numbers, and then call the function with the number of positions in the array as the first argument and the array name as the second argument. As the answer to this exercise, answer the following questions and explain why what you saw happened. What happens if the function tries to modify the values stored in the array? What happens if you change the function declaration and definition to pass an array of **const** unsigned integers, and again have the function try to modify the values stored in the array?

6. Repeat the previous exercise, but instead of declaring and defining the second argument of the function to be an array of unsigned integers (in the first part) or an array of const unsigned integers (in the second part), try declaring and defining it to be a pointer to an unsigned integer, or a pointer to a const unsigned integer, or a const pointer to an unsigned integer, or a const pointer to a const unsigned integer. As the answer to this exercise, please describe briefly what happens in each of these cases, and why it happens that way.