Last updated 25 October, 2014

Assignment 5:

Please see Submission Checklist (below) for submission requirements.

It is time to bring together some concepts we have been learning!

Remember that you can reuse code to solve these problems!

Remember that there are recursive and non-recursive solutions to most of the things we will do, but that some problems will be easier to solve with recursion while some will be easier to solve with iteration (loops).

(10 points) Programming Styles and Convention.
http://classes.engr.oregonstate.edu/~jessjo/CS161/OSU_IntroCodeStandards_v1.1.pdf

(30) Remember to submit your report!

Which includes:

- Understanding: a description of what you understand is being asked of you for the assignment.
- Design: an outline in **pseudocode or a drawing** as to how your code will be implemented. This must include **both** the exercise and project component. The design section is to be done before you start coding.
- Testing: a description of the tests you will implement to ensure your code works along with the actual testing. This must include **both** the exercise and project component. The testing section is to be done before you start coding. **(Make a table with three columns named Input, Expected Output and Actual Output. Discuss any discrepancies.)**
- Reflection: a discussion of the problems you encountered and how you solved them.

(20) Exercise components: **for each of the following exercises you should write code in main that asks the user for the parameter values, passes them to the function, and then displays the result to the user.**

Discuss on Discussion Board: why might I ask you to solve so many of these problems more than once?

1. (4) Passing by reference: write a simple **function** that acts like a random number generator with the following behavior:

   File must be called: randFun.cpp

   a. its definition looks like the following: **void rand_int(const int &min, const int &max, int &val)**,

   b. it accepts references to two int values and a reference to a third value (one for the bottom of the random range, one for the top of the random range, and one to store the final value), generates a number within that range, and then sets the value of the third number to that generated value,

   Discuss on Discussion Board:
   i. Why might I use const in the signature of the function?
   ii. Why might I want to do this with references to values instead of copies of values?
   iii. Can I return two values from a function? Is there a way to alter multiple values passed into a function call?

2. (4) Passing by reference: write a simple **function** that accepts two references to string and returns whether they have the same contents with the use of a loop to check each character individually.

   File must be called: refFun.cpp

3. (4) Passing by reference: write a function that takes three integer parameters **by reference.** The function should rearrange the parameter values so that the first parameter gets set to the smallest value, the second parameter gets set to the second smallest value, and the third parameter gets set to the largest value. For

example, given the variable assignments a = 30; b = 10; c = 20; then the function call sort(a,b,c) should result in a =10, b =20, and c =30. Or if you start with those same variable assignments, but call sort(c,a,b), this should result in c=10, a=20 and b=30.

File must be called: sortFun.cpp

4. (4) Recursion: write a recursive function that takes one parameter *n* of type int and returns the *nth* Fibonacci number. The Fibonacci numbers are as follows: F_0 is 1, F_1 is 1, F_2 is 2, F_3 is 3, F_4 is 5, and in general $F_{i+2} = F_i + F_{i+1}$ for i = 0, 1, 2, … (you get the next number by adding the previous two). The naive way of writing this function is inefficient because it ends up calculating the same numbers multiple times. It's fine for you to do it this way for now, since you're just learning recursion. If you get done with your assignment early, you might try to figure out a recursive approach that doesn't have this problem.

   File must be called: recFun.cpp

   Discuss on Discussion Board: later elements rely on earlier elements, so how many elements do we need to have provided to start this recursive process?

5. (4) Recursion: write a recursive function with the following behavior:

   File must be called: hailstone.cpp

   A hailstone sequence starts with a given integer. If that integer is even, then you divide it by two to get the next integer in the sequence, but if it is odd, then you multiply it by three and add one to get the next integer in the sequence. Then you use the value you just generated to find out the next value, according to the same rules. Your recursive function should take as input the starting integer and count how many steps it takes to reach 1 (technically you could keep going 1, 4, 2, 1, 4, 2, etc. but we will stop when you first reach 1). If the starting integer is 1, the output should be 0, since it takes no steps to reach one (we're already there). If the starting integer is 3, then the sequence would go: 3, 10, 5, 16, 8, 4, 2, 1, and the output should be 7.

   Discuss on Discussion Board: Why do you think writing recursive code initially feels more challenging than writing iterative code?

6. **Be sure to test that each of your sections of code work correctly in a variety of cases.** After you think your code is good, trade it with another person and try to break their code (share with the other person what you find works well and what does not).

(40 points) Project components:

Remember before you begin coding to draw or write out what this program will do. Also, remember to think of what tests you could use as you design and code up your solution.

Discuss on Discussion Board: what do your designs looked like, how did you go from design to program, what did you do to make sure your program was working correctly (**consider letting someone else test your compiled program to see if they can break it**).

File must be called: recConvert.cpp

Your main function will allow the user to choose between 1) converting a binary number to a decimal number; 2) converting a decimal number to a binary number; or 3) exiting the program. This menu should be in a loop so the user can convert as many numbers as they wish before exiting. When the user selects either of the first two options, your program will ask the user for a number to convert. It should first check to make sure the input is valid, then call a function that returns the desired value, and then print it for the user (the printing happens in main, not the conversion functions). **Both conversion functions must be recursive**. Represent binary numbers as string values. Do not use any number base conversion functionality that is built into C++. Checking for a valid binary

number should be simple, since there are only two allowable characters. One way of validating values that aren't strings is to read them in as strings anyway, check to make sure the characters form a valid input, and then convert the string value to another type. For example, if *num* is a C++ string, then the following code converts it to a C-style string (with c_str) and then converts that to an int (with atoi):

```
int val = atoi(num.c_str());
```

You don't have to use that specific method of input validation, but there it is if you want it. Of course if the user enters invalid input, you should print a message to that effect. Then (instead of looping till they enter valid input) just return back to the main menu. Don't worry about clearing the screen to re-display the menu – you want the user to be able to read the message that their input was invalid.

- (6) menu loop works correctly
- (10) input validation works correctly
- (12) decimal to binary conversion works correctly
- (12) binary to decimal conversion works correctly

Remember to submit your **report** and **source files** to TEACH before the end of Sunday.

Remember to keep discussion going!

Submission Checklist:

makefile

Exercises (20 pts):
- randFun.cpp
- refFun.cpp
- sortFun.cpp
- recFun.cpp
- hailstone.cpp

Project (40 pts):

- recConvert.cpp

Report, ***in PDF format***, must address these four sections (30 pts):

- Understanding
- Design
- Testing
- Reflection

The implementation part of the assignment is the .cpp file you submit.