CIS220 In Class/Lab 1:

Due Wednesday night at midnight. Submit all files through Canvas (45 pts)

Prework: Install Eclipse + CDT (or, as an alternative, Netbeans). Follow the instructions on my web site. Note that if you don't still have java installed on your computer, you will have to install it as well. Both Eclipse and Netbeans require Java.

From D. Sorenson: On Macs, you may need to open a terminal window and type in the command, "xcode-select --install".

****

Formatting Rules:

You want your code to be as easy and clear to read as possible for your partner, other programmers, your TA, and me. *<u>We reserve the right to take off points for code that works but is difficult to read.</u>*

Programmers all use pretty much the same formatting rules, making code easier to follow. At this point I expect your code to be formatted properly.

Function comments (defined below) must be written before you write the function code. We will not help you with code unless you have the function commented properly

Comments: ALL code should be commented.

Comments you should include:

1. Every file should start with your name, possibly your partner(s)'s name, the date, and what the file holds in general.
2. A.Every function should state what type and what purpose the input parameters have, what is being returned from the function (again, both type and purpose), and the function's purpose. The goal of these comments are so that someone who never sees your function can call it properly and use the returned result properly.

   b. if function definitions and declarations are in the same file, you may include comments for the function definitions alone. If, however (and we will do this in future labs) the function declarations and function definitions are in different files, comments for the function should be in both places. The function definition comments should have a bit more information about the guts of the function (i.e., how it actually works) whereas the function declaration comments will probably be shorter and focus on what the input values and results are.
3. If there is any tricky, unusual, or difficult to read code within your function, that should be commented. As much as possible, though, with good variable names and clean code, your code should be fairly readable without comments.
4. While not required, inside the function it often helps to indicate what a closing squiggly bracket closes, e.g.,:

```
int main() {
        int x = 3;
        if (x > 5) {
                cout << "large" << endl;
        } // if (x > 5)
} // main
```

Formatting: (aka indenting and where to put the squiggly brackets) When coding, I expect you to use a systematic form. This means that the opening squiggly may or may not be on the same line as the function or condition it opens, but the closing squiggly bracket should be on its own line and vertically aligned with the beginning of the code it opened, e.g.,

```
int main() {
        …
} //main
```

Or

```
Int main()
{ …
}//main
```

Everything inside the squiggly brackets should be indented consistently, e.g.,:

```
int main() {
        int x=3;
        int y=10;
        for (;x<y;x+=2) {
                cout << x << endl;
        } //for
} //main
```

DO NOT place more than one programming instruction on the same line (yes, you can do it. Don't do it). E.g., the following is not formatted properly, although it will compile:

```
if (x < 3) { cout << "hi" << endl; } //bad!
```

Equally, the following is bad form:

```
int x = 3; int y = 4; int z = x + y; cout << z << endl; //bad bad bad!!!
```

Naming:

Function names and variables should start with small letters.

If the function or variable name consist of more than one word, there are two formats that are conventional:

1. Camel case, e.g., dataStructures(), getValues() etc. or
2. Underscore, e.g., placing a '_' between words, e.g., data_structures(), get_values(). Whatever of these you prefer is fine, but be consistent.

Boolean Functions: If a function returns a Boolean variable, it is convention to name the function with the first word of 'is', e.g., isDivisibleByThree() (or is_divisible_by_three()) for a function that returns a Boolean value indicating whether a number is divisible by 3.

Constants can either be defined with the first letter being capitalized or the entire thing being capitalized, e.g.,

```
#define Pi 3.1415

#define MIN(X,Y) ((X) < (Y)?(X):(Y));
```

It is more typical to capitalize the entire constant.

Please follow these formatting conventions when writing code for this class. More formatting conventions will follow.

Equally, your TA has the right to require specific formatting rules in order to make his/her grading experience easier. Pay attention to these rules!

****

Programming Problems:

Note: At this point in the semester you will be using one file. The file will start with comments including you and your partner's names, your TA's names, the date, and the lab (lab 1). It will then have your include files, and then it will have your function declarations (with the function definitions below your main function), followed by your main function, whose sole purpose is to call your functions for each problem, and then under the main function you will have your function definitions, in the order in which they occur for this lab.

So your file should look something like this (date and time don't have to look exactly like this – I know eclipse automatically inserts the date in a different way and that's fine):

```
/* Debra Yarrington
 * TA's name
 * 2/14/17
 * This file contains functions for lab 1.  The functions aren't necessarily related
 * in any way other than that they are required for lab 1.
*/

#include <iostream>
#include <stdlib.h>
using namespace std;

int func(int k[]) ;  //function declaration

int main() {
	…  //call functions here
	return(0);
}

int func(int k[]) { //function definition
	int a = 0;
	for (int i = 0; i < 4; i++) {
		a += k[i];
	}
	return a;
}
```

****

Note: For each problem except problems noted , the Collatz conjecture problems, and the leap year problem, call the function created 3 times in your main function with 3 different input parameters (with a comment next to it of the expected output parameter.

After the 3 test calls for a particular function, print out a line of
****

and then Problem x on the next line, so the TA can clearly see where each problem's output starts and stops.

****

Problem 0: Say "hello" to your partner and introduce yourself. Exchange emails (at the least). Now tell your partner about your strength as a partner (e.g., you meet deadlines, you're good with details, you're good at taking directions, something like that,). Tell your partner your weakness as a partner (you procrastinate, you have a hard time concentrating on things that bore you, etc.). Tell your partner what you think is your strength as a programmer. Tell your partner your weakness as a programmer. Briefly think about and discuss how you can work with those strengths and weaknesses together (this is something that may not be resolved in one meeting, but it is definitely something you should think about and plan around).

Problem 1. (2 pts) In your main function, print out "Hello World!". If you can get this to work, we'll know you've gotten a c++ compiler to work correctly on your computer.

Problem 2. **(2 pts)** Write a function that takes no input parameters and returns no values and prints out a line of stars. This is the function that will be called after you've called each of the below problems with your 3 test cases, resulting in the line of stars being printed out to separate each problem.

Problem 3: (3 pts) Write a c++ function that takes as input parameter an integer. The function uses a while loop to determine whether that number is a prime number or not, and returns True if it is, and False otherwise. Test 3 times by calling this function from main with different numbers, and use cout<< in main to print out the results.

Problem 4: (3 pts) Write a function to calculate the sum 1+2+3+...+300. Display the total after every 20 terms by using an if statement to check if the current number of terms is a multiple of 20. (Note: this function only has to be tested from main once)

Problem 5: (3 pts) Write a function that will figure out how many terms in the sum 1+2+3+... it requires for the sum to exceed 100,000. This function should use a while loop and should return that number, printing it out in main using cout<< (Note: this function only has to be tested once).

Problem 6 (4 pts): Write a c++ function that takes as input an integer. It then prints out (using cout <<) the multiplication tables between 1 and 12 and that integer using a for loop. (Note that this must use a for loop!)

Problem 7 (4 pts): Write a c++ function that loops from 1-12 and uses the function in Problem 3 to print out all of the multiplication tables between 1 and 12. (FYI, 12 separate statements instead of a loop will definitely be penalized in a really nasty way).

Problem 8 (4 pts): The Collatz Conjecture states that, given any positive natural number, you will always eventually reach 1 if you follow the following formula: if the number is even, divide by 2, and if the number is odd, multiply by 3 and add 1. Continue the process indefinitely or until the number is equal to 1. This problem has never been proven, and someone did actually prove that a generalization of the problem is unprovable. That said, no one's found an exception either.

Write a program that asks the user to input a natural positive number using cin >>. Then follow the above formula until the number is equal to 1. Count how many times you loop, and write that out to the console window using cout <<. Return the count of how many times the Collatz Conjecture had to loop and write that to the console in main.

Problem 9 (4pts): To determine whether a year is a leap year, you use the following rules (I googled this):

1. If the year is evenly divisible by 4 go to step 2. Otherwise go to step 5:
2. If the year is evenly divisible by 100, go to step 3. Otherwise go to step 4:
3. If the year is evenly divisible by 400, go to step 4. Otherwise go to step 5:
4. The year is a leap year (which, oddly, means it has an extra day, or 366 days).
5. The year is not a leap year (which means it has 365 days

Now you know how to calculate a leap year. I'm thinking step 3 isn't going to come in handy for most of us. That said, write a function that calculates all the leap years for the next 400 years and prints them out (starting with 2017, which won't be printed out).

This problem should return nothing. It should print out the leap years as it loops.

Problem 10 (5 pts): Write a function that takes an integer as an input parameter, and then prints out an x of stars such that, for example, if the number is 5, the following would be printed out:

```
*       *
 *     *
  *   *
   * *
    *
   * *
  *   *
 *     *
*       *
```

Note that if the input parameter is an even number, round up to the next odd number. This should use a nested loop. Test this function with 3 different integers.

Problem 11 (4 pts): Copy the program in problem 4 and modify it so that it is a function that takes as input parameters two integers, assuming the first is smaller than the second number. Continue modifying the function so that for every number between the first number and the second number (assume the first number is less than the second number), the number is printed out, and then it is tested with the Collatz Conjecture. If the loop results in a 1 the program should print out, “Collatz Conjecture is still working”. (Now, if it doesn’t, the Collatz conjecture will loop infinitely and your program will hang . You should write down the numbers you sent in, because you just did something really amazing and found a number that doesn’t work with the Collatz conjecture. Or you did something wrong. You decide which is more likely). This function should return nothing.

Problem 12 (3 pts): Write a function that is a recursive version of problem 3:

Problem 13 (4 pts): Write a function that is a recursive version of problem 6

To Turn In: One of you should turn in the .cpp file (with the names of you and your partner) containing the main function containing test cases for each of the functions, followed by each of the functions. The other should enter in canvas just the name of your partner.