LAB #4
Design, Loops, & Errors/Debugging

Remember, if you need to get Lab #3 graded, you need to show your **lab to the TAs within 10 minutes of getting to lab**, and you and your partner **will not receive lab credit if you do not get checked off** before leaving each lab. Once you have a zero on a lab, then it cannot be changed because we have no way of know if you were there or not!!!

Reminder: All of our labs involve paired programming. You do not have to keep the same partner for each lab, but you MUST work with someone in each lab!!! **First, find a partner for this lab.** It can be the same partner from the previous lab or a different partner.

New Labs: We are going to change the way we do labs by providing relevancy through videos and using larger group activities to encourage design, while utilizing a broader set of strengths. Each lab will begin with a 10-15 minute video shown on the TV by a TA, followed by a large group activity. The group activity requires design, input from everyone, and no computers!!!!

If you are feeling unsure about your decision as a CS major or programming, i.e. this class☺, then please take time to fill out the "Comfort Survey", under your class (not lab) Blackboard documents.

Learn more about flowcharts and pseudo-code:
http://www.schooltube.com/video/d4b724178f5b49dabc5f/Algorithms%20in%20pseudocode%20and%20flow%20diagrams

Practice Designing as a Group (NO Computers Allowed!!!) (5 pts)

We need to get into a bigger group before you begin your paired-programming. Get into groups of 10 for a 30 person lab, i.e. 5 (or less) pairs in 3 different groups. Each group will have a dedicated TA as a project leader. The role of the TA is to make sure all members in the group are participating and that everyone understands the requirements of the problem being solved. In addition, your group might want to dedicate someone with good handwriting to capture your thoughts and design. Each group will begin by writing a flowchart for the solution, and then, write the pseudocode.

Problem Statement: With the super bowl coming up this weekend (and the new star wars at the end of the year!), we need to be able to read roman numerals. **Design a program that converts a roman numeral into a decimal value and a decimal value into a roman numeral.** Below are the specific rules and instructions for doing this.

Possible Roman Numerals:

1	5	10	50	100	500	1000
I	**V**	**X**	**L**	**C**	**D**	**M**

When a symbol appears **after a larger** symbol it is **added.**

- Example: VI = V + I = 5 + 1 = 6

If the symbol appears **before a larger** symbol it is **subtracted**. You can only use I, X, and C as a subtractor in front of a bigger numeral that is not more than 10 times bigger, i.e. I before V and X, X before L and C, and C before D and M are valid.

- Example: IX = X - I = 10 - 1 = 9

Don't use the same symbol more than three times in a row.

- Example: IIII is not how you represent 4, IV = V - I = 5 - 1 = 4

You should walk yourself through your design with the super bowl number **XLIX**
(L-X) + (X-I) is 40+9 = 49

Show your project manager (TA) how the super bowl number can be converted to and from roman numerals using the flowchart and pseudocode you designed. **After your design is approved by your project manager (TA), you can begin with the rest of the lab.** In lab #5, each pair will code their group's solution, and we will look at how much code differs among the pairs within the same group using the same design versus among pairs in other groups.

INDIVIDUAL WORK (Computers Allowed!!!!), Due by lab #5
This is to ensure that all students get a .vimrc file and understand errors and debugging.

VIM Exercises (2 pts)

1. For this part of the lab, you will create a .vimrc file that will help you develop your C++ programs using Vim. First, we need to create a simple .vimrc file in your home directory on the engr server.

   ```
   vim .vimrc
   ```

   In this file, you can insert the following lines so that it makes working with vim easier. Comments are prefaced with a quotation mark, ".

```
filetype on
filetype plugin on
filetype indent on
autocmd FileType c,cpp set cindent   "Allows for c-like indentation
set sw=3   "Set the sw for the auto indent to 3 spaces
set mouse=a "You can use your mouse to select a line, scroll, etc.
set number  "Show the line numbers on the left

"Change the color of the text and turn on syntax highlighting
"colorscheme darkblue
colorscheme evening
syntax on   "Turn on syntax highlighting
set showmatch  "Show matching braces
set showmode   "Show current mode
```

There are many more commands you can insert in this file, and here is a reference guide to some of these: http://vimdoc.sourceforge.net/htmldoc/starting.html

Debugging (3 pts)

2. Here is a buoyancy program that has **syntax errors** and won't compile. Your job is to fix all the syntax errors☺ You can use the –Wall option with the compiler to turn on **all warnings**: **g++ -Wall errors.cpp –o errors** You can copy and paste the code below or download from here: errors.cpp

```
#include <iostrem>
#include <cmath>

#define WATER_WEIGHT 62.4;  //Defines the specific weight of water
using std::cout;
using std::cin;

int main {
   //Variables needed to calculate buoyancy
   float radius, bforce, volume;

   //Prompt user for weight and radius of sphere
   cout << "Please enter the weight (lbs): "
   cin >> weight;

   cout >> "Please enter the radius: ";
   cin << radius;

   //Calculate the volume and buoyancy force
   volume = 4./3. * PI * power(r, 3);
   b_force = voluem * WATER_WIEGHT;

   //Determine if the sphere will sink or float
   if(bforce >== weight) {
      cout >> "The sphere will float!\n";
   else;
     cout << "The sphere will sink!\n";
   }
   return;
```

3. Below is code that has **logic errors**. Logic errors are usually caught by **using print statements** to make sure the contents of variables are what you think they are, printing indicator messages, and **commenting out code** to localize the error.

   In the following code, go through the code and systematically fixing the logic errors by inserting print statements to check the contents of the variables. You might want to use comments to narrow the scope of your code to help find the errors. You can copy and paste the code below or download from here: prime_debug.cpp

```
#include <iostream>

#define PROMPT "Please enter a whole number:"
#define NOT_PRIME "The number is not a prime number./n"
#define PRIME "The number is a prime number./n"
#define DONE 0          /* ends successful program */
#define FIRST_FACTOR 3  /* initial value in for loop */

using std::cout;
using std::cin;

int main(){
    int i;       /* loop counter */
    char number; /* number provided by user */

    cout << PROMPT;  /* promt user */
    cin >> number;  /* wait for user input */

    /* Prime numbers are defined as any number
     * greater than one that is only divisible
     * by one and itself. Dividing the number
     * by two shortens the time it takes to
     * complete. */

    for(i = FIRST_FACTOR; i < number/2; ++i)
       if( number/i == 0 ){     /* if divisible */
          cout << NOT_PRIME << number; /* not prime */
          return DONE;        /* exit program */
       }

    // if not divisible by anything, then it must be prime
    cout << PRIME << number;
    return 0;     /* exit program */
}
```