CSE 20212 Fundamentals of Computing II
Spring 2016

Lab #2 handout (Week of February 1)

Objectives

1. Get more experience debugging your code and using text editors
2. Use overloading, this pointers, const and friend functions
3. Develop a class to store Rational numbers
4. Have fun!

Prerequisite reading

Chapter 11 in D&D
Chapters 12 and 13 in D&D (next week)

In-lab activities

Part 1: Overloading operators for Connect4 (in lab)

1. Report to lab **on time.** Attendance will be taken at the scheduled lab time.

2. Important! Copy the files you submitted for Lab #1, Part 2 into a **** new **** directory in your local AFS space (e.g., lab2). Please ask the TA if you are unsure.

3. Today, we want to practice overloading NOT copy constructors or the autowin function. For this reason we are starting from Part 2 for these later steps here.

4. To start, we'll add a "+= operator that calls the addDisc() function you developed last lab for Part 2 in the copied files: in your C4Col.h file, add the following prototype: C4Col operator+=(char); In general we'll follow examples similar to those in Chapter 11 if you need a reference, so please bring a text to lab if you can. In the .cpp file for C4Col, implement the operator+= function by calling the previously developed addDisc inside. Finish the function with: return(*this). What is this last line of code (the return) doing? If you are unsure, please ask a TA or your instructor to explain during lab.

5. <u>Please start Part 2 if you complete Part 1 prior to the end of lab.</u>

Part 2: overloading input and output operators of Connect4

1. Implement an overloaded << operator as a global friend function to replace the display() your created in Lab 1, Part 2. The easiest way, which we will provide a special exception today for, is to copy what you implemented for the C4Board::display() function as the new friend function (removing the "C4Board::" of course). Replace all instances of cout with the generic stream provided to the

friend function. If you are unsure of how or what this does, please ask a TA or your instructor. NOTE: Friend functions have access to the data members of a class just like a C- style struct. So, for example, your for loop could look something like this:

```
for (int i = B.board[0].getMaxDiscs() - 1; i >= 0; i--) {
```

This violates some of the premises of data hiding and encapsulation, but overloading the << and >> operators are one of the few exceptions where friend functions should be used in this manner.

2. To enable cascading of the "<<" operator you need to return the output stream given to your new friend function of type ostream. Do this by adding this line to the end of your friend function: "return (*output*);" replacing *output* with whatever you called the ostream & parameter in your friend function.

3. Finalize your fancy new Connect4 by dereferencing the "this" pointer as follows instead of display(): "cout << *this << endl;" Why do you think this works?

Part 3: Rational (fraction) class for a calculator Mark III

1. Create a simple class interface called Rational for performing arithmetic with fractions. Your class must achieve the following characteristics:

   - The "default" Rational is 1

   - Any denominator of 0 should display an error message

   - The class should always store Rational in reduced form. In other words, creating an object like this would store "3":
     Rational fraction(6, 2);

   - Must have an overloaded << operator such that it will display the fraction as either (x/y) or as a floating-point format (up to you).

   - Add an "interactive" mode that will
     - ask the user for a numerator, a denominator for at least two Rationals
     - compute three mathematical functions using four different operators which you will define in the Rational class, of your choosing, and display the result to the user. You are encouraged to use your creativity here and pick at least one equation of interest to you!
     - "co-opt" one of the additional operators provided by C++ for at least one mathematical function beyond add, subtract multiply and divide.

NOTE: For full credit you must pass Rational objects, and return the new Rational that you calculate. Please refer to the text or see us if you have questions.

6. Please discuss Rational (not Connect4) in the report you submit with this lab.

Handing it in

Please read and follow the general lab guidelines available on the course website:
http://www.cse.nd.edu/courses/cse20212/www/labs.html

If you have any questions about these or the grading rubric, please use Piazza.

Coder challenge! (prime number edition)

Prime numbers are natural numbers that have exactly two distinct natural number divisors: 1 and the prime number. Primes are important in several routines you may use later in your time at ND, for example, in public-key cryptography. Primes are also cool for a bunch of mathematical reasons, for example, the unproven twin prime conjecture states that the number of primes that differ by two (i.e., 5 and 7; 11 and 13; 41 and 43) is infinite. The largest known prime has 13 million decimal digits. Finding large prime numbers has led to financial awards, most recently via the Electronic Frontier Foundation that awarded a $100,000 cash prize to a group that found a 12-million digit prime. Here, on our team however, we need to solve our problems in a fixed amount of time. Your mission, if you choose to accept it, is to find as many primes as possible in under 5 minutes on the machine the TAs choose to test your code on. You may use any algorithm you find online but you ** must ** implement it yourself in C++ to receive up to $10 coder dollars. Primes must be stored in a dynamically allocated array using new/delete once found. Further, any attempt will earn up to $4 coder dollars.

Whoever finds the largest prime after five minutes will receive extra $20 coder dollars and whoever finds the most primes after five minutes will receive $20 coder dollars. The maximum award is $50. Have fun!