ECE 1305 Section 001

Lab 05 – Review Lab

If you made it through the lab assignments, this should be quite straight forward. No new material, just some exercises to make sure you are ready for the exam.

Do the following problems:

Chapter 1: Problem 11 on p. 44:

Write a program that inputs an integer that represents a length of time in seconds.
The program should then output the number of hours, munutes, and seconds that correspond to that number of seconds. For example, if the user inputs 50391 total seconds, then the program should output 13 hours, 59 minutes, and 51 seconds.

Name your program "int2time.cpp"

Chapter 2. Problem 9 on P. 96 of the textbook:

The Babylonian algorithm to compute the square root of a postiive number n is as follows:

1. Make a guess at the answer (you can pick n/2 as your initial guess)
2. Compjter r = n / guess.
3. Set guess to (guess + r)/2
4. Go back to step 2 for as many iterations as necessary. The more steps 2 and 3 are repeated, the closer guess will become to the square root of n

Write a program that inputs a double for n, iterates through the algorithm until the guess is within 1% of the previous guess, and outputs the answer as a double to four decimal places. Your answer should be accurate even for large values of n.

Addition to the problem: report the absolute error once the stopping criteria is met.
Addition to the problem: report the number of loops needed for 2 or 3 values of n between 1 & 10, between 10 & 100, and between 100 & 1000. Is the algorithm faster for small numbers, large numbers, or does it seem to make a difference.

Suggestion: use cout inside the loop while you are testing/debugging. You should output the loop number, the last guess, the revised guess, the difference from the last , and the relative error (difference from last guess / correct answer. Recall that you can get the square root using the sqrt(...) function.

Name your program "sqrt_approx.cpp"

Arrays:

Write a program which calculates values of a 2^{nd} order polynomial of the form, and an estimate of the derivative (the slope) at each point.

$$y = a_0 + a_1 x + a_2 x^2$$

for ***n*** values of x between x_1 and x_2.

Your program should prompt the user for:

n
x_1 and x_2
a_0 , a_1, a_2

Assume n will be no larger than 1001. Store the x values, the y values, and the slopes in 3 arrays, then print out a table showing x, y, and the slope at each point. Print all values with 3 decimal places.

Use one loop to initialize the x array. Use a second loop to calculate the y values. Use a third loop to calculate the slopes.

For example: if n is 101, x_1 and x_2 are 10.0 and 20.0 respectively, your program should evaluate the expression at x = 10.0, 10.1, 10.2, ...19.8, 19.9, 20.0.

If a_0 , a_1, and a_2 are 3.2, -1.5, and 0.25, then the value for y, when x is 14.2, is:

y = 3.2 - 1.5* 14.2 + 0.25* (14.2)2 = 74.91

Recall that the slope is dy/dx = (y2-y1) / delta-x.

And delta x is simply the total range divided by # of steps (which is one less than the number of points.) For the example above, dx is (20 – 10)/100 = 0.1.

The comments in the code should explain how you calculate the slope for the last point. Note that you won't be able to calculate the slope of the last point,

because you won't have a next-point to use to get the slope. You may do one of 3 things for the last point:

1. Estimate the slope using the value of the slope from the next-to-last point
2. Estimate the slope using the value of the slope from the last point and the change in slope from the last 2 points
3. actually calculate n+1 points, going one step farther so you have the data to calculate the last slope (but only print out n points)

Name your program "poly.cpp"