ECE 4122/6122 Lab #2

(100 pts)

Section	Due Date
ECE 4122 - A	Oct 13th, 2021 by 11:59 PM
ECE 6122 - A	Oct 13th, 2021 by 11:59 PM
ECE 6122 – Q, QSZ, PDO	Oct 13th, 2021 by 11:59 PM

ECE 4122 & **ECE 6122** students do all problems.

Note:

You can write, debug and test your code locally on your personal computer. However, the code you submit must compile and run correctly on the PACE-ICE server.

Submitting the Assignment:

The solution to each problem should be contained in a **single zip file** and you must use C\C++ as the programming language. The zip file for each problem needs to contain all the files needed so that your solution can be compiled and run. Each zip file should be labeled Lab2_Problem**#**.zip, where **#** is problem number.

Speed matters in this assignment (see grading rubric)

The TAs will compile your code using the -O3 optimization flag when compiling your code.

Grading Rubric

If a student's program runs correctly and produces the desired output, the student has the potential to get a 100 on his or her homework; however, TA's will **randomly** look through this set of "perfect-output" programs to look for other elements of meeting the lab requirements. The table below shows typical deductions that could occur.

AUTOMATIC GRADING POINT DEDUCTIONS PER PROBLEM:

Element	Percentage Deduction	Details
Files named incorrectly	10%	Per problem.
Execution Time	Up to 8%	0 pts deducted < 5x shortest time pts deducted = (4/5) (your time/shortest time) - 4 (rounded up to whole point value, with 8 pts maximum deduction)
Does Not Compile	30%	Code does not compile on PACE-ICE!
Does Not Match Output	10%-90%	The code compiles but does not produce correct outputs.
Clear Self-Documenting Coding Styles	10%-25%	This can include incorrect indentation, using unclear variable names, unclear/missing comments, or compiling with warnings. (See Appendix A)

LATE POLICY

Element	Percentage Deduction	Details
Late Deduction Function	score - (20/24)*H	H = number of hours (ceiling function) passed deadline note : Sat/Sun count as one day; therefore H = $0.5*H_{weekend}$

Problem 1: Ant and seeds (50 pts)

(www.projecteuler.net) A laborious ant walks randomly on a 5x5 grid. The walk starts from the central square. At each step, the ant moves to an adjacent square at random (up, down, left, right), without leaving the grid; thus there are 2, 3 or 4 possible moves at each step depending on the ant's position.

At the start of the walk, a seed is placed on each square of the lower row. When the ant isn't carrying a seed and reaches a square of the lower row containing a seed, it will start to carry the seed. The ant will drop the seed on the first empty square of the upper row it eventually reaches.

1) What's the expected number of steps until all seeds have been dropped in the top row? Give your answer rounded to 6 decimal places.
2) How many total runs did it take for your answer to converge? (Or you can just do a total of 10 million runs and output the expected number)

Write a program using std::thread to solve this problem.

Your program needs to detect the number, **N**, of threads that can be run concurrently on the system being used and create N threads to answer the two questions above. Your program should then generate an output file called **ProblemOne.txt** that contains the following:

Number of threads created: #

Expected number of steps: #.######

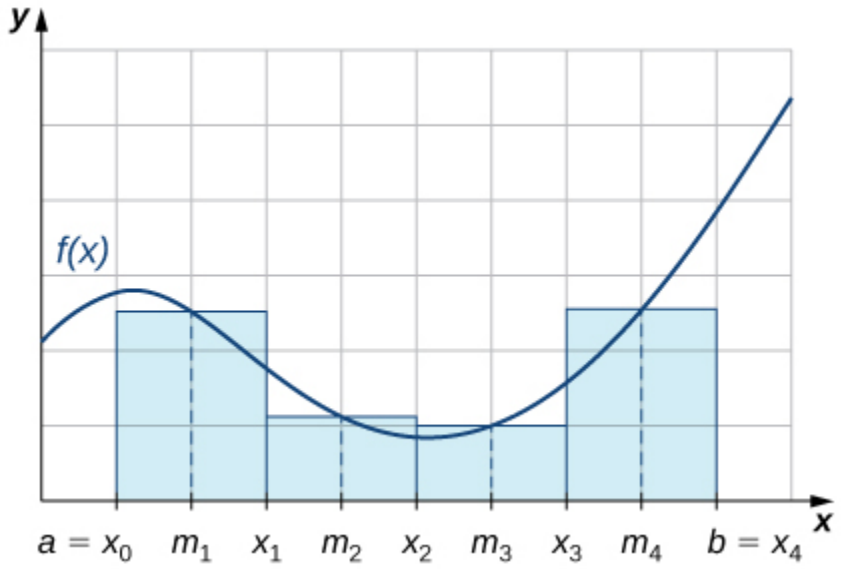
Total number of runs needed for solution convergence: #

Problem 2: Numerical Integration (50 pts)

Write a C++ program that uses OpenMP to compute the integral of:

$$\int_{0}^{\frac{\ln(2)}{7}} 14 \cdot e^{7x} dx \approx \sum_{i=0}^{N} F(x) \cdot \Delta x = 2.0$$

using the midpoint rule. The midpoint rule divides the region to be integrated into N evenly spaced segments. The value of the function is determined at each location and then multiplied by the size of the subsection. This product is then summed for all the subsections to get an estimate of the actual value.



Your program needs to take a single command line argument that is the number of subdivisions (N) to use in the calculations. Your program needs to output to the text file **Lab2Prob2.txt** the estimation of the integral, overwriting any previous results.

Make sure to use **doubles** in your program.

Appendix A: Coding Standards

Indentation:

When using *if/for/while* statements, make sure you indent 4 spaces for the content inside those. Also make sure that you use spaces to make the code more readable.
For example:

```
for (int i; i < 10; i++)
{
    j = j + i;
}
```

If you have nested statements, you should use multiple indentions. Each { should be on its own line (like the *for* loop) If you have *else* or *else if* statements after your *if* statement, they should be on their own line.

```
for (int i; i < 10; i++)
{
   if (i < 5)
   {
       counter++;
       k -= i;
   }
   else
   {
       k +=1;
   }
   j += i;
}
```

Camel Case:

This naming convention has the first letter of the variable be lower case, and the first letter in each new word be capitalized (e.g. firstSecondThird). This applies for functions and member functions as well! The main exception to this is class names, where the first letter should also be capitalized.

Variable and Function Names:

Your variable and function names should be clear about what that variable or function is. Do not use one letter variables, but use abbreviations when it is appropriate (for example: “imag" instead of “imaginary”). The more descriptive your variable and function names are, the more readable your code will be. This is the idea behind self-documenting code.

File Headers:

Every file should have the following header at the top

```
/*
Author: <your name>
Class: ECE4122 or ECE6122
Last Date Modified: <date>

Description:

What is the purpose of this file?

*/
```

Code Comments:

1. Every function must have a comment section describing the purpose of the function, the input and output parameters, the return value (if any).
2. Every class must have a comment section to describe the purpose of the class.
3. Comments need to be placed inside of functions/loops to assist in the understanding of the flow of the code.