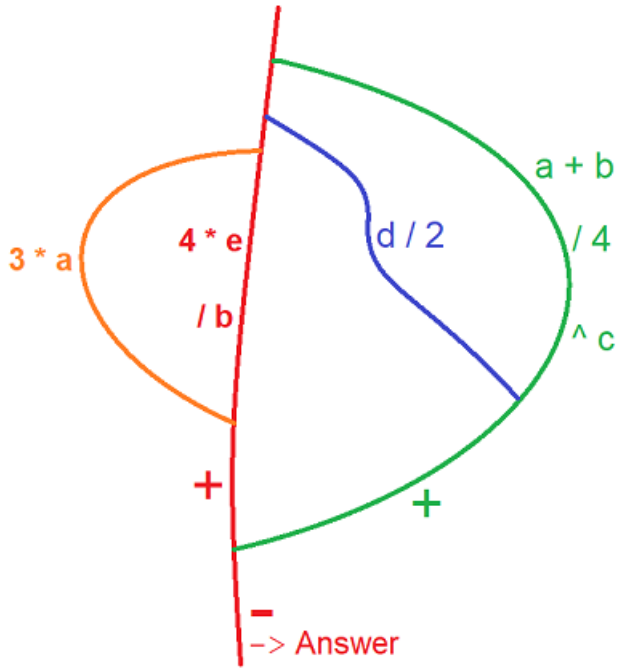
Computer Science 315
Laboratory 5 - Concurrent Threads
Due: 11:00pm Monday March 6, 2023.

In this lab, you will solve a concurrency problem that is similar in concept to the problem that was presented in Laboratory 3:

answer = (((a + b) / 4) ^ c + d / 2) - (3 * a + (4 * e / b))

In Lab 3, we achieved concurrency by forking processes to do the operations concurrently. The diagram below shows my solution for achieving optimal concurrency.



The colour coding should be interpreted as follows:

- The red line represents the parent process. (Call this P0.)
- The green line represents the first child process. (Call this P1.)
- The blue line represents the second child forked from the parent. (Call this P2).
- The orange line represents the third child forked from the parent. (Call this P3).

Here are the details of the process synchronizations:

- Processes P2 and P3 do not have to wait for anything.
- Process P1 must wait for process P2 before it can do the green **+** operation.
- Process P0 must wait for process P3 before it can do the red **+** operation.
- Process P0 must wait for process P1 to finish before it can do the last red operation (-).

In this lab, we want to achieve the same result (i.e. concurrency) using *threads*. As discussed in class, a thread is an individual path of execution within a process. All threads in a process are in the same address space, so sharing of data is much easier than with separate processes (no pipes are needed!). Also, context switches between threads are much cheaper in terms of time than context switches between processes. (On Solaris, the ratio is in the range of 20 or 30 to 1, where it takes 20 (or 30) units of time for a process context switch and 1 unit of time for a thread context switch).

To get some experience with threads in both C and Java, we will solve the Lab 3 problem twice -- once in C (using *PThreads*), and again in Java (using Java threads).

For your reference, here are two sample threaded programs, one written in C and the other in Java (right-click to download):

ptsample.c

JSample.java

Experiment with these programs to get comfortable working with threads in both languages.

Your job is to create solutions in C and Java that work as follows:

Compiling and running in C:

> **gcc lab5.c -o lab5 -lm -lpthread**
> **lab5 1 2 3 4 5**
(((1.0 + 2.0) / 4) ^ 3.0 + 4.0 / 2) - (3 * 1.0 + (4 * 5.0 / 2.0)) = -10.578
> **lab5 -2.5 19.8 5.4 1.6 3.1**
(((-2.5 + 19.8) / 4) ^ 5.4 + 1.6 / 2) - (3 * -2.5 + (4 * 3.1 / 19.8)) = 2726.147
>

Compiling and running in Java can be done using an IDE such as eclipse or through the command line:

> **javac Lab5.java**
> **java Lab5 1 2 3 4 5**
(((1.0 + 2.0) / 4) ^ 3.0 + 4.0 / 2) - (3 * 1.0 + (4 * 5.0 / 2.0)) = -10.578
> **java Lab5 -2.5 19.8 5.4 1.6 3.1**
(((-2.5 + 19.8) / 4) ^ 5.4 + 1.6 / 2) - (3 * -2.5 + (4 * 3.1 / 19.8)) = 2726.147
>

Notes and Requirements:

- You should store your C solution in a file called lab5.c and your Java solution in a file called Lab5.java. You may wish to consult my solution for Lab 3 on Moodle. **Be sure to use the correct names for these files**.
- Your solution should follow the same pattern of thread creation as the diagram shown at the top of this page. For example, the main thread should start the first, second and third threads.
- To get the same kind of formatting in Java as we achieved in C you can use the printf() function of the System.out object.

- Use Math.pow(x,y) to calculate the x ^ y in Java. Math.pow() returns a double since it can be used with floating point numbers.
- Note that every arithmetic operation should perform the operation **on a separate line of code** like the sample code at the start of this lab. This will make it clear to see the flow of execution of operations. For example, an expression like this (a - 2) * 2 should be written like this:

```
t1 = a - 2;
t2 = t1 * 2;
```

or

```
t1 = a - 2;
t1 *=  2;
```

Submit:

Submit your programs prior to the deadline. Again, marks will be awarded according to how well you've taken advantage of the inherent parallelism of the problem. Source files (should only be lab5.c and lab5.java but if you created the java threads as separate files then include those as well) submitted on Moodle.