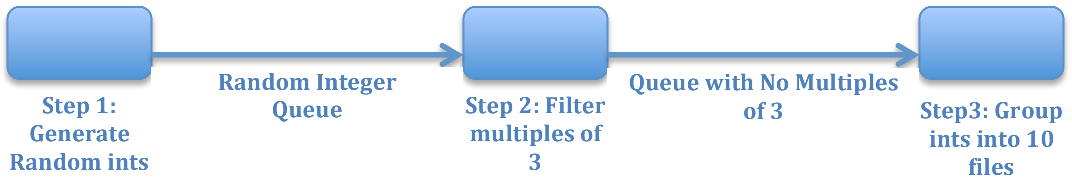
CS 3370 – Program 7
Pipelined Tasks

Write a program with a concurrent pipeline architecture that has three steps: 1) threads that produce random integers, 2) threads that filter out multiples of 3 from the integers produced in step 1, and 3) threads that groups the integers from step 2 into one of 10 files, as explained below. See the figure after the next paragraph. The first two steps are similar to what is in *wait5.cpp*. Place everything in a class named Pipeline, making sure that items related to thread communication are static members. Make the number of Step1 threads and Step 2 threads command-line arguments (argv[1] and argv[2]) and run your program with the values 4 and 3 for these steps, respectively.

For the third step, create exactly 10 threads that retrieve integers the second queue and group the numbers by their modulus base 10. Each of these 10 "grouper" threads will only remove a number from the front of the queue if its modulus corresponds to theirs. For example, grouper thread 0 will check to see if the first number in the queue ends in a 0 (i.e., it is congruent to 0 (mod 10)). If so, it will remove it and write it to its file (see below). Otherwise it does nothing. If the number at the front of the queue does not end with the proper digit, the thread leaves it in the queue for the appropriate thread to process. Each grouper thread does likewise for its respective residue base 10. The following diagram represents the architecture of this program.


Step 1: Generate Random ints
Random Integer Queue
Step 2: Filter multiples of 3
Queue with No Multiples of 3
Step3: Group ints into 10 files

When you are done, print out a report like the following (your numbers will vary):

```
$ ./a.out 4 3
Press Enter to quit...
Group 0 has 12383 numbers
Group 1 has 12264 numbers
Group 2 has 12339 numbers
Group 3 has 12329 numbers
Group 4 has 12136 numbers
Group 5 has 12146 numbers
Group 6 has 12213 numbers
Group 7 has 12325 numbers
Group 8 has 12266 numbers
Group 9 has 12358 numbers
```

Collect your 10 output text files and your source code, along with the execution output like what you see above, into a zip file for submission. (*Note*: on Visual Studio, you may have to press Enter twice and the program may run slowly. Be patient and let it finish. This program may run faster from the command line than in the IDE.)

FYI, my program is 106 lines of executable code, much of which is based on *wait5.cpp*.

Assessment Rubric

Competency ↓	**Emerging →**	**Proficient →**	**Exemplary**
Concurrency		Effective use of mutexes and condition variables to synchronize threads that share data. No mutex is locked for longer than needed. Distinguish between **notify_one** and **notify_all** correctly. Use of lambdas with **.wait**.	Use of **async** for launching the grouper threads and using **.get** on the corresponding **future**s to retrieve the number of numbers processed from each of the 10 groups.
Clean Code		No repeated code; No unnecessary code. No global data: use static class members, as in wait5.cpp.	
Other	Correct data in the 10 output files. Use range-based **for** to traverse collections of threads.	Proper initialization of static data.	