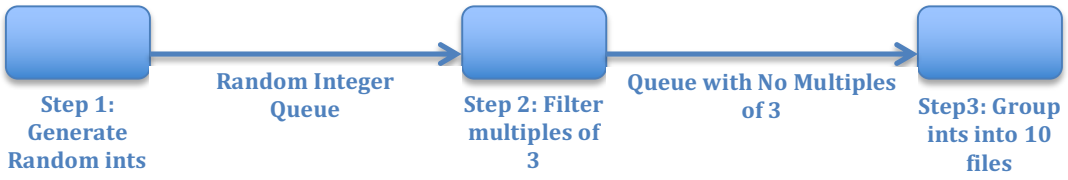
CS 3370 – Program 7

Pipelined Tasks

Write a program with a concurrent pipeline architecture that has three steps: 1) threads that produce 1000 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.

For the third step, create exactly 10 threads that retrieve integers from the second queue populated by the threads in Step 2, and group the numbers by their modulus (remainder) 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 from the queue 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 remainder mod 10. The following diagram represents the architecture of this program.



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

```
Group 7 has 271 numbers
Group 1 has 244 numbers
Group 9 has 264 numbers
Group 4 has 278 numbers
Group 0 has 275 numbers
Group 5 has 267 numbers
Group 8 has 285 numbers
Group 3 has 279 numbers
Group 2 has 268 numbers
Group 6 has 242 numbers
```

Note that the order of the group reports varies on the order that the grouper threads terminate.

Use 4 producer threads and 3 filter threads.

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: my solution to this program runs on all platforms, but it is HORRIBLY SLOW (wait 2 minutes, but it works) on Visual Studio 2015. It runs instantaneously on clang, g++, MinGW, and Visual Studio 13.

FYI, my program is 91 lines of actual code, much of which is copied from/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**.	
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.	