**CS 350 Spring 2015** **Program #1: Producer / Consumer**

**Assigned**: *Thursday March 5, 2015*

This is a producer-consumer problem. "Producers" will generate work for "consumers", store the work in a buffer, and then consumers will pick it up and perform the work. The producers in this case are Professors, and the consumers are Students. Each assignment created by a Professor thread will have an Assignment ID, a number of hours, and a field holding the ID of the Professor who created it. Each Professor thread will create a random number of work jobs a random number of times, and put them into a queue of limited size. Student (consumer) threads will read "work jobs" from the queue and simulate carrying them out by sleeping and printing output to the screen. The program should have a `main()` function that creates all of the Professor threads and Student threads.

Program usage:

```
        ./program2 [-a <num_assignings>]
                [-w <min_prof_wait>] [-W <max_prof_wait>]
                [-n <min_num_assignments>] [-N <max_num_assignments>]
                [-h <min_assignment_hours>] [-H <max_assignment_hours>]
                [-p <num_professors>] [-s <num_students>]
                [-d <students_per_assignment>] [-q <queue_size>]
```

## Professor Threads

Each Professor does the following, in a loop, `<num_assignings>` times:

- Waits for a random amount of time (select a random value in the range between `<min_prof_wait>` and `<max_prof_wait>` seconds, inclusive) while deciding what to assign.
- Chooses a random number of assignments to assign (select a random value in the range between `<min_num_assignments>` and `<max_num_assignments>`, inclusive)
- Adds each assignment to the queue, in a loop... the assignment takes a random number of hours to complete (select a random number in the range between `<min_assignment_hours>` and `<max_assignment_hours>`, inclusive).... one hour is simulated as one second in your program.

Each Professor thread should print a line with the following format once upon startup:

```
STARTING Professor 4
```

Each Professor thread should print output with the following format, each time it adds to the work queue:

```
ASSIGN Professor 4 adding Assignment 5: 2 Hours.
```

## Student Threads

Each student thread does the following, in a continuous loop:

- Waits until there is work in the queue that the student has not completed.
- When there is, reads an assignment from the queue.
- "Does" the work by sleeping for 1 second for each hour in the assignment, and printing a line of output for each hour worked, as shown below.

Each Student should print a line with the following format once upon startup:

```
STARTING Student 2
```

After picking up each assignment, Student threads should print output as follows:

```
BEGIN Student 2 working on Assignment 4 from Professor 2
WORK Student 2 working on Assignment 4 Hour 1 from Professor 2
WORK Student 2 working on Assignment 4 Hour 2 from Professor 2
END Student 2 working on Assignment 4 from Professor 2
```

Printed lines from one student may (and should!) be interspersed with printed lines from another student. But each line itself should not be interrupted by another. For example, the following is OK:

```
BEGIN Student 3 working on Assignment 4 from Professor 2
BEGIN Student 1 working on Assignment 2 from Professor 3
WORK Student 2 working on Assignment 4 Hour 1 from Professor 2
WORK Student 1 working on Assignment 4 Hour 1 from Professor 3
WORK Student 2 working on Assignment 4 Hour 2 from Professor 2
END Student 2 working on Assignment 4 from Professor 2
BEGIN Student 1 working on Assignment 2 from Professor 3
etc.
```

But this is not OK:

```
BEGIN Student 3 working oBEGIN Student 1 working on Assignment 2 from Professor 3
n Assignment 4 from Professor 2
```

Note that to simulate 4 hours of work, your program should sleep for one second four times; your program should not call sleep(4). We want to see interleaving of completed work, because carrying out the work is not in a critical section, reading what work to do from the shared buffer is the critical section.

Make sure none of the following (or any other incorrect behavior) happens in your program:

- A student removes an item while a professor is in the process of putting it in the queue.
- A student removes items that are not there at all.
- A student removes items that have already been removed.
- A professor puts something in the queue when there is no free slot.
- A professor overwrites an item that has not been removed.
- Work that is added to the queue does not get completed.
- Work that is added to the queue gets completed too many times (by too many students).
- A student does the same assignment two or more times.

Every assignment that is added to the queue should get done exactly `<students_per_assignment>` times. Note that this makes the program different from a typical producer/consumer problem, in which each item of work gets taken out of the queue exactly once. You must place the item in the queue exactly once, and it should be read and completed by exactly `<students_per_assignment>` Students, and then removed from the queue.

Your program must have a *clean exit strategy*! In other words, all Student threads should terminate, but only after all professor threads have terminated. Professor threads should print a line with the following format, just before exiting:

```
EXITING Professor 5
```

Student threads should print a line with the following format, just before exiting:

```
EXITING Student 2
```

(Obviously, "5" and "2" should be replaced by the appropriate Professor and Student ids).

Your code may use any strategy you want for exiting cleanly... the only constraints are that (a) Professors and Students should announce their own exit (i.e. it should not be announced from within the main thread), and (b) no Student should exit before all professors have exited (after all, professors may decide to assign more work!).

The program should be implemented in C for Unix, using pthreads and the associated synchronization functions that come with them, including condition variables.

We will specify positive integers for all values specified on the command line. Use the following minimums, maximums, and defaults:

| | | | |
|---|---|---|---|
| `<num_assignings>` | min: 1 | max: 100000 | default: 10 |
| `<min_prof_wait>` | min: 1 | max: 10 | default: 1 |
| `<max_prof_wait>` | min: 1 | max: 100 | default: 5 |
| `<min_num_assignments>` | min: 1 | max: 10 | default: 1 |
| `<max_num_assignments>` | min: 1 | max: 100 | default: 10 |
| `<min_assignment_hours>` | min: 1 | max: 5 | default: 1 |
| `<max_assignment_hours>` | min: 1 | max: 10 | default: 5 |
| `<num_professors>` | min: 1 | max: 10 | default: 2 |
| `<num_students>` | min: 1 | max: 10 | default: 2 |
| `<students_per_assignment>` | min: 1 | max: 10 | default: `<num_students>` |
| `<queue_size>` | min: 1 | max: 256 | default: 8 |

Please check that pairs of variables that specify a range have the property that the max exceeds the min, and enforce other logical constraints.

**Submission**

Submit your program as a single tar ball file, according to the specifications of Lab 1, but with “Lab” replaced with “Program”.