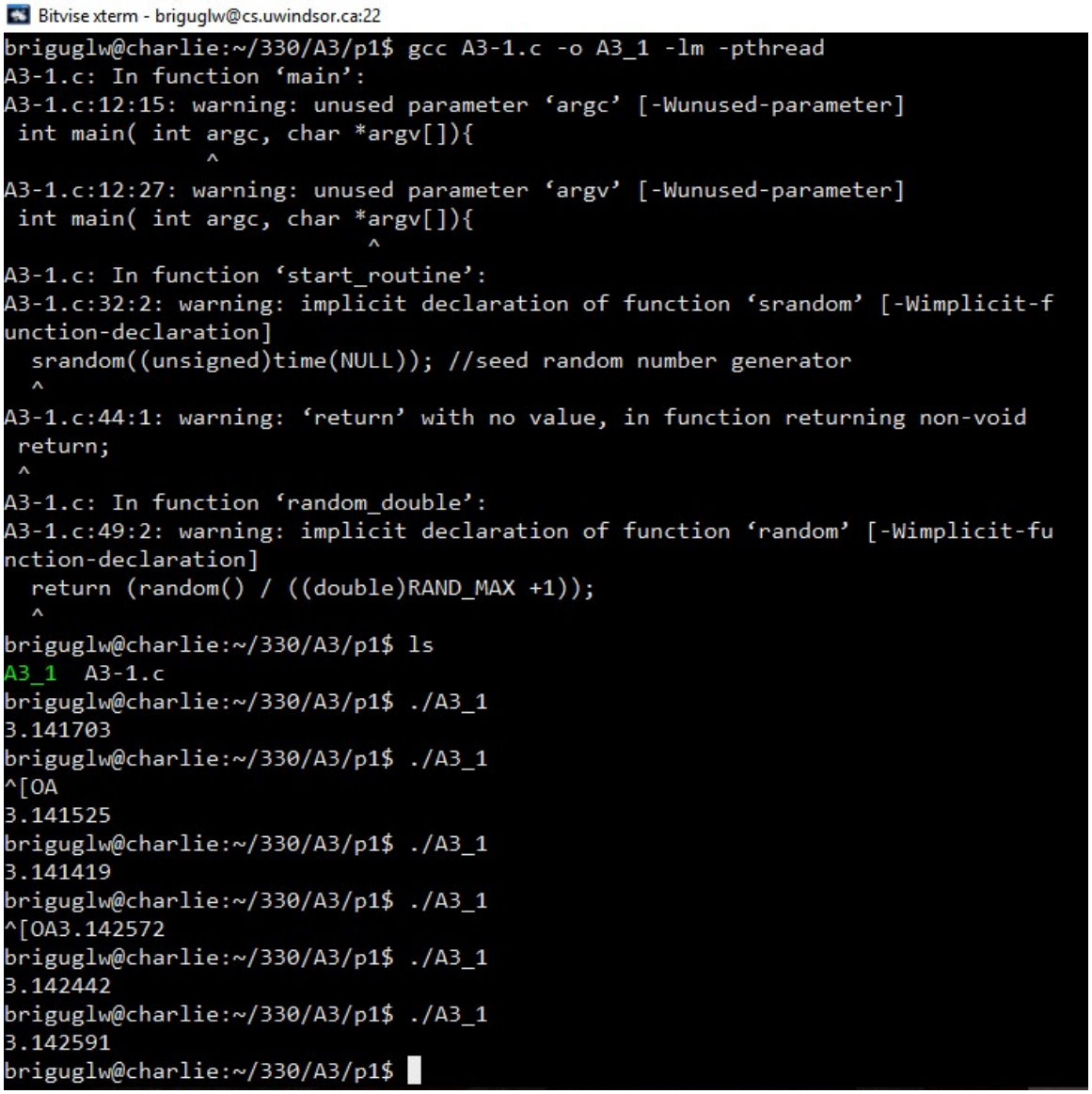
2.
-the comments on the results of Program-1 to Program-4,

In your report, you must provide details about the implementation, show how you run the two programs and the comments that address relationship between the running time and the concepts learned in class. Marks will be deducted if explanations/comments are missing.

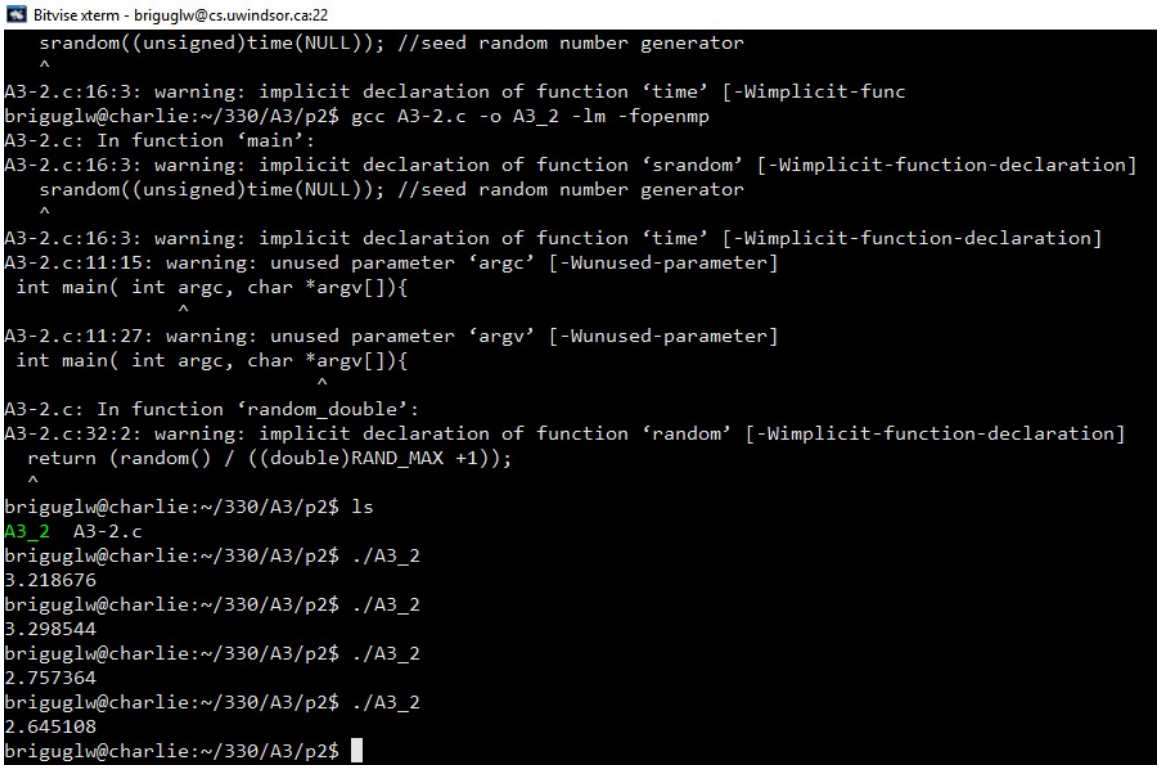
I confirm that I will keep the content of this assignment confidential. I confirm that I have not received any unauthorized assistance in preparing for or writing this assignment. I acknowledge that a mark of 0 may be assigned for copied work. William Briguglio

Program 1)



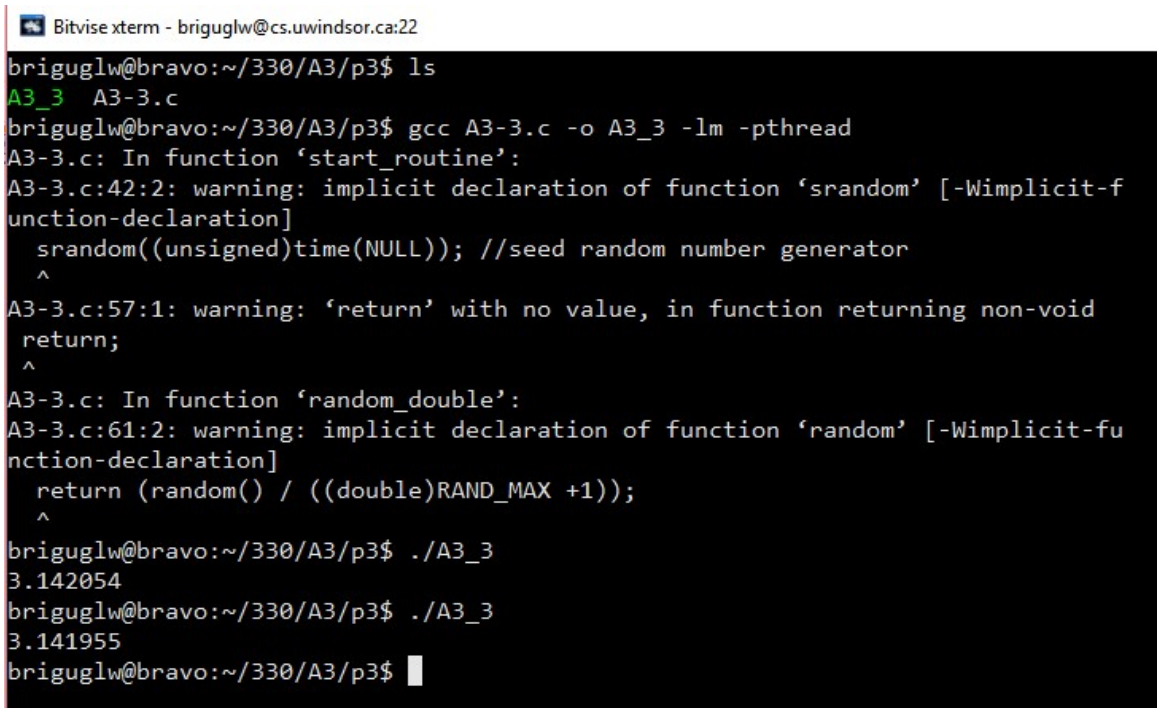
This program's estimation of pi is always with 2 decimal points of accuracy, beyond 3.14 it was unreliable. I implemented this by using a single pthread, which generated random points. Then determined the hit count.

Program 2)



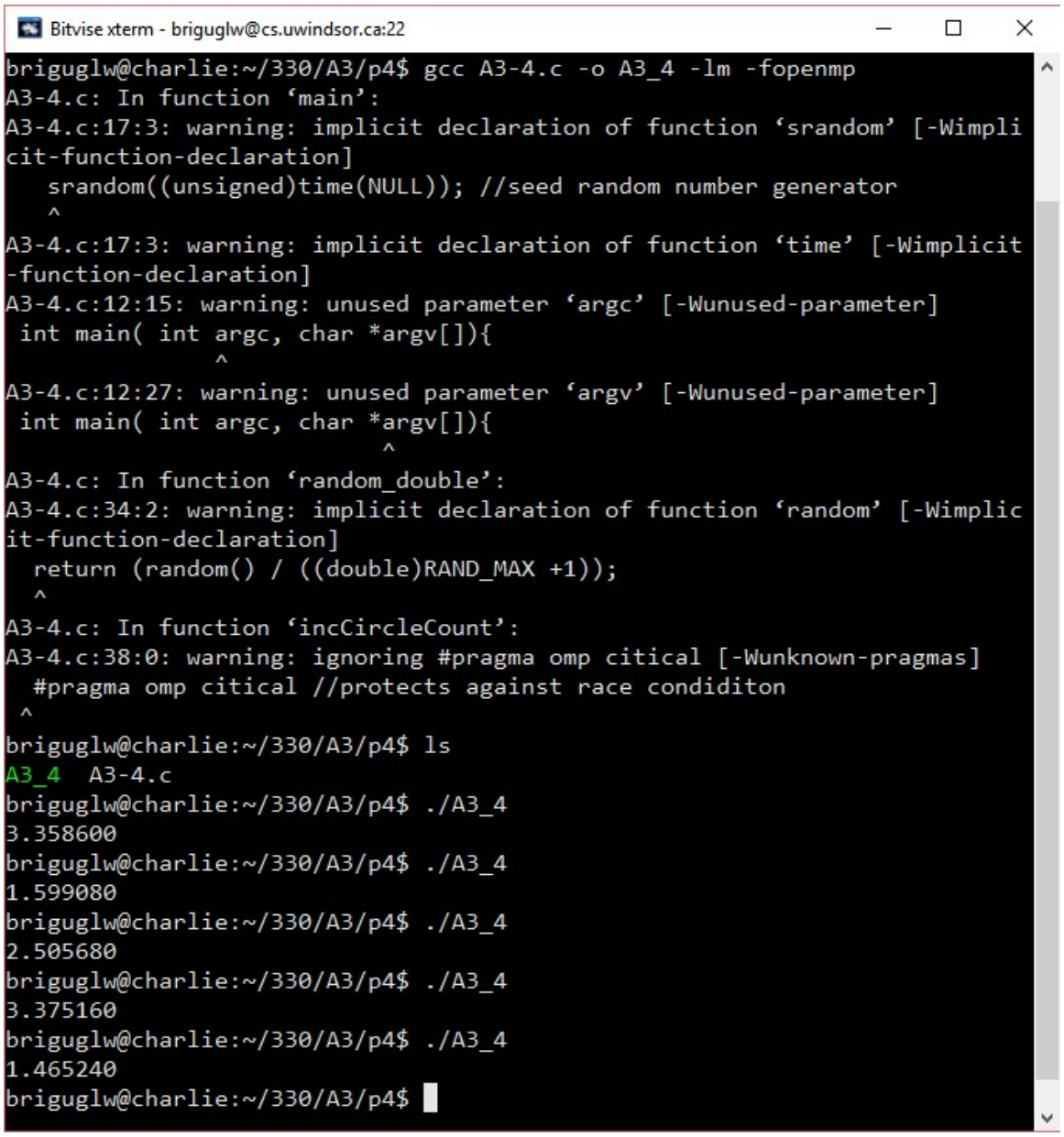
This program's estimation of pi was much less accurate and slower, although making the number of estimations larger does make the program more accurate. I implement this program by using OpenMP directives to define ***parallel zones*** which are areas of code that can run in parallel. Specifically I used the *parallel for* directive to create a for loop for which the work is divided among multiple threads (data parallelism).

Program 3)



This program's estimation of pi is always with 2 decimal points of accuracy, beyond 3.14 it was unreliable. I implemented this by using a 4 pthreads (however this number can be easily changed by changing a single macro in the code), each of which generated random points. Then determined the hit count. They updated a global variable called circle_count and then the threads all exited leaving the parent thread to calculate pi using circle_count. I assured that a race condition didn't arise from the multiple threads modifying the same variable by using mutex locks. Specifically I used the pthread_mutex_lock(&mutex) function to acquire the lock and the pthread_mutex_unlock(&mutex) function to release the lock

Program 4)



This program's estimation of pi was much about as accurate as program-2's estimation and was the same speed. Of course, since no race conditions can arise so in this program, I can only assume it to be more accurate. The only difference from program 2 to this program is I protected against race conditions by using the *#pragma omp critical* directive. This made it so that no two threads can alter the circle_count variable at once.

Additional notes: program 2 and program 4 were less accurate but only because I used less points. I assumed they'd be just as accurate if they used the same number of point as program 1 and 3 but I found that the OpenMP method was incredibly slow. I was working on my computer from home and using bitvise SSH client to access the schools ubuntu terminals. I believe this may have something to do with the slow speeds when using OpenMP because upon examining my code I can only determine that it should run at least at the same if not faster.