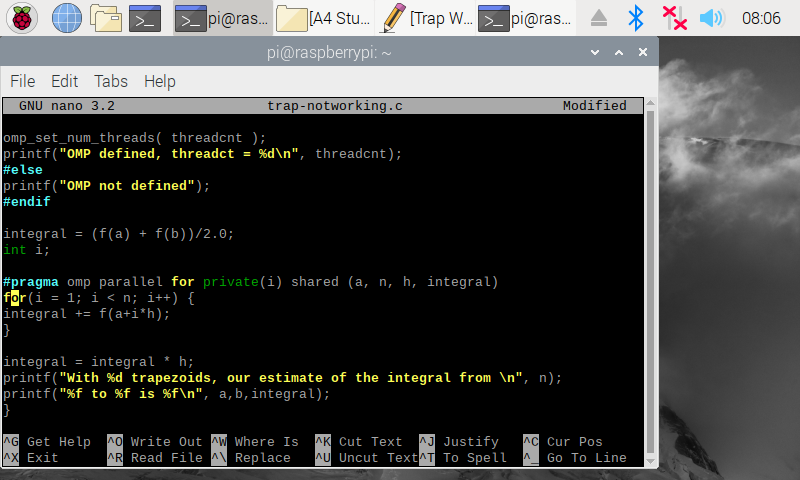
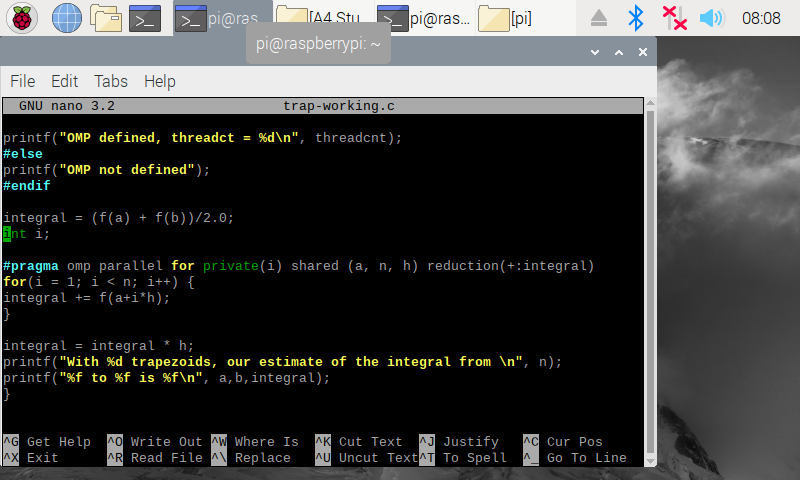
Nathan Heckman

A4 Parallel Programming Task

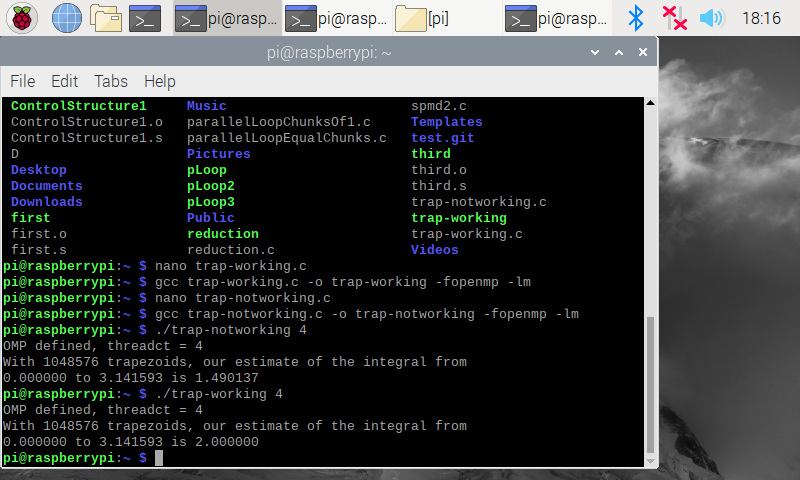
**Question 2.1 - trap-notworking.c Code**

This is the code for the incorrect trap-notworking.c program. The only difference between this code and the working code is the line highlighted in red above. This code doesn’t allow the program to perform the reduction clause and therefore gives an incorrect result.

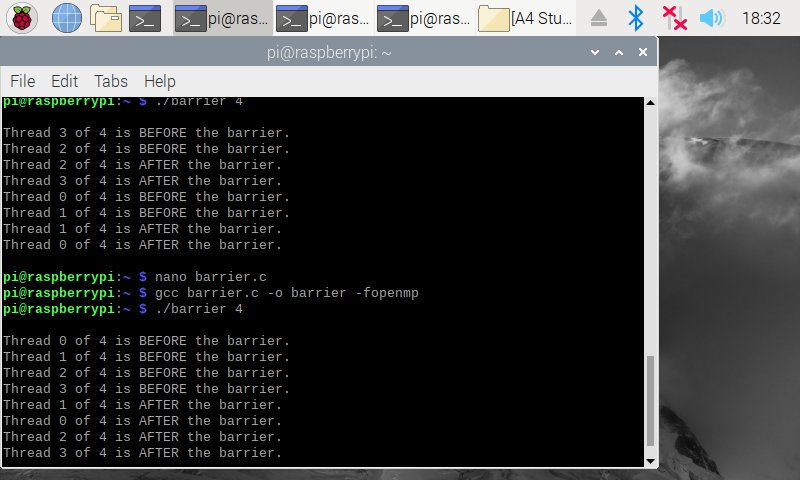
**Question 2.2 trap-working Code**

This is the code for the incorrect trap-working.c program. The reduction clause has been added and the code will now work as intended. The integral accumulator requires the reduction clause to give the correct output.

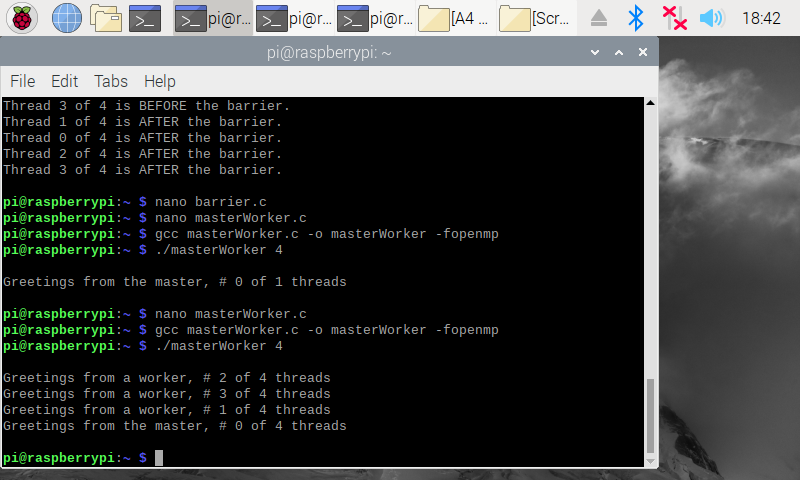
**Question 2.2 Trap Output Comparison**



The outputs for each trap program using 4 threads. We can see that the trap-working.c program produces the desired output of 2 and uses the reduction clause while the trap-notworking.c program the incorrect output and doesn’t use the reduction clause. The incorrect output is given when the reduction clause isn’t used since the program equally divides the amount of trapezoids by 4, which causes problems with synchronization.

**Question 3.1 barrier.c Outputs Comparison**

The outputs for the two barrier.c programs, the first has #pragma omp barrier commented out and the second doesn’t. The second execution with #pragma omp barrier included properly splits the execution in groups of 4, with each thread included before and after the barrier. The first execution doesn’t properly split the threads before and after the barrier.

**Question 4.1 masterWorker.c Outputs Comparison**

The outputs for the two masterWorker.c programs, the first has #pragma omp parallel commented out and the second doesn’t. As shown by the outputs, the first program is not run as parallel and therefore only on one core, apparent by the #0 of 1 threads. The updated program that includes #pragma omp parallel properly utilizes all threads and assigns thread 0 as the master thread.