Chris Keefe
Assignment 1

1. **What does process rank 5's counter store at the end of the computation?**
   Rank 5's counter stores 20, because it has received the integer 4 five times from rank 4.

Note: I run all jobs (local and remote) from a shared hostfile in the parent directory.

```
chris@tumbleBox:~/src/hpc599/a1/a1 (module1)> mpicc pingpong_act1_crk239.c -lm -o pingpong
chris@tumbleBox:~/src/hpc599/a1/a1 (module1)> mpirun -np 6 -hostfile ../myhostfile.txt ./pingpo
ng
Rank 0 received the value 5
Rank 1 received the value 0
Rank 2 received the value 15
Rank 3 received the value 10
Rank 4 received the value 25
Rank 5 received the value 20
```

2. **How many process ranks are used in the script above?**
   Ten, numbered 0-9.

```
Rank 9 received the value 40
Rank 8 received the value 45
Rank 7 received the value 30
Rank 2 received the value 15
Rank 5 received the value 20
Rank 6 received the value 35
Rank 4 received the value 25
Rank 3 received the value 10
Number of processes: 10
Rank 0 received the value 5
Rank 1 received the value 0
```

3. **What does process rank 5's counter store at the end of the computation?**
   40. It receives 10 4's from rank 4.

```
Rank 5 received the value 40
Rank 4 received the value 30
Rank 2 received the value 10
Rank 1 received the value 0
Rank 3 received the value 20
Rank 0 received the value 50
```

4. **Comparing Programming Activities #2 and #3, which was easier to implement? Explain.**

Activity two was marginally easier to implement, but only because I'm not really clear on how MPI_Wait works, and what the purpose of &request and &status are. As such, I spent some time thinking about whether I could skip the MPI_Wait call. I got correct results over ~10 trials with and without MPI_Wait, and suspect that unhelpful overhead here, because MPI_Recv is a blocking function, and follows MPI_Send in all cases in my implementation.

I suspect activity three would have been the easier one if I were more fluent with the library, because the order of the send/receive calls can be consistent across all ranks, making for simpler and more readable code.

5. **Comparing Programming Activities #4 and #5, which was easier to implement? Explain.**

Activity 5 (with MPI_ANY_SOURCE) was easier to implement. Not having to manage source and destination ranks simplified the communications and the logic in the loop (allowing me to remove a branch). It didn't hurt that I spent a couple of hours thinking through activity 4 before starting in on 5, either.

```
My rank: 37, old counter: 53
My rank: 37, new counter: 90
My rank: 37, next to recv: 27
My rank: 23, old counter: 117
My rank: 23, new counter: 140
My rank: 23, next to recv: 26
My rank: 5, old counter: 166
My rank: 5, new counter: 171
My rank: 5, next to recv: 4
My rank: 36, old counter: 186
My rank: 36, new counter: 222
My rank: 36, next to recv: 34
Master: first rank: 14
My rank: 4, old counter: 171
My rank: 4, new counter: 175
My rank: 4, next to recv: 11
My rank: 39, old counter: 14
My rank: 39, new counter: 53
My rank: 39, next to recv: 37
My rank: 26, old counter: 140
My rank: 26, new counter: 166
My rank: 26, next to recv: 5
My rank: 27, old counter: 90
My rank: 27, new counter: 117
My rank: 27, next to recv: 23
My rank: 14, old counter: 0
My rank: 14, new counter: 14
My rank: 14, next to recv: 39
My rank: 11, old counter: 175
My rank: 11, new counter: 186
My rank: 11, next to recv: 36
~
```