Exercise 1
One C Solution

```
#include <stdio.h>
#include "mpi.h"

main(int argc, char** argv){

  int my_PE_num, number_to_send, message_received;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_PE_num);

  number_to_send = my_PE_num;

  if (my_PE_num==7)
      MPI_Send( &number_to_send, 1, MPI_INT, 0, 10, MPI_COMM_WORLD);
  else
      MPI_Send( &number_to_send, 1, MPI_INT, my_PE_num+1, 10, MPI_COMM_WORLD);

  MPI_Recv( &message_received, 1, MPI_INT, MPI_ANY_SOURCE, 10, MPI_COMM_WORLD, &status);

  printf("PE %d received %d.\n", my_PE_num, message_received);

  MPI_Finalize();

}
```

Exercise 1
A Possible Fortran Solution

```
program shifter
implicit none

include 'mpif.h'

integer my_pe_num, errcode, numbertosend, message_received
integer status(MPI_STATUS_SIZE)

call MPI_INIT(errcode)

call MPI_COMM_RANK(MPI_COMM_WORLD, my_pe_num, errcode)

numbertosend = my_pe_num

if (my_pe_num.EQ.7) then
   call MPI_Send(numbertosend, 1, MPI_INTEGER, 0, 10, MPI_COMM_WORLD, errcode)
else
   call MPI_Send(numbertosend, 1, MPI_INTEGER, my_pe_num+1, 10, MPI_COMM_WORLD, errcode)
endif

call  MPI_Recv(message_received, 1, MPI_INTEGER, MPI_ANY_SOURCE, 10, MPI_COMM_WORLD, status, errcode)

print *,'PE', my_pe_num, ' received ', message_received, '.'

call MPI_FINALIZE(errcode)
end
```

Exercise 1
Output

```
c557-603$ mpicc solution1.c
c557-603$ mpirun -n 8 a.out
PE 2 received 1.
PE 0 received 7.
PE 4 received 3.
PE 3 received 2.
PE 5 received 4.
PE 1 received 0.
PE 7 received 6.
PE 6 received 5.
```

Exercise 1
Technically not perfect.

```
#include <stdio.h>
#include "mpi.h"

main(int argc, char** argv){

  int my_PE_num, number_to_send, message_received;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_PE_num);

  number_to_send = my_PE_num;

  if (my_PE_num==7)
      MPI_Ssend( &number_to_send, 1, MPI_INT, 0, 10, MPI_COMM_WORLD);
  else
      MPI_Ssend( &number_to_send, 1, MPI_INT, my_PE_num+1, 10, MPI_COMM_WORLD);

  MPI_Recv( &message_received, 1, MPI_INT, MPI_ANY_SOURCE, 10, MPI_COMM_WORLD, &status);

  printf("PE %d received %d.\n", my_PE_num, message_received);

  MPI_Finalize();

}
```

Deadlock!

Exercise 1
For the pedants...

```
#include <stdio.h>
#include "mpi.h"

main(int argc, char** argv){

  int my_PE_num, number_to_send, message_received;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_PE_num);

  number_to_send = my_PE_num;

  if (my_PE_num==7){
    MPI_Recv( &message_received, 1, MPI_INT, MPI_ANY_SOURCE, 10, MPI_COMM_WORLD, &status);
    MPI_Ssend( &number_to_send, 1, MPI_INT, 0, 10, MPI_COMM_WORLD);
  }
  else{
    MPI_Ssend( &number_to_send, 1, MPI_INT, my_PE_num+1, 10, MPI_COMM_WORLD);
    MPI_Recv( &message_received, 1, MPI_INT, MPI_ANY_SOURCE, 10, MPI_COMM_WORLD, &status);
  }

  printf("PE %d received %d.\n", my_PE_num, message_received);

  MPI_Finalize();

}
```

Breaks the Deadlock!

Exercise 2
Impossible Solution

- There is no possible solution.
- You can not accomplish this task with the commands you were given.
- It is simply impossible to be sure there isn't a node somewhere "out there" that hasn't yet responded.
- It is possible to create many "solutions" that will work *almost* all of the time. Particularly on a tightly coupled machine like Stampede.
- What if Bridges was nodes spread around the solar system. Would your answer still work?
- *It is generally not hard to write MPI codes that will always work. I gave you a really tricky problem to keep you humble, and not even all of our most basic set of commands to use.*

Exercise 2
Almost Solution

```
#include <stdio.h>
#include "mpi.h"

main(int argc, char** argv){

  int my_PE_num, numberofnodes, data;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_PE_num);

  if (my_PE_num==0)
    for (numberofnodes=1;numberofnodes<512;numberofnodes++)
      if(MPI_Send( &data, 1, MPI_INT, numberofnodes, 10, MPI_COMM_WORLD) != MPI_SUCCESS)
         break;

  printf("The number of nodes is %d.", numberofnodes);

  MPI_Finalize();

}
```

Actually MPI has a very comprehensive error handling capability. You can redefine it to abort (the default here), return an error condition, or even call your own handler to do anything you want.

This would work here:

```
MPI_Comm_set_errhandler(MPI_COMM_WORLD, MPI_ERRORS_RETURN)
```