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

const int MAX_STRING = 100;

int main(int argc, char **argv) {
    char    greeting[MAX_STRING];
    int     comm_sz;    /* number of processes  */
    int     my_rank;    /* process rank         */

    //Initialize the command line arguments on every process
    MPI_Init(&argc, &argv);

    //Get the number of processes in MPI_COMM_WORLD, and put
    //it in the 'comm_sz" variable; ie., how many processes
    //are running this program (the same as what you put in the
    //-np argument).
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

    //Get the rank of this particular process in MPI_COMM_WORLD,
    //and put it in the 'my_rank' variable -- ie., what number
    //is this process
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    if (my_rank != 0) {
        //Rank 0 is typically the head process
        //All other processes will execute this code

        //sprintf writes into a string
        sprintf(greeting, "Greetings from process %d of %d!", my_rank, comm_sz);

        //Send the data in 'greeting' to process 0
        //Note that every other process will send this
        //message to process 0
        if (my_rank != 3) {
        MPI_Send(greeting /*data*/,
                 strlen(greeting)+1 /*data size*/,
                 MPI_CHAR /*data type*/,
                 0 /*destination process -- this sends to process 0*/,
                 0 /*tag -- you can ignore this*/,
                 MPI_COMM_WORLD /* communicator -- all our communication
                                   is in MPI_COMM_WORLD */);
        }
    } else {
        //The head process (process 0) goes into this part of
        //the if statement.

        //All the processes not 0, so processes 1 to comm_sz - 1
        //will have sent a message to process 0.
        //Each MPI_Send message needs to get paired with a
        //MPI_Recv message
        for (int q = 1; q < comm_sz; q++) {
//        for (int q = comm_sz - 1; q >= 1; q--) {
            MPI_Recv(greeting /*data target -- the data send will be
                               received into here*/,
                     MAX_STRING /*maximum number of elements to
                                  receive*/,
                     MPI_CHAR /*data type*/,
                     q /*process we're receiving the data from -- there
                         must have been a MPI_Send
                         call from that process, or else this will
                         block waiting for it */,
                     0 /*tag -- can ignore this */,
```

```
                    MPI_COMM_WORLD /*communicator*/,
                    MPI_STATUS_IGNORE /*can ignore this too for the
                                       time being*/);

            printf("%s\n", greeting);
        }
    }

    /*Inform MPI that this process has finished*/
    MPI_Finalize();
    return 0;
}
```