```cpp
#include "greetings.h"
#include <cassert>
#include <cstring>
#include <iostream>
#include <mpi.h>                                        // MPI
#include <string>
using namespace std;

// see  http://www.open-mpi.org/doc/current
// for details on MPI functions

void greetings(MPI_Comm const &icomm)
{
    int myrank, numprocs;
    MPI_Comm_rank(icomm, &myrank);                      // my MPI-rank
    MPI_Comm_size(icomm, &numprocs);                    // #MPI processes
    char *name  = new char [MPI_MAX_PROCESSOR_NAME],
         *chbuf = new char [MPI_MAX_PROCESSOR_NAME];

    int reslen, ierr;
    MPI_Get_processor_name( name, &reslen);

    if (0==myrank) {
        cout << "   " << myrank << " runs on " << name << endl;
        for (int i = 1; i < numprocs; ++i) {
            MPI_Status stat;
            stat.MPI_ERROR = 0;             // M U S T   be initialized!!


            //ierr = MPI_Recv(chbuf, MPI_MAX_PROCESSOR_NAME, MPI_CHAR, MPI_ANY_SOURCE, MPI_ANY_TAG, icomm, &stat);
            ierr = MPI_Recv(chbuf, MPI_MAX_PROCESSOR_NAME, MPI_CHAR, i, i, icomm, &stat);
            assert(0==ierr);

            cout << "   " << stat.MPI_SOURCE << " runs on " << chbuf;
            int count;
            MPI_Get_count(&stat, MPI_CHAR, &count); // size of received data
            cout << " (length: " << count << " )" << endl;
            //  stat.Get_error()        // Error code
        }
    }
    else {
        int dest = 0;
        ierr = MPI_Send(name, strlen(name) + 1, MPI_CHAR, dest, myrank, icomm);
        assert(0==ierr);
    }
    delete [] chbuf;
    delete [] name;
    return;
}


void greetings_cpp(MPI_Comm const &icomm)
{
    int myrank, numprocs;
    MPI_Comm_rank(icomm, &myrank);                      // my MPI-rank
    MPI_Comm_size(icomm, &numprocs);                    // #MPI processes
    string name(MPI_MAX_PROCESSOR_NAME,'#'),            // C++
           recvbuf(MPI_MAX_PROCESSOR_NAME,'#');         // C++: receive buffer, don't change size

    int reslen, ierr;
    MPI_Get_processor_name(name.data(), &reslen);
    name.resize(reslen);                                // C++


    if (0==myrank) {
```

```
   66:         cout << "    " << myrank << " runs on " << name << endl;
   67:         for (int i = 1; i < numprocs; ++i) {
   68:             MPI_Status stat;
   69:             stat.MPI_ERROR = 0;            // M U S T   be initialized!!
   70:
   71:             //ierr = MPI_Recv(recvbuf.data(), MPI_MAX_PROCESSOR_NAME, MPI_CHAR, MPI_
ANY_SOURCE, MPI_ANY_TAG, icomm, &stat);
   72:             ierr = MPI_Recv(recvbuf.data(), MPI_MAX_PROCESSOR_NAME, MPI_CHAR, i, i,
icomm, &stat);
   73:             assert(0==ierr);
   74:
   75:             int count;
   76:             MPI_Get_count(&stat, MPI_CHAR, &count); // size of received data
   77:             string const chbuf(recvbuf,0,count);    // C++
   78:             cout << "    " << stat.MPI_SOURCE << " runs on " << chbuf;
   79:             cout << " (length: " << count << " )" << endl;
   80:             //  stat.Get_error()        // Error code
   81:         }
   82:     }
   83:     else {
   84:         int dest = 0;
   85:         ierr = MPI_Send(name.data(), name.size(), MPI_CHAR, dest, myrank, icomm);
   86:         assert(0==ierr);
   87:     }
   88:     return;
   89: }
```

```
1: //      general header for all functions in directory
2:
3: #ifndef GREETINGS_FILE
4: #define GREETINGS_FILE
5:
6: #include <mpi.h>
7:
8: void greetings(MPI_Comm const &icomm);
9: void greetings_cpp(MPI_Comm const &icomm);
10:
11: #endif
```

```cpp
#include <iostream>
#include <mpi.h>

#include "greetings.h"

using namespace std;


int main(int argc , char **argv )
{
    // -------------------- E2 --------------------
    MPI_Init(&argc, &argv);     // Initializes the MPI execution environment

    // -------------------- E1 --------------------
    MPI_Comm const icomm(MPI_COMM_WORLD);   // MPI_COMM_WORLD ... all processes

    // -------------------- E3 --------------------
    int rank;
    MPI_Comm_rank(icomm, &rank);   // Determines the rank of the calling process in the communicator.

    if (rank == 0)
    {
        int size;
        MPI_Comm_size(icomm, &size); // Returns the size of the group associated with a communicator.
        cout << "Process " << rank << " says: " << size << " proesses are running." << endl;
    }
    // To vary number of processes: changed number in GCC_default.mk file
    // alternatively, call in terminal:
    // /usr/bin/mpirun --oversubscribe -display-map -mca btl ^openib  -np 4 ./main.GCC_
    // or
    // /usr/bin/mpirun --oversubscribe -display-map -mca btl ^openib  -np 8 ./main.GCC_


    // -------------------- E4 --------------------
    greetings_cpp(MPI_COMM_WORLD);  // greetings with sorted output


    MPI_Finalize();     // Terminates MPI execution environment
    return 0;
}
```