```
// Demonstrate simple MPI program
// George F. Riley, Georgia Tech, Fall 2011


#include <iostream>
#include <stdio.h>
#include <stdlib.h>

#include "mpi.h"

using namespace std;

#define MSG_SIZE 1000
char buf[MSG_SIZE];  // Message contents

int main(int argc,char**argv)
{
  int  numtasks, rank, rc;

  rc = MPI_Init(&argc,&argv);
  if (rc != MPI_SUCCESS) {
    printf ("Error starting MPI program. Terminating.\n");
    MPI_Abort(MPI_COMM_WORLD, rc);
  }

  MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  printf ("Number of tasks= %d My rank= %d\n", numtasks,rank);
  for (int round = 0; round < 100;++round)
    {
      if (rank == 0)
        { // Rank zero sends first then receives, all other
          // receive first then send
          cout << "Rank " << rank
               << " sending to rank" << rank + 1
               << " round " << round << endl;
          rc = MPI_Send(buf, sizeof(buf), MPI_CHAR, rank + 1,
                        0, MPI_COMM_WORLD);
          if (rc != MPI_SUCCESS)
            {
              cout << "Rank " << rank
                   << " send failed, rc " << rc << endl;
              MPI_Finalize();
              exit(1);
            }
          MPI_Status status;
          rc = MPI_Recv(buf, sizeof(buf), MPI_CHAR, MPI_ANY_SOURCE,
                        0, MPI_COMM_WORLD, &status);
          if (rc != MPI_SUCCESS)
            {
              cout << "Rank " << rank
                   << " recv failed, rc " << rc << endl;
              MPI_Finalize();
              exit(1);
            }
          int count = 0;
```

Program testMPI.cc

```
          MPI_Get_count(&status, MPI_CHAR, &count);
          cout << "Rank " << rank
               << " received " << count << " bytes from "
               << status.MPI_SOURCE << endl;
        }
      else
        {
          MPI_Status status;
          rc = MPI_Recv(buf, sizeof(buf), MPI_CHAR, MPI_ANY_SOURCE,
                        0, MPI_COMM_WORLD, &status);
          if (rc != MPI_SUCCESS)
            {
              cout << "Rank " << rank
                   << " recv failed, rc " << rc << endl;
              MPI_Finalize();
              exit(1);
            }
          // Now send to next rank (0 if we are last rank)
          int nextRank = rank + 1;
          if (nextRank == numtasks) nextRank = 0;
          cout << "Rank " << rank
               << " sending to rank" << nextRank
               << " round " << round << endl;
          rc = MPI_Send(buf, sizeof(buf), MPI_CHAR, nextRank,
                        0, MPI_COMM_WORLD);
          if (rc != MPI_SUCCESS)
            {
              cout << "Rank " << rank
                   << " send failed, rc " << rc << endl;
              MPI_Finalize();
              exit(1);
            }
        }
    }
  cout << "Rank " << rank << " exiting normally" << endl;
  MPI_Finalize();
}
```

Program testMPI.cc (continued)

```
// Demonstrate simple MPI program
// This one uses non-blocking ISend/Irecv
// George F. Riley, Georgia Tech, Fall 2011


#include <iostream>
#include <stdio.h>
#include <stdlib.h>

#include "mpi.h"

using namespace std;

#define MSG_SIZE 1000
char buf[MSG_SIZE];  // Message contents

int main(int argc,char**argv)
{
  int  numtasks, rank, rc;

  rc = MPI_Init(&argc,&argv);
  if (rc != MPI_SUCCESS) {
    printf ("Error starting MPI program. Terminating.\n");
    MPI_Abort(MPI_COMM_WORLD, rc);
  }

  MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  printf ("Number of tasks= %d My rank= %d\n", numtasks,rank);
  for (int round = 0; round < 100;++round)
    {
      if (rank == 0)
        { // Rank zero sends first then receives, all other
          // receive first then send
          cout << "Rank " << rank
               << " sending to rank" << rank + 1
               << " round " << round << endl;
          MPI_Request request;
          rc = MPI_Isend(buf, sizeof(buf), MPI_CHAR, rank + 1,
                         0, MPI_COMM_WORLD, &request);
          // Presumably more processing here....
          // Eventually must call either MPI_Wait (wait for transfer
          // complete, or MPI_Test (test if transfer complete);
          // The sender must insure the data buffer (buf in this example)
          // is unchanged until the transfer has completed.
          if (rc != MPI_SUCCESS)
            {
              cout << "Rank " << rank
                   << " send failed, rc " << rc << endl;
              MPI_Finalize();
              exit(1);
            }
          MPI_Status status;
          MPI_Wait(&request, &status);
          // At this point the send is complete and the buffer can be
          // reused.
```

Program testMPI2.cc

```
          // Now queue the receive request, also non-blocking
          rc = MPI_Irecv(buf, sizeof(buf), MPI_CHAR, MPI_ANY_SOURCE,
                         0, MPI_COMM_WORLD, &request);
          if (rc != MPI_SUCCESS)
            {
              cout << "Rank " << rank
                   << " recv failed, rc " << rc << endl;
              MPI_Finalize();
              exit(1);
            }
          int count = 0;
          // Presumably more work here; the receive has not completed
          // yet and the data is not yet available.
          // Now wait for the Irecv to complete.  You can use either
          // MPI_Wait or MPI_Test or MPI_Test_Any
          MPI_Wait(&request, &status);
          // The receive is now completed and data available.
          MPI_Get_count(&status, MPI_CHAR, &count);
          cout << "Rank " << rank
               << " received " << count << " bytes from "
               << status.MPI_SOURCE << endl;
        }
      else
        {
          MPI_Status status;
          rc = MPI_Recv(buf, sizeof(buf), MPI_CHAR, MPI_ANY_SOURCE,
                        0, MPI_COMM_WORLD, &status);
          if (rc != MPI_SUCCESS)
            {
              cout << "Rank " << rank
                   << " recv failed, rc " << rc << endl;
              MPI_Finalize();
              exit(1);
            }
          // Now send to next rank (0 if we are last rank)
          int nextRank = rank + 1;
          if (nextRank == numtasks) nextRank = 0;
          cout << "Rank " << rank
               << " sending to rank" << rank + 1
               << " round " << round << endl;
          rc = MPI_Send(buf, sizeof(buf), MPI_CHAR, nextRank,
                        0, MPI_COMM_WORLD);
          if (rc != MPI_SUCCESS)
            {
              cout << "Rank " << rank
                   << " send failed, rc " << rc << endl;
              MPI_Finalize();
              exit(1);
            }
        }
    }
  cout << "Rank " << rank << " exiting normally" << endl;
  MPI_Finalize();
}
```

Program testMPI2.cc (continued)

```
// Demonstrate simple MPI barriers and collectives
// This one uses non-blocking ISend/Irecv
// George F. Riley, Georgia Tech, Fall 2011


#include <iostream>
#include <stdio.h>
#include <stdlib.h>

#include "mpi.h"

using namespace std;

int main(int argc,char**argv)
{
  int  numtasks, rank, rc;

  // As always, we must call MPI_Init
  rc = MPI_Init(&argc,&argv);
  if (rc != MPI_SUCCESS) {
    printf ("Error starting MPI program. Terminating.\n");
    MPI_Abort(MPI_COMM_WORLD, rc);
  }

  // Get information about the number of tasks and which
  // rank this task is.
  MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  printf ("Number of tasks= %d My rank= %d\n", numtasks,rank);
  for (int round = 0; round < 2;++round)
    { // Each task delays for an amount of time and then barrier’s
      double delaySecs = drand48() * 10.0;
      int    sleepSecs = (int)delaySecs;
      cout << "Rank " << rank
           << " delaying for " << sleepSecs << " seconds" << endl;
      sleep(sleepSecs);
      MPI_Barrier(MPI_COMM_WORLD);
    }
  // Now each rank chooses a random value and distributed to all
  // other ranks using allGather
  int    groupSize = 0;
  MPI_Comm_size(MPI_COMM_WORLD, &groupSize);
  double* pGatherBuffer = new double[groupSize];
  // Set my own value in the buffer
  double myValue = drand48();
  cout << "Rank " << rank << " reporting value " << myValue
       << " groupSize " << groupSize
       << endl;
  MPI_Allgather(&myValue, 1, MPI_DOUBLE, // These 3 are my data
                pGatherBuffer, 1, MPI_DOUBLE, // Receive buffer
                MPI_COMM_WORLD);
  // To reduce amount of output, only rank 0 reports the results
  if (rank == 0)
    {
      for (int i = 0; i < numtasks; ++i)
        {
```

Program testMPI3.cc

```
          cout << "Rank " << i << " reports " << pGatherBuffer[i]
               << endl;
        }
    }
  // Finally try MPI_Allreduce to get a global minimum
  double minValue = 0;  // Global min calculated by allreduce
  MPI_Allreduce(&myValue, &minValue, 1, MPI_DOUBLE, MPI_MIN, MPI_COMM_WORLD);
  cout << "Rank " << rank << " exiting normally, global min is "
       << minValue << endl;
  MPI_Finalize();
}
```

Program testMPI3.cc (continued)