```
1  // Demonstrate simple MPI program
2  // George F. Riley, Georgia Tech, Fall 2011
3
4
5  #include <iostream>
6  #include <stdio.h>
7  #include <stdlib.h>
8
9  #include "mpi.h"
10
11 using namespace std;
12
13 #define MSG_SIZE 1000
14 char buf[MSG_SIZE];  // Message contents
15
16 int main(int argc,char**argv)
17 {
18   int  numtasks, rank, rc;
19
20   rc = MPI_Init(&argc,&argv);
21   if (rc != MPI_SUCCESS) {
22     printf ("Error starting MPI program. Terminating.\n");
23     MPI_Abort(MPI_COMM_WORLD, rc);
24   }
25
26   MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
27   MPI_Comm_rank(MPI_COMM_WORLD,&rank);
28   printf ("Number of tasks= %d My rank= %d\n", numtasks,rank);
29   for (int round = 0; round < 100;++round)
30     {
31       if (rank == 0)
32         { // Rank zero sends first then receives, all other
33           // receive first then send
34           cout << "Rank " << rank
35                << " sending to rank" << rank + 1
36                << " round " << round << endl;
37           rc = MPI_Send(buf, sizeof(buf), MPI_CHAR, rank + 1,
38                         0, MPI_COMM_WORLD);
39           if (rc != MPI_SUCCESS)
40             {
41               cout << "Rank " << rank
42                    << " send failed, rc " << rc << endl;
43               MPI_Finalize();
44               exit(1);
45             }
46           MPI_Status status;
47           rc = MPI_Recv(buf, sizeof(buf), MPI_CHAR, MPI_ANY_SOURCE,
48                         0, MPI_COMM_WORLD, &status);
49           if (rc != MPI_SUCCESS)
50             {
51               cout << "Rank " << rank
52                    << " recv failed, rc " << rc << endl;
53               MPI_Finalize();
54               exit(1);
55             }
56           int count = 0;
```

Program testMPI.cc

```
57            MPI_Get_count(&status, MPI_CHAR, &count);
58            cout << "Rank " << rank
59                 << " received " << count << " bytes from "
60                 << status.MPI_SOURCE << endl;
61          }
62        else
63          {
64            MPI_Status status;
65            rc = MPI_Recv(buf, sizeof(buf), MPI_CHAR, MPI_ANY_SOURCE,
66                          0, MPI_COMM_WORLD, &status);
67            if (rc != MPI_SUCCESS)
68              {
69                cout << "Rank " << rank
70                     << " recv failed, rc " << rc << endl;
71                MPI_Finalize();
72                exit(1);
73              }
74            // Now send to next rank (0 if we are last rank)
75            int nextRank = rank + 1;
76            if (nextRank == numtasks) nextRank = 0;
77            cout << "Rank " << rank
78                 << " sending to rank" << nextRank
79                 << " round " << round << endl;
80            rc = MPI_Send(buf, sizeof(buf), MPI_CHAR, nextRank,
81                          0, MPI_COMM_WORLD);
82            if (rc != MPI_SUCCESS)
83              {
84                cout << "Rank " << rank
85                     << " send failed, rc " << rc << endl;
86                MPI_Finalize();
87                exit(1);
88              }
89          }
90      }
91    cout << "Rank " << rank << " exiting normally" << endl;
92    MPI_Finalize();
93  }
```

Program testMPI.cc (continued)

```
1  // Demonstrate simple MPI program
2  // This one uses non-blocking ISend/Irecv
3  // George F. Riley, Georgia Tech, Fall 2011
4
5
6  #include <iostream>
7  #include <stdio.h>
8  #include <stdlib.h>
9
10 #include "mpi.h"
11
12 using namespace std;
13
14 #define MSG_SIZE 1000
15 char buf[MSG_SIZE];  // Message contents
16
17 int main(int argc,char**argv)
18 {
19   int  numtasks, rank, rc;
20
21   rc = MPI_Init(&argc,&argv);
22   if (rc != MPI_SUCCESS) {
23     printf ("Error starting MPI program. Terminating.\n");
24     MPI_Abort(MPI_COMM_WORLD, rc);
25   }
26
27   MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
28   MPI_Comm_rank(MPI_COMM_WORLD,&rank);
29   printf ("Number of tasks= %d My rank= %d\n", numtasks,rank);
30   for (int round = 0; round < 100;++round)
31     {
32       if (rank == 0)
33         { // Rank zero sends first then receives, all other
34           // receive first then send
35           cout << "Rank " << rank
36                << " sending to rank" << rank + 1
37                << " round " << round << endl;
38           MPI_Request request;
39           rc = MPI_Isend(buf, sizeof(buf), MPI_CHAR, rank + 1,
40                          0, MPI_COMM_WORLD, &request);
41           // Presumably more processing here....
42           // Eventually must call either MPI_Wait (wait for transfer
43           // complete, or MPI_Test (test if transfer complete);
44           // The sender must insure the data buffer (buf in this example)
45           // is unchanged until the transfer has completed.
46           if (rc != MPI_SUCCESS)
47             {
48               cout << "Rank " << rank
49                    << " send failed, rc " << rc << endl;
50               MPI_Finalize();
51               exit(1);
52             }
53           MPI_Status status;
54           MPI_Wait(&request, &status);
55           // At this point the send is complete and the buffer can be
56           // reused.
```

Program testMPI2.cc

```
57            // Now queue the receive request, also non-blocking
58            rc = MPI_Irecv(buf, sizeof(buf), MPI_CHAR, MPI_ANY_SOURCE,
59                           0, MPI_COMM_WORLD, &request);
60            if (rc != MPI_SUCCESS)
61              {
62                cout << "Rank " << rank
63                     << " recv failed, rc " << rc << endl;
64                MPI_Finalize();
65                exit(1);
66              }
67            int count = 0;
68            // Presumably more work here; the receive has not completed
69            // yet and the data is not yet available.
70            // Now wait for the Irecv to complete.  You can use either
71            // MPI_Wait or MPI_Test or MPI_Test_Any
72            MPI_Wait(&request, &status);
73            // The receive is now completed and data available.
74            MPI_Get_count(&status, MPI_CHAR, &count);
75            cout << "Rank " << rank
76                 << " received " << count << " bytes from "
77                 << status.MPI_SOURCE << endl;
78          }
79        else
80          {
81            MPI_Status status;
82            rc = MPI_Recv(buf, sizeof(buf), MPI_CHAR, MPI_ANY_SOURCE,
83                          0, MPI_COMM_WORLD, &status);
84            if (rc != MPI_SUCCESS)
85              {
86                cout << "Rank " << rank
87                     << " recv failed, rc " << rc << endl;
88                MPI_Finalize();
89                exit(1);
90              }
91            // Now send to next rank (0 if we are last rank)
92            int nextRank = rank + 1;
93            if (nextRank == numtasks) nextRank = 0;
94            cout << "Rank " << rank
95                 << " sending to rank" << rank + 1
96                 << " round " << round << endl;
97            rc = MPI_Send(buf, sizeof(buf), MPI_CHAR, nextRank,
98                          0, MPI_COMM_WORLD);
99            if (rc != MPI_SUCCESS)
100             {
101               cout << "Rank " << rank
102                    << " send failed, rc " << rc << endl;
103               MPI_Finalize();
104               exit(1);
105             }
106         }
107     }
108   cout << "Rank " << rank << " exiting normally" << endl;
109   MPI_Finalize();
110 }
```

Program testMPI2.cc (continued)

```
1  // Demonstrate simple MPI barriers and collectives
2  // This one uses non-blocking ISend/Irecv
3  // George F. Riley, Georgia Tech, Fall 2011
4
5
6  #include <iostream>
7  #include <stdio.h>
8  #include <stdlib.h>
9
10 #include "mpi.h"
11
12 using namespace std;
13
14 int main(int argc,char**argv)
15 {
16   int  numtasks, rank, rc;
17
18   // As always, we must call MPI_Init
19   rc = MPI_Init(&argc,&argv);
20   if (rc != MPI_SUCCESS) {
21     printf ("Error starting MPI program. Terminating.\n");
22     MPI_Abort(MPI_COMM_WORLD, rc);
23   }
24
25   // Get information about the number of tasks and which
26   // rank this task is.
27   MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
28   MPI_Comm_rank(MPI_COMM_WORLD,&rank);
29   printf ("Number of tasks= %d My rank= %d\n", numtasks,rank);
30   for (int round = 0; round < 2;++round)
31     { // Each task delays for an amount of time and then barrier's
32       double delaySecs = drand48() * 10.0;
33       int    sleepSecs = (int)delaySecs;
34       cout << "Rank " << rank
35            << " delaying for " << sleepSecs << " seconds" << endl;
36       sleep(sleepSecs);
37       MPI_Barrier(MPI_COMM_WORLD);
38     }
39   // Now each rank chooses a random value and distributed to all
40   // other ranks using allGather
41   int    groupSize = 0;
42   MPI_Comm_size(MPI_COMM_WORLD, &groupSize);
43   double* pGatherBuffer = new double[groupSize];
44   // Set my own value in the buffer
45   double myValue = drand48();
46   cout << "Rank " << rank << " reporting value " << myValue
47        << " groupSize " << groupSize
48        << endl;
49   MPI_Allgather(&myValue, 1, MPI_DOUBLE, // These 3 are my data
50                 pGatherBuffer, 1, MPI_DOUBLE, // Receive buffer
51                 MPI_COMM_WORLD);
52   // To reduce amount of output, only rank 0 reports the results
53   if (rank == 0)
54     {
55       for (int i = 0; i < numtasks; ++i)
56         {
```

Program testMPI3.cc

```
57            cout << "Rank " << i << " reports " << pGatherBuffer[i]
58                 << endl;
59          }
60      }
61    // Finally try MPI_Allreduce to get a global minimum
62    double minValue = 0;  // Global min calculated by allreduce
63    MPI_Allreduce(&myValue, &minValue, 1, MPI_DOUBLE, MPI_MIN, MPI_COMM_WORLD);
64    cout << "Rank " << rank << " exiting normally, global min is "
65         << minValue << endl;
66    MPI_Finalize();
67  }
```

Program testMPI3.cc (continued)