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

#include "vector_operations.h"

using namespace std;


int main(int argc , char **argv)
{
    MPI_Init(&argc, &argv);     // Initializes the MPI execution environment
    MPI_Comm const icomm(MPI_COMM_WORLD);
    int myrank;
    MPI_Comm_rank(icomm, &myrank);

    int n = 20;
    vector<double> x(n);
    vector<double> y = x;
    for (int i = 0; i < n; ++i)
    {
        x[i] = myrank*100 + (i % 5)*10 + i;
        y[i] = 1.0/(x[i]);
    }

    if(myrank == 0) // so scalar product is well defined (avoid division by 0)
        y[0] = 0;


    // -------------------- E5 --------------------
    if (myrank == 0) cout << "-------------------- E5 --------------------" << endl;
    DebugVector(x, icomm);


    cout.flush();
    MPI_Barrier(icomm);
    // -------------------- E6 --------------------
    if (myrank == 0) cout << "-------------------- E6 --------------------" << endl;
    double scalar_product = par_scalar(x, y, icomm);

    if (myrank == 0)
    {
        cout << "<x,y> = " << scalar_product << endl << endl;
    }


    cout.flush();
    MPI_Barrier(icomm);
    // -------------------- E7 --------------------
    if (myrank == 0) cout << "-------------------- E7 --------------------" << endl;
    double xmin, xmax;
    par_minmax(x, xmin, xmax, icomm);

    if (myrank == 0)
    {
        cout << "Global min: " << xmin << endl;
        cout << "Global max: " << xmax << endl << endl;
    }

    cout.flush();
    MPI_Barrier(icomm);
    // -------------------- E8 --------------------
    if (myrank == 0) cout << "-------------------- E8 --------------------" << endl;
    vector<double> x_new(n);


    cout.flush();
    MPI_Barrier(icomm);
```

```
   69:     // All to all
   70:     if (myrank == 0) cout << "----- All to all -----" << endl;
   71:     auto sendbuf = x.data();
   72:     int sendcount = 5;
   73:     auto recvbuf = x_new.data();
   74:     int recvcount = 5;
   75:     MPI_Alltoall(sendbuf, sendcount, MPI_DOUBLE, recvbuf, recvcount, MPI_DOUBLE, ico
mm);
   76:
   77:     DebugVector(x_new, icomm);
   78:
   79:
   80:     cout.flush();
   81:     MPI_Barrier(icomm);
   82:     // All to all v
   83:     if (myrank == 0) cout << "----- All to all v -----" << endl;
   84:     int sendcounts[4] = {5, 5, 5, 5};
   85:     int senddispls[4] = {0, 5, 10, 15};
   86:     int rcvcounts[4] = {5, 5, 5, 5};
   87:     int rcvdispls[4] = {0, 5, 10, 15};
   88:     MPI_Alltoallv(x.data(), sendcounts, senddispls, MPI_DOUBLE, x_new.data(), rcvcou
nts, rcvdispls, MPI_DOUBLE, icomm);
   89:
   90:     DebugVector(x_new, icomm);
   91:
   92:
   93:     cout.flush();
   94:     MPI_Barrier(icomm);
   95:     // All to all (in place), sendcount and sendtype are ignored
   96:     if (myrank == 0) cout << "----- All to all (in place) -----" << endl;
   97:     MPI_Alltoall(MPI_IN_PLACE, sendcount, MPI_DOUBLE, x.data(), recvcount, MPI_DOUBL
E, icomm);
   98:
   99:     DebugVector(x, icomm);
  100:
  101:
  102:
  103:
  104:     MPI_Finalize();     // Terminates MPI execution environment
  105:
  106:     return 0;
  107: }
```

```cpp
#include "vector_operations.h"
#include <cassert>
#include <cfloat>

void DebugVector(const vector<double> &xin, const MPI_Comm &icomm)
{
    int myrank, numprocs;
    MPI_Comm_rank(icomm, &myrank);                // my MPI-rank
    MPI_Comm_size(icomm, &numprocs);              // #MPI processes
    int ierr;


    int n = xin.size();


    int chosen_process;
    for (int k = 0; k < numprocs; ++k)
    {
        MPI_Barrier(icomm);

        if (myrank == 0)
        {
            cout << "Choose next process: ";
            cin >> chosen_process;

        }
        ierr = MPI_Bcast(&chosen_process, 1, MPI_INT, 0, icomm); // broadcast value of "chosen_process" to all processes
        assert(ierr == 0);

        MPI_Barrier(icomm);

        if (chosen_process == myrank)
        {
            for (int i = 0; i < n; ++i)
            {
                cout << "x_" << i << " = " << xin[i] << "\t(Process " << myrank << ")" << endl;
            }
            cout.flush();
        }
    }
    return;
}


double par_scalar(const vector<double> &x, const vector<double> &y, const MPI_Comm &icomm)
{
    int n = x.size();
    assert(n == (int)y.size());

    double sum = 0.0;
    double local_sum = 0.0;


    for (int i = 0; i < n; ++i)
    {
        local_sum += x[i]*y[i];
    }

    int ierr = MPI_Allreduce(&local_sum, &sum, 1, MPI_DOUBLE, MPI_SUM, icomm);  // reduce local sums to global sum
    assert(ierr == 0);


    return sum;
```

```
    65: }
    66:
    67:
    68: void par_minmax(const vector<double> &x, double &global_min, double &global_max, con
st MPI_Comm &icomm)
    69: {
    70:     int myrank, numprocs;
    71:     MPI_Comm_rank(icomm, &myrank);                // my MPI-rank
    72:     MPI_Comm_size(icomm, &numprocs);              // #MPI processes
    73:
    74:     int n = x.size();
    75:
    76:     double local_min = DBL_MAX;
    77:     double local_max = -DBL_MAX;
    78:
    79:     for (int i = 0; i < n; ++i)
    80:     {
    81:         if (x[i] < local_min)
    82:             local_min = x[i];
    83:         if (x[i] > local_max)
    84:             local_max = x[i];
    85:     }
    86:
    87:     vector<double> local_mins(numprocs);
    88:     vector<double> local_maxs(numprocs);
    89:     MPI_Gather(&local_min, 1, MPI_DOUBLE, local_mins.data(), 1, MPI_DOUBLE, 0, icomm
);
    90:     MPI_Gather(&local_max, 1, MPI_DOUBLE, local_maxs.data(), 1, MPI_DOUBLE, 0, icomm
);
    91:
    92:     if (myrank == 0)
    93:     {
    94:         global_min = DBL_MAX;
    95:         global_max = -DBL_MAX;
    96:
    97:         for (int i = 0; i < numprocs; ++i)
    98:         {
    99:             if (local_mins[i] < global_min)
   100:                 global_min = local_mins[i];
   101:             if (local_maxs[i] > global_max)
   102:                 global_max = local_maxs[i];
   103:         }
   104:     }
   105:
   106:     MPI_Bcast(&global_min, 1, MPI_DOUBLE, 0, icomm); // make sure every process is u
p to date
   107:     MPI_Bcast(&global_max, 1, MPI_DOUBLE, 0, icomm);
   108:
   109:     return;
   110: }
```

Exchange !?

```
    1: #include <mpi.h>
    2: #include <vector>
    3:
    4: using namespace std;
    5:
    6: void DebugVector(const vector<double> &xin, const MPI_Comm &icomm);
    7:
    8: double par_scalar(const vector<double> &x, const vector<double> &y, const MPI_Comm &
icomm);
    9:
   10: void par_minmax(const vector<double> &x, double &global_min, double &global_max, con
st MPI_Comm &icomm);
```