```cpp
#include <iostream>
#include <mpi.h>
#include <vector>
#include "VecFuncs.h"
using namespace std;


int main(int argc, char** argv)
{
    MPI_Init(&argc, &argv);
    MPI_Comm 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);

    //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;
    }

    // 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;
    }

    // E8
    if (myrank == 0) cout << "E8" << endl;
    vector<double> x_new(n);

    // All to all
    if (myrank == 0) cout << "All to all" << endl;
    MPI_Alltoall(x.data(), 5, MPI_DOUBLE, x_new.data(), 5, MPI_DOUBLE, icomm);
    DebugVector(x_new, icomm);

    if (myrank == 0) cout << "All to all (in place)" << endl;
    MPI_Alltoall(MPI_IN_PLACE, 0, MPI_DOUBLE, x.data(), 5, MPI_DOUBLE, icomm);

    DebugVector(x, icomm);


    MPI_Finalize();
    return 0;
}
```

```cpp
#include "VecFuncs.h"
#include <vector>
#include <iostream>
#include <cassert>
#include <cfloat>

void DebugVector(const std::vector<double>& xin, MPI_Comm icomm)
{
    int rank, size;
    MPI_Comm_rank(icomm, &rank);
    MPI_Comm_size(icomm, &size);
    int ierr;

    int active_rank = -1;

    for (int step = 0; step < size; ++step)
    {
        if (rank == 0)
        {
            std::cout << "Enter rank to display vector: " << std::endl;
            std::cin >> active_rank;
        }

        ierr = MPI_Bcast(&active_rank, 1, MPI_INT, 0, icomm);
        assert(ierr == 0);

        MPI_Barrier(icomm);

        if (rank == active_rank)
        {
            std::cout << "Output from process " << rank  << std::endl;

            for (size_t i = 0; i < xin.size(); ++i)
            {
                std::cout << "xin[" << i << "] = " << xin[i] << std::endl;
            }

            std::cout << std::endl;
        }

        MPI_Barrier(icomm);
    }
}

double par_scalar(const std::vector<double> &x, const std::vector<double> &y, const MPI_Comm &comm) {
    assert(x.size()==y.size());

    double local_sum = 0.0;
    for (int k = 0; k < x.size(); ++k) {
        local_sum += x[k] * y[k];
    }

    double global_sum = 0.0;
    int mpi_error = MPI_Allreduce(&local_sum, &global_sum, 1, MPI_DOUBLE, MPI_SUM, comm);
    assert(mpi_error == 0);

    return global_sum;
}

void par_minmax(std::vector<double> &x, double &min_value, double &max_value, const MPI_Comm &icomm)
{
        int myrank;
    MPI_Comm_rank(icomm, &myrank);

    int local_n = x.size();
```

```
   66:         int global_offset = myrank * local_n;
   67:
   68:         struct {double value; int idx;} local_min, local_max, global_min, global_max
;       // global index
   69:
   70:         local_min.value = local_max.value = x[0];
   71:     local_min.idx = local_max.idx = global_offset;
   72:
   73:         // finding local min/max with the corresponding global index
   74:         for (int i = 1; i < local_n; ++i) {
   75:         int global_idx = global_offset + i;
   76:             if (x[i] < local_min.value){
   77:                 local_min.value = x[i];
   78:                 local_min.idx = global_idx;
   79:             }
   80:             if (x[i] > local_max.value){
   81:                 local_max.value = x[i];
   82:                 local_max.idx = global_idx;
   83:             }
   84:         }
   85:
   86:         // reduction to the global one including the global index (need it later for
 interchanging)
   87:         MPI_Allreduce(&local_min, &global_min, 1, MPI_DOUBLE_INT, MPI_MINLOC, icomm)
;
   88:         MPI_Allreduce(&local_max, &global_max, 1, MPI_DOUBLE_INT, MPI_MAXLOC, icomm)
;
   89:         min_value = global_min.value;
   90:         max_value = global_max.value;
   91:
   92:         // calculating the process and the local index for interchanging the min and
 max value
   93:         int rank_min = global_min.idx / local_n;
   94:         int rank_max = global_max.idx / local_n;
   95:         int local_min_idx = global_min.idx % local_n;
   96:         int local_max_idx = global_max.idx % local_n;
   97:
   98:         // interchanging
   99:         if (rank_min == rank_max){
  100:                 std::swap(x[local_min_idx], x[local_max_idx]);
  101:         }
  102:         else {
  103:             double recv_value;
  104:             if (myrank == rank_min){
  105:                 MPI_Sendrecv(&x[local_min_idx], 1, MPI_DOUBLE, rank_max, 0, &recv_va
lue, 1, MPI_DOUBLE, rank_max, 0, icomm, MPI_STATUS_IGNORE);
  106:                 x[local_min_idx] = recv_value;
  107:             }
  108:             if (myrank == rank_max){
  109:                 MPI_Sendrecv(&x[local_max_idx], 1, MPI_DOUBLE, rank_min, 0, &recv_va
lue, 1, MPI_DOUBLE, rank_min, 0, icomm, MPI_STATUS_IGNORE);
  110:                 x[local_max_idx] = recv_value;
  111:             }
  112:
  113:         }
  114:
  115:
  116:         return;
  117: }
```

```
    1: #pragma once
    2:
    3: #include <vector>
    4: #include <mpi.h>
    5: #include <iostream>
    6: #include <cassert>
    7: #include <cfloat>
    8:
    9: void DebugVector(const std::vector<double>& xin, MPI_Comm icomm);
   10:
   11: double par_scalar(const std::vector<double> &x, const std::vector<double> &y, const
MPI_Comm &icomm);
   12:
   13: void par_minmax(std::vector<double> &x, double &min_val, double &max_val, const MPI_
Comm &icomm);
```