```cpp
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "mpi.h" // message passing interface
using namespace std;

//
// Program 2
// Whack-An-Orc - 10 points
//  Rachel Burke
//

int main(int argc, char *argv[])
{

    int my_rank;       // my CPU number for this process
    int p;             // number of CPUs that we have
    int source;        // rank of the sender
    int dest;          // rank of destination
    int tag = 0;       // message number
    char message[100]; // message itself
    MPI_Status status; // return status for receive

    // Start MPI
    MPI_Init(&argc, &argv);

    // Find out my rank!
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    // Find out the number of processes!
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    // THE REAL PROGRAM IS HERE

    // Pseudo-random number generator seeded at 1251
    srand(1251);

    // Initializing overall variables and a big array
    int max = 0, min = 0;
    double average = 0;
    int n = 100000;
    int *big_array = new int[n];

    // Setting pseudo-random data in the big array
    if (my_rank == 0)
        for (int x = 0; x < n; x++)
            big_array[x] = rand() % 50;

    // Divide up the problem
    int local_n = n / p;
    int *local_array = new int[local_n];

    // Scattering the array into pieces to each process
    MPI_Scatter(&big_array[0], local_n, MPI_INT, local_array, local_n, MPI_INT, 0,
    MPI_COMM_WORLD);

    // Do the local work

    // Initializing local variables
    int local_max = 0, local_min = 0;
    double local_average = 0;

    // Local max will be largest item in the local array
    // Local min will be smallest item in the local array
    // Local average will be sum of local array divided by local_n.....
    // BUT we want overall in the end so divide by n.....
    // AND use the distributive property!
    for (int x = 0; x < local_n; x++)
```

```
    {
        if (local_array[x] > local_max)
            local_max = local_array[x];
        if (local_array[x] < local_min)
            local_min = local_array[x];
        local_average += ((double) local_array[x] / (double) n);
    }

    delete [] local_array;

    // Reduce the results
    MPI_Allreduce(&local_max, &max, 1, MPI_INT, MPI_MAX, MPI_COMM_WORLD);
    MPI_Allreduce(&local_min, &min, 1, MPI_INT, MPI_MIN, MPI_COMM_WORLD);
    MPI_Allreduce(&local_average, &average, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);

    // Print the results
    if (my_rank == 0)
        cout << "Max: " << max << "\nMin: " << min << "\nAverage: " << average << endl;

    // Shut down MPI
    MPI_Finalize();

    delete [] big_array;

    return 0;
}
```