```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>
#include <algorithm>
// #include <execution>   // only include if the compiler supports it
#include <omp.h>

using namespace std;

// OpenMP parallel computations (already implemented)
void computeMeansVectorParallel(const vector<int> &v, double &arith, double &geom, double &harm) {
    int n = v.size();
    double sum = 0.0, logSum = 0.0, recSum = 0.0;

    #pragma omp parallel for reduction(+:sum,logSum,recSum)
    for (int i = 0; i < n; ++i) {
        int x = v[i];
        sum += x;
        logSum += log(x);
        recSum += 1.0 / x;
    }

    arith = sum / n;
    geom  = exp(logSum / n);
    harm  = n / recSum;
}

double computeStdDevParallel(const vector<int> &v, double mean) {
    int n = v.size();
    double sumSq = 0.0;

    #pragma omp parallel for reduction(+:sumSq)
    for (int i = 0; i < n; ++i) {
        double diff = v[i] - mean;
        sumSq += diff * diff;
    }

    return sqrt(sumSq / n);
}

int main() {
    cout << "Exercise Sheet 5: Task 2\n";
    // Read data
    vector<int> v;
    ifstream fin("data_1.txt");
    int x;
    while (fin >> x) v.push_back(x);
    fin.close();

    cout << "Data size: " << v.size() << "\n";

    double t_start = omp_get_wtime();

    // OpenMP parallel computations
    #pragma omp parallel
    {
        #pragma omp single
        cout << "Using " << omp_get_num_threads() << " OpenMP threads\n";
    }

    int minVal = v[0], maxVal = v[0];
    #pragma omp parallel
    {
        int local_min = v[0], local_max = v[0];
        #pragma omp for nowait
        for (int i = 0; i < v.size(); ++i) {
```

reduction(max: gmax, min: gmin)

```
68:             if (v[i] < local_min) local_min = v[i];
69:             if (v[i] > local_max) local_max = v[i];
70:         }
71:         #pragma omp critical
72:         {
73:             if (local_min < minVal) minVal = local_min;
74:             if (local_max > maxVal) maxVal = local_max;
75:         }
76:     }
77:
78:     double arith, geom, harm;
79:     computeMeansVectorParallel(v, arith, geom, harm);
80:
81:     double stddev = computeStdDevParallel(v, arith);
82:
83:     double t_end = omp_get_wtime();
84:
85:     cout << "OpenMP results:\n";
86:     cout << "Min: " << minVal << ", Max: " << maxVal << "\n";
87:     cout << "Arithmetic mean: " << arith
88:          << ", Geometric mean: " << geom
89:          << ", Harmonic mean: " << harm << "\n";
90:     cout << "Standard deviation: " << stddev << "\n";
91:     cout << "Elapsed time (s): " << t_end - t_start << "\n\n";
92:
93:     cout << "\nNote: The C++17 execution::par version is implemented in the main fil
e, "
94:         "but my Mac's Clang compiler does not support parallel execution policies.\n
";
95:
96:     // C++17 execution policy version
97:     /*
98:     double t_ep_start = omp_get_wtime();
99:
100:    auto p = minmax_element(execution::par, v.begin(), v.end());
101:    double min_ep = *p.first;
102:    double max_ep = *p.second;
103:
104:    double sum_ep = reduce(execution::par, v.begin(), v.end(), 0.0);
105:    double ar_ep = sum_ep / v.size();
106:
107:    double geom_ep = transform_reduce(
108:        execution::par,
109:        v.begin(), v.end(),
110:        0.0,
111:        plus<>(),
112:        [n=v.size()](int val){ return log(val); }
113:    );
114:    geom_ep = exp(geom_ep / v.size());
115:
116:    double sum_inv_ep = transform_reduce(
117:        execution::par,
118:        v.begin(), v.end(),
119:        0.0,
120:        plus<>(),
121:        [](int val){ return 1.0 / val; }
122:    );
123:    double harm_ep = v.size() / sum_inv_ep;
124:
125:    double sum_sq_ep = transform_reduce(
126:        execution::par,
127:        v.begin(), v.end(),
128:        0.0,
129:        plus<>(),
130:        [ar_ep](int val){ double d = val - ar_ep; return d*d; }
131:    );
132:    double std_ep = sqrt(sum_sq_ep / v.size());
133:
```

```
134:     double t_ep_end = omp_get_wtime();
135:
136:     cout << "Execution policy results:\n";
137:     cout << "Min: " << min_ep << ", Max: " << max_ep << "\n";
138:     cout << "Arithmetic mean: " << ar_ep
139:          << ", Geometric mean: " << geom_ep
140:          << ", Harmonic mean: " << harm_ep << "\n";
141:     cout << "Standard deviation: " << std_ep << "\n";
142:     cout << "Elapsed time (s): " << t_ep_end - t_ep_start << "\n";
143:     */
144:     return 0;
145: }
```