```cpp
#include "file_io.h"
#include <cassert>
#include <fstream>
#include <iostream>
#include <stdexcept>
#include <string>
#include <vector>
using namespace std;

// [Str10, p.364]
void fill_vector(istream& istr, vector<short>& v)
{
    double d=0;
    while ( istr >> d) v.push_back(d); // Einlesen
    if (!istr.eof())
    { // Fehlerbehandlung
        cout << " Error handling \n";
        if ( istr.bad() )  throw runtime_error("Schwerer Fehler in istr");
        if ( istr.fail() )   // Versuch des Aufraeumens
        {
            cout << " Failed in reading all data.\n";
            istr.clear();
        }
    }
    v.shrink_to_fit();                  // C++11
    return;
}


void read_vector_from_file(const string& file_name, vector<short>& v)
{
    ifstream fin(file_name);            // Oeffne das File im ASCII-Modus
    if( fin.is_open() )                 // File gefunden:
    {
       v.clear();                       // Vektor leeren
       fill_vector(fin, v);
    }
    else                                // File nicht gefunden:
    {
        cout << "\nFile " << file_name << " has not been found.\n\n" ;
        assert( fin.is_open() && "File not found." );       // exeption handling for the poor programmer
    }

 return;
}

void write_vector_to_file(const string& file_name, const vector<double>& v)
{
    ofstream fout(file_name);           // Oeffne das File im ASCII-Modus
    if( fout.is_open() )
    {
       for (unsigned int k=0; k<v.size(); ++k)
       {
          fout << v.at(k) << endl;
       }
    }
    else
    {
        cout << "\nFile " << file_name << " has not been opened.\n\n" ;
        assert( fout.is_open() && "File not opened."  );         // exeption handling for the poor programmer
    }

 return;
}

```

```
    1: #ifndef FILE_IO_H_INCLUDED
    2: #define FILE_IO_H_INCLUDED
    3:
    4: #include <string>
    5: #include <vector>
    6: //using namespace std;
    7:
    8:
    9: /**
   10:   This function opens the ASCII-file named @p file_name and reads the
   11:   double data into the C++ vector @p v.
   12:   If the file  @p file_name does not exist then the code stops with an appropriate m
essage.
   13:   @param[in]    file_name   name of the ASCII-file
   14:   @param[out]   v           C++ vector with double values
   15: */
   16:
   17: void read_vector_from_file(const std::string& file_name, std::vector<short>& v);
   18:
   19:
   20: /**
   21:   This function opens the ASCII-file named @p file_name and rewrites its with the
   22:   double data from the C++ vector @p v.
   23:   If there are problems in opening/generating file  @p file_name
   24:   then the code stops with an appropriate message.
   25:   @param[in]    file_name   name of the ASCII-file
   26:   @param[in]    v           C++ vector with double values
   27: */
   28:
   29: void write_vector_to_file(const std::string& file_name, const std::vector<double>& v
);
   30:
   31: /**
   32:   Fills the double-vector @p v with data from an input stream @p istr until this inp
ut stream
   33:   ends regularily. The vector is cleared and its memory is automatically allocated.
   34:   @param[in]    istr        input stream
   35:   @param[out]    v           C++ vector with double values
   36:   @warning     An exception is thrown in case of wrong data format or corrupted data
.
   37: */
   38: void fill_vector(std::istream& istr, std::vector<double>& v);
   39:
   40: #endif // FILE_IO_H_INCLUDED
```

```cpp
#include "file_io.h"
#include "means.h"
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <cmath>
#include <execution>
#include <omp.h>
using namespace std;

int main()
{
    cout << "File einlesen." << endl;

    const string name("data_1.txt");               // name of input file
    const string name2("out_1.txt");                // name of output file
    vector<short> a;    //-2^15 to 2^15-1 fits the values from the file
    double  min, max, ar, ge, ha, std;


    read_vector_from_file(name, a);
    const unsigned size = a.size();
double t_omp_start = omp_get_wtime();
    min = a[0];
    max = a[0];

#pragma omp parallel for reduction(min:min) reduction(max:max)
    for (int i = 0; i < static_cast<int>(size); ++i) {
        if (a[i] < min)
        {
            min = a[i];
        }
        if (a[i] > max) {
            max = a[i];
        }

    }



    means_vector(a, ar, ge, ha);

    std = 0.0;
    #pragma omp parallel for reduction(+:std)
    for(unsigned int i = 0; i < size; i++)
    {
        std += pow(a.at(i)-ar,2);
    }
    std = sqrt(std/size);

     double t_omp_end = omp_get_wtime();
    double time_omp = t_omp_end - t_omp_start;

    cout << "min: " << min << ", max: " << max << endl;
    cout << "Arithmetic mean: " << ar
         << ", geometric mean: " << ge
         << ", harmonic mean: " << ha  << endl;
    cout << "std: " << std << endl;

    vector<double> results = {min, max, ar, ge, ha, std};
    write_vector_to_file(name2, results);



    //C++17 execution policies

    double min_ep = 0.0, max_ep = 0.0, ar_ep = 0.0, ge_ep = 0.0, ha_ep = 0.0, std_ep
```

```
= 0.0;
   69:
   70:     double t_ep_start = omp_get_wtime();
   71:
   72:
   73:     auto p = minmax_element(execution::par, a.begin(), a.end());
   74:     min_ep = *p.first;
   75:     max_ep = *p.second;
   76:
   77:     double sum_ep = reduce(
   78:         execution::par,
   79:         a.begin(), a.end(),
   80:         0.0
   81:     );
   82:     ar_ep = sum_ep / size;
   83:
   84:     ge_ep = transform_reduce(
   85:         execution::par,
   86:         a.begin(), a.end(),
   87:         1.0,
   88:         multiplies<>(),
   89:         [size](short x){
   90:             return pow((double)x, 1.0 / size);
   91:         }
   92:     );
   93:
   94:     double sum_inv_ep = transform_reduce(
   95:         execution::par,
   96:         a.begin(), a.end(),
   97:         0.0,
   98:         plus<>(),
   99:         [](short x){ return 1.0 / (double)x; }
  100:     );
  101:     ha_ep = size / sum_inv_ep;
  102:
  103:     double sum_sq_ep = transform_reduce(
  104:         execution::par,
  105:         a.begin(), a.end(),
  106:         0.0,
  107:         plus<>(),
  108:         [ar_ep](short x){
  109:             double d = x - ar_ep;
  110:             return d * d;
  111:         }
  112:     );
  113:     std_ep = sqrt(sum_sq_ep / size);
  114:
  115:
  116:     double t_ep_end = omp_get_wtime();
  117:     double time_ep = t_ep_end - t_ep_start;
  118:
  119:     cout << "min: " << min_ep << ", max: " << max_ep << endl;
  120:     cout << "Arithmetic mean: " << ar_ep
  121:          << ", geometric mean: " << ge_ep
  122:          << ", harmonic mean: " << ha_ep  << endl;
  123:     cout << "std: " << std_ep << endl;
  124:
  125:
  126:     cout << "OpenMP per run:              " << (time_omp) << endl;
  127:     cout << "Execution policy per run:    " << (time_ep) << endl;
  128:     return 0;
  129: }
```

```cpp
#include <iostream>
#include <cmath>
#include <vector>
#include <omp.h>

using namespace std;

void means(int a, int b, int c, double &ar, double &ge, double &ha) {
    ar = (a+b+c) / 3.0;


    ge = pow(a,1.0/3.0) * pow(b,1.0/3.0) * pow(c,1.0/3.0); //do it instead of pow(a*b*c,1.0/3.0) to prevent integer overflow

    ha = 3.0 / (1.0/a +1.0/b +1.0/c);
}

void means_vector(const vector<short> &input, double &ar, double &ge, double &ha) {
    int size = input.size();

    if (size == 0) {
        cout << "Empty input" << endl;
        return;
    }

    ar = 0;
    ge = 1;
    ha = 0;
    #pragma omp parallel for reduction(+:ar,ha) reduction(*:ge)
    for (int i = 0; i < size; i++) {
        ar += input.at(i);
        ge *=  pow(input.at(i), 1.0 / size);
        ha += 1.0 / input.at(i);
    }

    ar /= size;
    ha = size / ha;
}
```

```
    1: #ifndef MEANS_H_INCLUDED
    2: #define MEANS_H_INCLUDED
    3:
    4: #include <vector>
    5:
    6: void means(int a, int b, int c, double &ar, double &ge, double &ha);
    7: void means_vector(const std::vector<short> &input, double &ar, double &ge, double &h
a);
    8:
    9: #endif // MEANS_H_INCLUDED
```