```cpp
#include "bsp_5_2_lib.h"

#include <cassert>
#include <fstream>
#include <execution>
#include <iostream>
#include <limits>
#include <stdexcept>
#include <string>
#ifdef _OPENMP
#include <omp.h>
#endif
#include <vector>
#include <cmath>
using namespace std;


void meandevvec(vector<int> x, float &a, float &g, float &h, float &d)
{
    a = 0;
#pragma omp parallel for default(none) shared(x) reduction(+:a)
    for(size_t k=1; k<=x.size(); ++k)
    {
        a = a+x.at(k-1);
    }
    a = a/x.size();

    g = 1;
#pragma omp parallel for default(none) shared(x) reduction(*:g)
    for(size_t k=1; k<=x.size(); ++k)
    {
        g = g*pow(x.at(k-1),1.0/x.size());
    }

    h = 0;
#pragma omp parallel for default(none) shared(x) reduction(+:h)
    for(size_t k=1; k<=x.size(); ++k)
    {
        h = h + 1.0/x.at(k-1);
    }
    h = x.size()/h;

    d = 0;
#pragma omp parallel for default(none) shared(x,a) reduction(+:d)
    for(size_t k=1; k<=x.size(); ++k)
    {
        d = d + pow(x.at(k-1)-a,2);
    }
    d = d/x.size();
    d = sqrt(d);

    return;
}


void minmaxvec(vector<int> &x, int &minv, int &maxv)
{
        minv = numeric_limits<int>::max();
    maxv = numeric_limits<int>::min();

#pragma omp parallel for default(none) shared(x) reduction(min:minv) reduction(max:maxv)
    for (size_t i = 0; i < x.size(); ++i)
    {
        minv = min(minv, x[i]);
        maxv = max(maxv, x[i]);
    }
        return;
```

```
   68: }
   69:
   70:
   71: void meandevmaxminvec(std::vector<int> &x, float &a, float &g, float &h, float &d, f
loat &minv, float &maxv)
   72: {
   73:         const size_t n = x.size();
   74:
   75:     float sum = reduce(execution::par, x.begin(), x.end());
   76:     float logsum = transform_reduce(execution::par, x.begin(), x.end(), 0.0, plus<>(
), [](float y){ return log(y); });
   77:     float invsum = transform_reduce(execution::par, x.begin(), x.end(), 0.0, plus<>(
), [](float y){ return 1.0/y; });
   78:
   79:     a = sum / n;
   80:     g = exp(logsum / n);
   81:     h = n / invsum;
   82:
   83:     // for calculating the deviation - splitting of the square in the sum
   84:     float sq_sum = transform_reduce(execution::par, x.begin(), x.end(), 0.0, plus<>(
), [](float y){ return y * y; });
   85:     d = sqrt(sq_sum / n - a*a);
   86:
   87:         auto [min_it, max_it] = minmax_element(execution::par, x.begin(), x.end());
   88:         minv = *min_it;
   89:         maxv = *max_it;
   90:
   91:         return;
   92: }
   93:
   94: // for reading a file and writing into a txt-file
   95: // [Str10, p.364]
   96: void fill_vector(istream& istr, vector<int>& v)
   97: {
   98:     int d=0;
   99:     while ( istr >> d) v.push_back(d); // Einlesen
  100:     if (!istr.eof())
  101:     { // Fehlerbehandlung
  102:         cout << " Error handling \n";
  103:         if ( istr.bad() )  throw runtime_error("Schwerer Fehler in istr");
  104:         if ( istr.fail() )   // Versuch des Aufraeumens
  105:         {
  106:             cout << " Failed in reading all data.\n";
  107:             istr.clear();
  108:         }
  109:     }
  110:     v.shrink_to_fit();                  // C++11
  111:     return;
  112: }
  113:
  114:
  115: void read_vector_from_file(const string& file_name, vector<int>& v)
  116: {
  117:     ifstream fin(file_name);            // Oeffne das File im ASCII-Modus
  118:     if( fin.is_open() )                 // File gefunden:
  119:     {
  120:        v.clear();                       // Vektor leeren
  121:        fill_vector(fin, v);
  122:     }
  123:     else                                // File nicht gefunden:
  124:     {
  125:         cout << "\nFile " << file_name << " has not been found.\n\n" ;
  126:         assert( fin.is_open() && "File not found." );       // exeption handling for
 the poor programmer
  127:     }
  128:
  129:  return;
  130: }
```

```
131:
132: void write_vector_to_file(const string& file_name, const vector<float>& v)
133: {
134:     ofstream fout(file_name);          // Oeffne das File im ASCII-Modus
135:     if( fout.is_open() )
136:     {
137:        for (unsigned int k=0; k<v.size(); ++k)
138:        {
139:           fout << v.at(k) << endl;
140:        }
141:     }
142:     else
143:     {
144:         cout << "\nFile " << file_name << " has not been opened.\n\n" ;
145:         assert( fout.is_open() && "File not opened."  );         // exeption handlin
g for the poor programmer
146:     }
147:
148:  return;
149: }
```

```
    1: #ifndef BSP_5_2_LIB_H_INCLUDED
    2: #define BSP_5_2_LIB_H_INCLUDED
    3:
    4: #include <iostream>
    5: #include <omp.h>
    6: #include <vector>
    7:
    8:
    9: /** \brief Funktion zur Berechnung des arithmetischen, geometrischen und harmonische
n Mittels und der Standardabweichung aus den Eintraegen eines gegebenen Vektors
   10:  *
   11:  * \param[in,out]   x   Vektor mit Eintraegen
   12:  * \param[in,out]   a   arithmetisches Mittel
   13:  * \param[in,out]   g   geometrisches Mittel
   14:  * \param[in,out]   h   harmonisches Mittel
   15:  * \param[in,out]   d   Standardabweichung (deviation)
   16:  * \return
   17:  *
   18:  */
   19: void meandevvec(std::vector<int> x, float &a, float &g, float &h, float &d);
   20:
   21:
   22: /**     Calculating the minimal and maximal element of a given vector with openMP (p
arallel)
   23:  * @param[in,out]       x       vector
   24:  * @param[in,out]       minv    minimal entry of x
   25:  * @param[in,out]       maxv    maximal entry of x
   26:  *
   27:  */
   28: void minmaxvec(std::vector<int> &x, int &minv, int &maxv);
   29:
   30: /** Calculating the arithmetic, geometric and harmonic mean value of a given vector
as well as the standard deviation and the minimal and maximal element with execution polici
es
   31:  *
   32:  * \param[in,out]   x   vector
   33:  * \param[in,out]   a   arithmetic mean
   34:  * \param[in,out]   g   geometric mean
   35:  * \param[in,out]   h   harmonic mean
   36:  * \param[in,out]   d   standard deviation
   37:  * @param[in,out]       minv    minimal entry of x
   38:  * @param[in,out]       maxv    maximal entry of x
   39:  *
   40:  */
   41: void meandevmaxminvec(std::vector<int> &x, float &a, float &g, float &h, float &d, f
loat &minv, float &maxv);
   42:
   43: void fill_vector(std::istream& istr, std::vector<int>& v);
   44:
   45: void read_vector_from_file(const std::string& file_name, std::vector<int>& v);
   46:
   47: void write_vector_to_file(const std::string& file_name, const std::vector<float>& v)
;
   48:
   49: #endif // BSP_5_2_LIB_H_INCLUDED
```

```cpp
#include "bsp_5_2_lib.h"
#include <iostream>
#include <algorithm>
using namespace std;
// BSP 5_2

int main()
{
    const string name1("data_1.txt");              // name of input file
    const string name2("out.txt");                // name of output file
    vector<int> v;

    read_vector_from_file(name1, v);

    float a, g, h, d, aex, gex, hex, dex, minvex, maxvex, control1(0);
    int minvp, maxvp, control2(0);

    int NLOOPS = 100000;

    // testing parallel
    double tstart1 = omp_get_wtime();
    for(int i=0; i<NLOOPS; ++i)
    {
                meandevvec(v,a,g,h,d);
                minmaxvec(v,minvp,maxvp);
                control1 += a;
                control2 += minvp;
        }
        double t1all = omp_get_wtime() - tstart1;
        double t1 = t1all/NLOOPS;

    // testing execution policies
    double tstart2 = omp_get_wtime();
    for(int i=0; i<NLOOPS; ++i)
    {
                meandevmaxminvec(v,aex,gex,hex,dex,minvex,maxvex);
                control1 += aex;
                control2 += minvex;
        }
        double t2all = omp_get_wtime() - tstart2;
        double t2 = t2all/NLOOPS;

    cout << "\nTiming for parallel\n";
    cout << "Timing NLOOPS:                    " << t1all << endl;
    cout << "Timing per it. (sec):      " << t1 << endl;

    cout << "\n\nTiming for execution policies\n";
    cout << "Timing NLOOPS:                    " << t2all << endl;
    cout << "Timing per it. (sec):      " << t2 << endl;


    cout << "\n\nAuswertung (parallel)" << endl;
    cout << "arithmetisches Mittel:     " << a << endl;
    cout << "geometrisches Mittel:      " << g << endl;
    cout << "harmonisches Mittel:       " << h << endl;
    cout << "standard deviation:        " << d << endl;
    cout << "Minimum:    " << minvp << endl;
    cout << "Maximum:    " << maxvp << endl;

    cout << endl;
    cout << "\nAuswertung (execution policies)" << endl;
    cout << "arithmetisches Mittel:     " << aex << endl;
    cout << "geometrisches Mittel:      " << gex << endl;
    cout << "harmonisches Mittel:       " << hex << endl;
    cout << "standard deviation:        " << dex << endl;
    cout << "Minimum:    " << minvex << endl;
    cout << "Maximum:    " << maxvex << endl;

```

```
   69:     vector<float> ve{a,g,h,d,minvp,maxvp};
   70:     write_vector_to_file(name2, ve);
   71:
   72:     return 0;
   73: }
```