```
/* ==== File Config.h ==== */
#ifndef ICING_CONFIG_H
#define ICING_CONFIG_H

#include <string>

struct Config{
        unsigned int n1=500, n2=500, n3=5, L=30, threadCount=3;
        int J = 1;
        double T=0.0;
    std::string alg="metro", initial="random";
        bool recordMain = false;
};

struct LabConfig{
        double TMin = 0.1, TMax = 4.53;
        bool normalDist = false;
        unsigned int TSteps = 9;
        std::vector<std::string> output_filenames = std::vector<std::string>(6);
};

#endif //ICING_CONFIG_H
/* ==== File helpers.cpp ==== */
#include <boost/math/special_functions/erf.hpp>
#include <boost/math/distributions/normal.hpp>

std::vector<double> linspace(double x_min, double x_max, unsigned int N){
    std::vector<double> T_vec;
    double dT = (x_max - x_min)/(N -1);
    if(N==1)
        dT=0;
    for(int i=0; i< N; ++i)
        T_vec.push_back(x_min + i*dT);
    return T_vec;
}

std::vector<double> normalSpace(double x_min, double x_max, unsigned int N, double mu, double sigma){
    if(N==1)
        return std::vector<double> {x_min};
    boost::math::normal normal_dist(mu, sigma);
    double area_min = boost::math::cdf(normal_dist, x_min);
    double area_max = boost::math::cdf(normal_dist, x_max);
    double A = area_max - area_min;
    double A0 = A/(N-1.0);
    double area;
    std::vector<double> x;
    for(int i=0; i<N; ++i) {
        area = i*A0 + area_min;
        x.push_back(sqrt(2.0)*sigma* boost::math::erf_inv(area*2.0 - 1.0) + mu);
    }
    return x;
}

std::string to_string(double const & value){
    std::stringstream ss;
    ss << value;
    return ss.str();
}
/* ==== File helpers.h ==== */
#ifndef ICING_HELPERS_H
#define ICING_HELPERS_H

std::vector<double> linspace(double x_min, double x_max, unsigned int N);
std::vector<double> normalSpace(double x_min, double x_max, unsigned int N, double mu, double sigma);
std::string to_string(double const & value);

#endif
/* ==== File magneto.cpp ==== */
#include <iostream>
#include <chrono>
#include <vector>
#include <random>
#include <fstream>
#include <sstream>
#include <omp.h>
#include "physics.h"
#include "System.h"
#include "helpers.h"

void checkParam(int argc, char* argv[], Config& cfg, LabConfig& labCfg){
    std::string key, value;
    int eqPos;
    std::string argument;

    for (int i = 1; i < argc; ++i) {
        argument = argv[i];
        eqPos = argument.find("=");
        key = argument.substr(1, eqPos-1);
        value = argument.substr(eqPos+1);

        if (key == "L")
            cfg.L = atoi(value.c_str());
        else if (key == "N1")
            cfg.n1 = atoi(value.c_str());
        else if (key == "N2")
            cfg.n2 = atoi(value.c_str());
        else if (key == "N3")
            cfg.n3 = atoi(value.c_str());
        else if (key == "threads")
            cfg.threadCount = atoi(value.c_str());
        else if (key == "J")
```

```
            cfg.J = atoi(value.c_str());
        else if (key == "initial")
            cfg.initial = value;
        else if (key == "alg")
            cfg.alg = value;
        else if (key == "record")
            cfg.recordMain = value == "main";
        else if (key == "dist")
            labCfg.normalDist = value == "normal";
        else if (key == "en")
            labCfg.output_filenames[energy] = value;
        else if (key == "mag")
            labCfg.output_filenames[mag] = value;
        else if (key == "cv")
            labCfg.output_filenames[cv] = value;
        else if (key == "chi")
            labCfg.output_filenames[chi] = value;
        else if (key == "corr")
            labCfg.output_filenames[corrfun] = value;
        else if (key == "states")
            labCfg.output_filenames[states] = value;
        else if (key == "TSteps")
            labCfg.TSteps = atoi(value.c_str());
        else if (key == "TMin")
            labCfg.TMin = atof(value.c_str());
        else if (key == "TMax")
            labCfg.TMax = atof(value.c_str());
        else
            std::cerr << "Unknown parameter: " << argument << std::endl;
    }
}

int main(int argc, char* argv[]){
    if (argc < 2) {
        std::cerr << "Too few arguments. See documentation." << std::endl;
        return 1;
    }

    // use cmd arguments
    Config cfg;
    LabConfig labCfg;
    checkParam(argc, argv, cfg, labCfg);
    omp_set_num_threads(cfg.threadCount);

    // setup temperatures and systems
    std::vector<double> temps;
    if(labCfg.normalDist)
        temps = normalSpace(labCfg.TMin, labCfg.TMax, labCfg.TSteps, 2.269, 1.0);
    else
            temps = linspace(labCfg.TMin, labCfg.TMax, labCfg.TSteps);
    std::vector<System> systems;
    auto t0 = std::chrono::high_resolution_clock::now();
    for(double T : temps){
                cfg.T = T;
        systems.push_back(System(cfg, labCfg));
    }

    // do the computations
    std::string progressStr = std::string(labCfg.TSteps, '.');
    std::cout << progressStr.c_str() << std::endl;

    #pragma omp parallel for
    for(unsigned int i=0; i<systems.size(); ++i){
        systems[i].compute();
        std::cout << ".";
    }
    std::cout << std::endl;

    // save results
    std::ofstream fileOut;
    for(unsigned int i=0; i<labCfg.output_filenames.size(); ++i){
        if(! labCfg.output_filenames[i].empty()){
            if(i != states){
                fileOut.open(labCfg.output_filenames[i]+".txt");
                for (auto& sys : systems)
                    fileOut << to_string(sys.cfg.T) << sys.results[i] << std::endl;
                fileOut.close();
            }
            else{
                for(int j=0; j<systems.size(); ++j){
                    fileOut.open(labCfg.output_filenames[i]+to_string(j)+".txt");
                    fileOut << systems[j].results[states] << std::endl;
                    fileOut.close();
                }
            }
        }
    }

    // print runtime
    auto t1 = std::chrono::high_resolution_clock::now();
    double secs = (std::chrono::duration_cast <std::chrono::milliseconds > (t1-t0).count())*0.001;
    std::cout << "runtime: " << secs << "s" << std::endl;

        return 0;
}
/* ==== File physics.cpp ==== */
#include "physics.h"

double calc_E(std::vector<std::vector<int> > &grid) {
    unsigned int L = grid.size();
    double E = 0.0;
```

```
    for(int i = 0; i < L; ++i){
        for (int j = 0; j < L; ++j)
            E += -grid[i][j] * ( grid[i][(j+1)%L] + grid[(i+1)%L][j] );
    }
    return E/(L*L);
}


int calc_dE(std::vector<std::vector<int> >& grid, int idx1, int idx2, const int L) {
    return 2 * grid[idx1][idx2] * (
        grid[idx1][(idx2 + 1) % L] +
        grid[(idx1 + 1) % L][idx2] +
        grid[idx1][(idx2 - 1 + L) % L] +
        grid[(idx1 - 1 + L) % L][idx2]);
}


double calc_m_abs(std::vector<std::vector<int> >& grid) {
    unsigned int L = grid.size();
    int m = 0;
    for (int i = 0; i < L; ++i){
        for (int j = 0; j < L; ++j)
            m += grid[i][j];
    }
    return abs(m)*1.0f/(L*L);
}

/* ==== File physics.h ==== */

#ifndef _IPYNB_PHYSICS_H_
#define _IPYNB_PHYSICS_H_

#include <iostream>
#include <vector>
#include <array>
#include <random>
#include <chrono>
#include <fstream>
#include <functional>
#include <sstream>
#include <deque>
#include <iomanip>

double calc_E(    std::vector<std::vector<int> >& grid);
int    calc_dE(   std::vector<std::vector<int> >& grid, int idx1, int idx2, const int L);
double calc_m_abs(std::vector<std::vector<int> >& grid);

#endif //_IPYNB_PHYSICS_H_
/* ==== File System.cpp ==== */
#include <sstream>
#include "System.h"
#include "physics.h"
#include <boost/algorithm/string.hpp>

System::System(Config& p_cfg, LabConfig& labCfg) :
        cfg(p_cfg){

    // precalc exp values
    int buffer_offset = 8*abs(cfg.J);
    for(int dE=0; dE < buffer_offset*2+1; ++dE)
        exp_values.push_back(exp(-(dE-buffer_offset)/cfg.T));

    results = std::vector<std::string>(6, "");
    corr_count = 500;
    corr_range = cfg.L/2;
    long long int seed1 = std::chrono::_V2::system_clock::now().time_since_epoch().count();
    rng = std::mt19937(seed1);
    std::uniform_int_distribution<unsigned int> dist_grid(0, cfg.L-1);

    for(int i=0; i< corr_count; ++i){
        correlations.push_back(CorrelationPoint());
        correlations.back().i = dist_grid(rng);
        correlations.back().j = dist_grid(rng);
        correlations.back().corr_a.assign(corr_range, 0.0);
        correlations.back().corr_ab.assign(corr_range, 0.0);
    }

    if(!labCfg.output_filenames[energy].empty())
        calc_e = true;
    if(!labCfg.output_filenames[mag].empty())
        calc_m = true;
    if(!labCfg.output_filenames[cv].empty())
        calc_cv = true;
    if(!labCfg.output_filenames[chi].empty())
        calc_chi = true;
    if(!labCfg.output_filenames[corrfun].empty())
        calc_corrfun = true;
    if(!labCfg.output_filenames[states].empty())
        calc_states = true;

    grid = getRelaxedSys(0);
}

grid_type System::genRandomSystem(int seedOffset){
    unsigned seed1 = std::chrono::system_clock::now().time_since_epoch().count();
    std::default_random_engine generator(seed1 + seedOffset);
    std::uniform_int_distribution <int> dist(0,1);
    grid_type grid(cfg.L, std::vector<int>(cfg.L));
    for (int i = 0; i < cfg.L; ++i){
        for (int j = 0; j < cfg.L; ++j){
            grid[i][j] = dist(generator)*2 - 1;
```

```
        }
    }
    return grid;
}

grid_type System::getFileState(std::string filename){
    std::ifstream fileIn(filename);
    std::string line;
    std::vector<std::string> strs;
    grid_type grid(cfg.L, std::vector<int>(cfg.L));
    for (int i = 0; i < cfg.L; ++i){
        std::getline(fileIn, line);
        boost::split(strs, line, boost::is_any_of(","));
        for (int j = 0; j < cfg.L; ++j){
           grid[i][j] = strs[j]=="1"?1:-1;
        }
    }
    return grid;
}

grid_type System::getRelaxedSys(int seedOffset) {
    grid_type new_grid;
    if(cfg.initial == "random")
        new_grid = genRandomSystem(seedOffset);
    else
        new_grid = getFileState(cfg.initial);
    grid = new_grid;

    if(cfg.alg == "metro")
        grid = metropolis_sweeps(grid, cfg.n1);
    else
        std::cerr << "unknown alg!" << std::endl;
    return grid;
}

grid_type& System::metropolis_sweeps(grid_type& lattice, unsigned int n) {
    std::uniform_int_distribution <int> dist_grid(0, cfg.L-1);
    std::uniform_real_distribution <double > dist_one(0.0, 1.0);

    int buffer_offset = 8*abs(cfg.J);
    int flipIdx1, flipIdx2;
    int dE;
    for (int i=0; i < cfg.L*cfg.L*n; ++i){
        flipIdx1 = dist_grid(rng);
        flipIdx2 = dist_grid(rng);
        dE = cfg.J * calc_dE(lattice, flipIdx1, flipIdx2, cfg.L);
        if (dE <= 0 || (dist_one(rng) < exp_values[dE+buffer_offset]) )
            lattice[flipIdx1][flipIdx2] *= -1;
    }
    return lattice;
}

void System::compute() {
    if (calc_states)
        results[states] += to_string(cfg.T) + "\n";

    for(int evolveStep=0; evolveStep<cfg.n2; ++evolveStep) {
        measure();

        if(cfg.recordMain)
            recordResults();

        //  evolve
        if (cfg.alg == "metro")
            grid = metropolis_sweeps(grid, cfg.n3);
        else
            std::cerr << "unknown alg!" << std::endl;
    }
    recordResults();
}

void System::recordResults() {
    double tempResult;
    if (calc_e)
        results[energy] += ", " + to_string(e_avg / cfg.n2);

    if (calc_m)
        results[mag] += ", " + to_string(m_avg / cfg.n2);

    if (calc_cv) {
        tempResult = 1.0*(e2_avg / cfg.n2 - e_avg / cfg.n2* e_avg / cfg.n2) * cfg.L* cfg.L / (cfg.T* cfg.T);
        results[cv] += ", " + to_string(tempResult);
    }

    if (calc_chi) {
        tempResult = 1.0*(m2_avg / cfg.n2 - m_avg / cfg.n2* m_avg / cfg.n2) * cfg.L* cfg.L / cfg.T;
        results[chi] += ", " + to_string(tempResult);
    }

    if (calc_corrfun) {
        const unsigned int d_limit = cfg.L/2;
        std::vector<double> sigma_ij = std::vector<double>(d_limit, 0.0);
        std::vector<double> sigma_i = std::vector<double>(d_limit, 0.0);
        std::vector<double> sigma_j = std::vector<double>(d_limit, 0.0);
        std::vector<double> G = std::vector<double>(d_limit, 0.0);

        for(auto &corr : correlations){
            for (int d = 0; d < corr_range; d++) {
                sigma_ij[d] += corr.corr_ab[d];
                sigma_i[d]  += corr.corr_a[0];
                sigma_j[d]  += corr.corr_a[d];
```

```
            }
        }

        for (int d = 0; d < d_limit; d++) {
            G[d] = sigma_ij[d]/(cfg.n2* corr_count) - sigma_i[d]/(cfg.n2* corr_count)*sigma_j[d]/(cfg.n2*corr_count);
            results[corrfun] += ", " + to_string(G[d]);
        }
    }

    if (calc_states) {
        std::string str="";
        unsigned int L = grid.size();
        for (int i = 0; i < L; ++i) {
            for (int j = 0; j < L; ++j) {
                str += ((grid[i][j] == 1) ? "1" : "0");
                if (j < L - 1)
                    str += ",";
            }
            str += "\n";
        }
        results[states] += str+"\n";
    }
}

void System::measure() {
    if (calc_e || calc_cv){
        double en = calc_E(grid);
        e_avg += en;
        if(calc_cv)
            e2_avg += en*en;
    }
    if (calc_m || calc_chi){
        double mag = calc_m_abs(grid);
        m_avg += mag;
        if(calc_chi)
            m2_avg += mag*mag;
    }
    if (calc_corrfun){
        for(auto &corr : correlations){
            for (int d = 0; d < corr_range; d++) {
                corr.corr_a[d]  += grid[corr.i][(corr.j+d)%cfg.L];
                corr.corr_ab[d] += grid[corr.i][corr.j] * grid[corr.i][(corr.j+d)%cfg.L];
            }
        }
    }
}
/* ==== File System.h ==== */
#include <vector>
#include <random>
#include "Config.h"
#include "helpers.h"

enum resultTypes { energy, mag, cv, chi, states, corrfun };

struct CorrelationPoint {
        unsigned int i, j;
        std::vector<double> corr_ab, corr_a;
};

typedef std::vector<std::vector<int> > grid_type;

class System{
public:
        System(Config&, LabConfig&);
        void compute();
        const Config cfg;
        std::vector<std::string> results;
private:
        void measure();
        grid_type genRandomSystem(int seedOffset);
        grid_type getRelaxedSys(int seedOffsets);
        grid_type getFileState(std::string filename);
        grid_type& metropolis_sweeps(grid_type&, unsigned int);
        void recordResults();

        std::mt19937 rng;
        std::vector<double> exp_values;
        bool calc_e=false, calc_m=false, calc_cv=false, calc_chi=false, calc_states=false, calc_corrfun=false;
        grid_type grid;
        double e_avg=0.0, e2_avg=0.0, m_avg=0.0, m2_avg=0.0;
        std::vector<CorrelationPoint> correlations;
        unsigned int corr_range, corr_count;
        std::vector<int> access_i;
        std::vector<int> access_j;
};
```