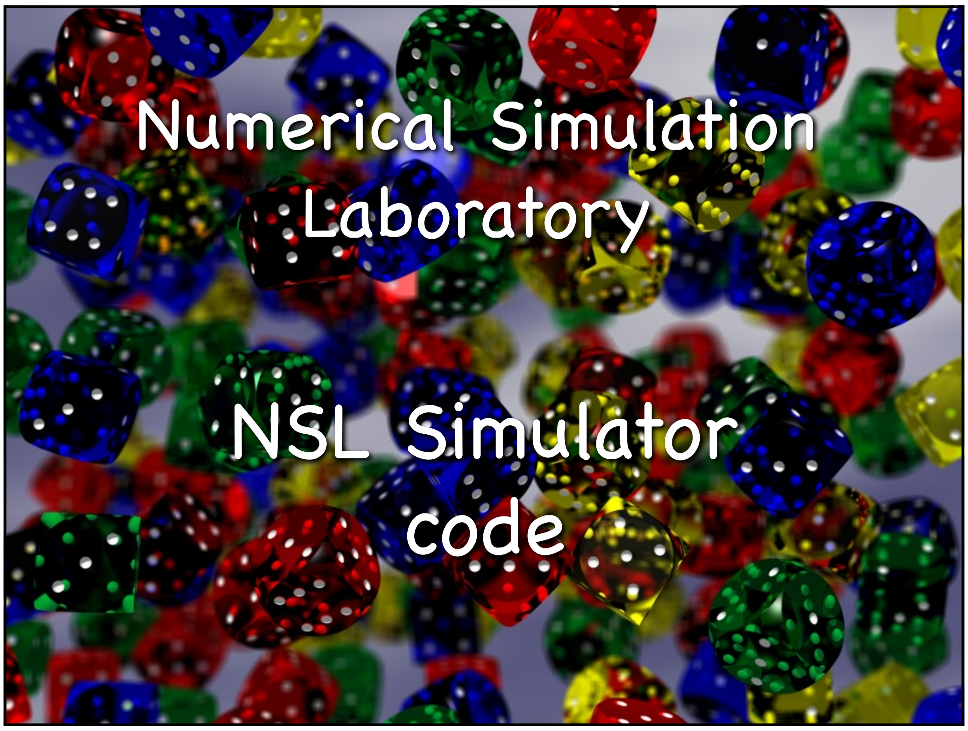
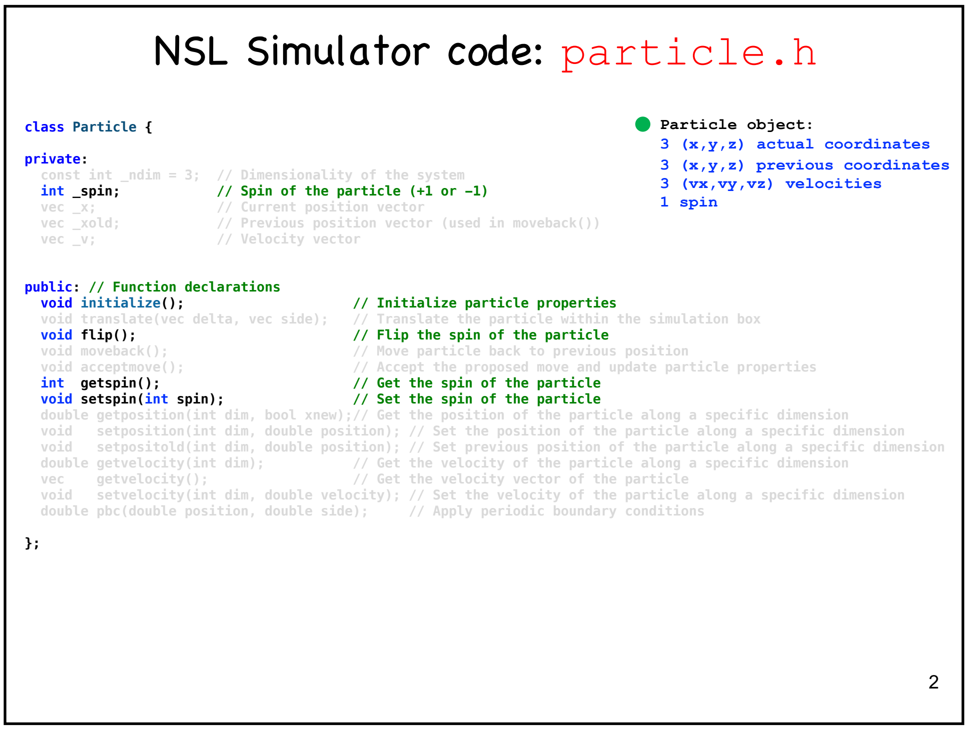


Numerical Simulation Laboratory

NSL Simulator code

1



NSL Simulator code: particle.h

```
class Particle {

private:
  const int _ndim = 3;  // Dimensionality of the system
  int _spin;            // Spin of the particle (+1 or -1)
  vec _x;               // Current position vector
  vec _xold;            // Previous position vector (used in moveback())
  vec _v;               // Velocity vector


public: // Function declarations
  void initialize();                // Initialize particle properties
  void translate(vec delta, vec side);   // Translate the particle within the simulation box
  void flip();                      // Flip the spin of the particle
  void moveback();                  // Move particle back to previous position
  void acceptmove();                // Accept the proposed move and update particle properties
  int  getspin();                   // Get the spin of the particle
  void setspin(int spin);           // Set the spin of the particle
  double getposition(int dim, bool xnew);// Get the position of the particle along a specific dimension
  void   setposition(int dim, double position); // Set the position of the particle along a specific dimension
  void   setpositold(int dim, double position); // Set previous position of the particle along a specific dimension
  double getvelocity(int dim);           // Get the velocity of the particle along a specific dimension
  vec    getvelocity();                  // Get the velocity vector of the particle
  void   setvelocity(int dim, double velocity); // Set the velocity of the particle along a specific dimension
  double pbc(double position, double side);     // Apply periodic boundary conditions

};
```

● Particle object:
3 (x,y,z) actual coordinates
3 (x,y,z) previous coordinates
3 (vx,vy,vz) velocities
1 spin

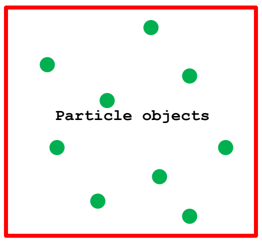
2

2

NSL Simulator code: system.h 1.

```
class System {

private:
  const int _ndim = 3;  // Dimensionality of the system
  bool _restart;        // Flag indicating if the simulation is restarted
  int _sim_type;        // Type of simulation (e.g., Lennard-Jones, Ising)
  int _npart;           // Number of particles
  int _nblocks;         // Number of blocks for block averaging
  int _nsteps;          // Number of simulation steps in each block
  int _nattempts;       // Number of attempted moves
  int _naccepted;       // Number of accepted moves
  double _temp, _beta;  // Temperature and inverse temperature
  double _rho, _volume; // Density and volume of the system
  double _r_cut;        // Cutoff radius for pair interactions
  double _delta;        // Displacement step for particle moves
  double _J, _H;        // Parameters for the Ising Hamiltonian
  vec _side;            // Box dimensions
  vec _halfside;        // Half of box dimensions
  Random _rnd;          // Random number generator
  field <Particle> _particle; // Field of particle objects representing the system
  vec _fx, _fy, _fz;    // Forces on particles along x, y, and z directions

  // Properties
  int _nprop; // Number of properties being measured
  bool _measure_penergy, _measure_kenergy, _measure_tenergy;// Flags for measuring different energies
  bool _measure_temp, _measure_pressure, _measure_gofr;     // Flags for measuring temp, pressure, radial dist. function
  bool _measure_magnet, _measure_cv, _measure_chi;          // Flags for measuring magnetization, specific heat, susceptibility
  int _index_penergy, _index_kenergy, _index_tenergy;       // Indices for accessing energy properties in vec _measurement
  int _index_temp, _index_pressure, _index_gofr;            // Indices for accessing temp, pressure, and radial dist. function
  int _index_magnet, _index_cv, _index_chi;                 // Indices for accessing magnetization, specific heat, susceptibility
  int _n_bins;           // Number of bins for radial distribution function
  double _bin_size;      // Size of bins for radial distribution function
  double _vtail, _ptail; // Tail corrections for energy and pressure
  vec _block_av;         // Block averages of properties
  vec _global_av;        // Global averages of properties
  vec _global_av2;       // Squared global averages of properties
  vec _average;          // Average values of properties
  vec _measurement;      // Measured values of properties
```

System object:
A field i.e. a vector of <Particles>
in a box with p.b.c.



3

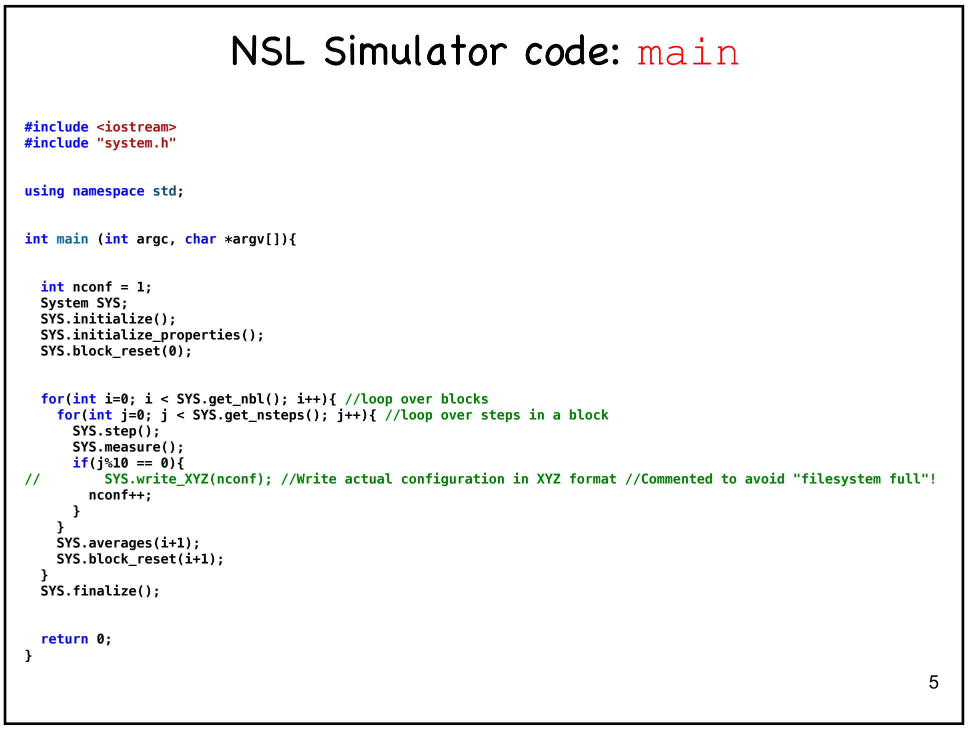
3

NSL Simulator code: system.h 2.

```
public: // Function declarations

  int get_nbl();              // Get the number of blocks
  int get_nsteps();           // Get the number of steps in each block
  void initialize();          // Initialize system properties
  void initialize_properties();// Initialize properties for measurement
  void finalize();            // Finalize system and clean up
  void write_configuration(); // Write final system configuration to XYZ file
  void write_XYZ(int nconf);  // Write system configuration in XYZ format on the fly
  void write_velocities();    // Write final particle velocities to file
  void read_configuration();  // Read system configuration from file
  void initialize_velocities();// Initialize particle velocities
  void step();                // Perform a simulation step
  void block_reset(int blk);  // Reset block averages
  void measure();             // Measure properties of the system
  void averages(int blk);     // Compute averages of properties
  double error(double acc, double acc2, int blk); // Compute error
  void move(int part);        // Move a particle
  bool metro(int part);       // Perform Metropolis acceptance-rejection step
  double pbc(double position, int i); // Apply periodic boundary conditions for coordinates
  int pbc(int i);             // Apply periodic boundary conditions for spins
  void Verlet();              // Perform Verlet integration step
  double Force(int i, int dim); // Calculate force on a particle along a dimension
  double Boltzmann(int i, bool xnew); // Calculate Boltzmann factor for Metropolis acceptance

};
```

4

4

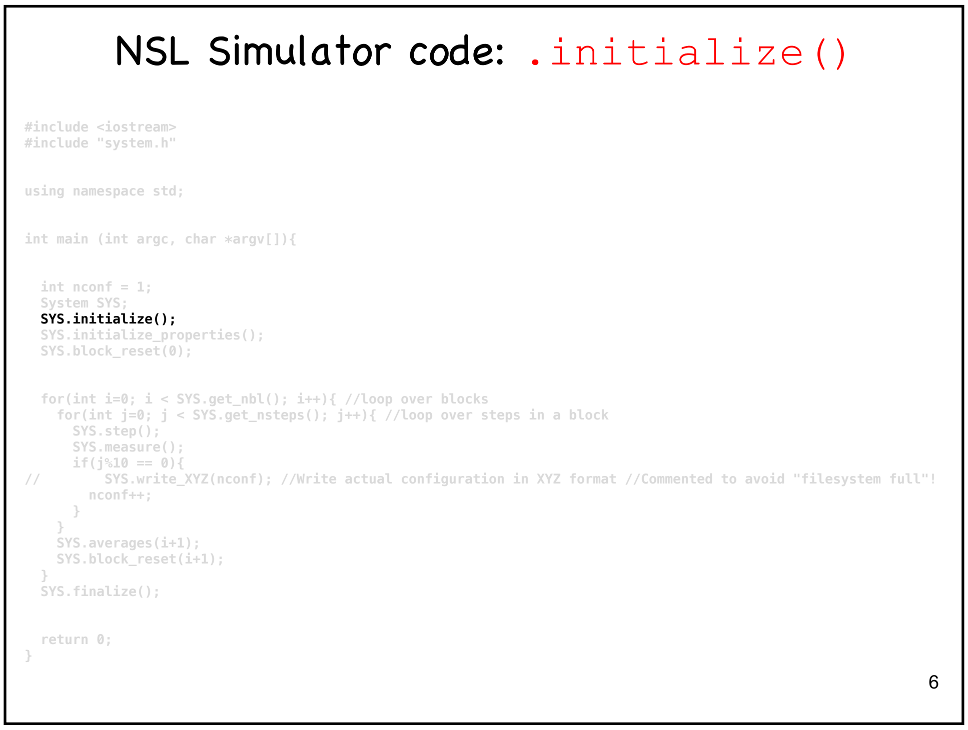


NSL Simulator code: main

```
#include <iostream>
#include "system.h"

using namespace std;

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

  int nconf = 1;
  System SYS;
  SYS.initialize();
  SYS.initialize_properties();
  SYS.block_reset(0);

  for(int i=0; i < SYS.get_nbl(); i++){ //loop over blocks
    for(int j=0; j < SYS.get_nsteps(); j++){ //loop over steps in a block
      SYS.step();
      SYS.measure();
      if(j%10 == 0){
//        SYS.write_XYZ(nconf); //Write actual configuration in XYZ format //Commented to avoid "filesystem full"!
        nconf++;
      }
    }
    SYS.averages(i+1);
    SYS.block_reset(i+1);
  }
  SYS.finalize();

  return 0;
}
```

5

5



NSL Simulator code: .initialize()

```
#include <iostream>
#include "system.h"

using namespace std;

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

  int nconf = 1;
  System SYS;
  SYS.initialize();
  SYS.initialize_properties();
  SYS.block_reset(0);

  for(int i=0; i < SYS.get_nbl(); i++){ //loop over blocks
    for(int j=0; j < SYS.get_nsteps(); j++){ //loop over steps in a block
      SYS.step();
      SYS.measure();
      if(j%10 == 0){
//        SYS.write_XYZ(nconf); //Write actual configuration in XYZ format //Commented to avoid "filesystem full"!
        nconf++;
      }
    }
    SYS.averages(i+1);
    SYS.block_reset(i+1);
  }
  SYS.finalize();

  return 0;
}
```

6

6

System :: initialize()   1.

```
void System :: initialize(){ //Initialize the System object according to the content of the input files

  int p1, p2; // Read from ../INPUT/Primes a pair of numbers to be used to initialize the RNG
  ifstream Primes("../INPUT/Primes");
  Primes >> p1 >> p2 ;
  Primes.close();
  int seed[4]; // Read the seed of the RNG
  ifstream Seed("../INPUT/seed.in");
  Seed >> seed[0] >> seed[1] >> seed[2] >> seed[3];
  _rnd.SetRandom(seed,p1,p2);

  ofstream couta("../OUTPUT/acceptance.dat"); // Set the heading line in file ../OUTPUT/acceptance.dat
  couta << "#   N_BLOCK:  ACCEPTANCE:" << endl;
  couta.close();

  ifstream input("../INPUT/input.dat"); // Start reading ../INPUT/input.dat
  ofstream coutf;
  coutf.open("../OUTPUT/output.dat");
  string property;
  double mass, delta;
  while ( !input.eof() ){
    input >> property;
    if( property == "SIMULATION_TYPE" ){
      input >> _sim_type;
      if(_sim_type > 1){
        input >> _J;
        input >> _H;
      }
      if(_sim_type > 3){
        cerr << "PROBLEM: unknown simulation type" << endl;
        exit(EXIT_FAILURE);
      }
      if(_sim_type == 0)      coutf << "LJ MOLECULAR DYNAMICS (NVE) SIMULATION"  << endl;
      else if(_sim_type == 1) coutf << "LJ MONTE CARLO (NVT) SIMULATION"         << endl;
      else if(_sim_type == 2) coutf << "ISING 1D MONTE CARLO (MRT^2) SIMULATION" << endl;
      else if(_sim_type == 3) coutf << "ISING 1D MONTE CARLO (GIBBS) SIMULATION" << endl;
    } else if( property == "RESTART" ){
      input >> _restart;
```

Input.dat

```
SIMULATION_TYPE        2    1.0    0.0
RESTART                0
TEMP                   1.0
NPART                  50
RHO                    1.0
R_CUT                  0.0
DELTA                  0.0
NBLOCKS                20
NSTEPS                 20000

ENDINPUT
```

… continues in the next slide …

System :: initialize()   2.

```
    } else if( property == "TEMP" ){
      input >> _temp;
      _beta = 1.0/_temp;
      coutf << "TEMPERATURE= " << _temp << endl;
    } else if( property == "NPART" ){
      input >> _npart;
      _fx.resize(_npart);
      _fy.resize(_npart);
      _fz.resize(_npart);
      _particle.set_size(_npart);
      for(int i=0; i<_npart; i++){
        _particle(i).initialize();
        if(_rnd.Rannyu() > 0.5) _particle(i).flip(); // to randomize the spin configuration
      }
      coutf << "NPART= " << _npart << endl;
    } else if( property == "RHO" ){
      input >> _rho;
      _volume = _npart/_rho;
      _side.resize(_ndim);
      _halfside.resize(_ndim);
      double side = pow(_volume, 1.0/3.0);
      for(int i=0; i<_ndim; i++) _side(i) = side;
      _halfside=0.5*_side;
      coutf << "SIDE= ";
      for(int i=0; i<_ndim; i++){
        coutf << setw(12) << _side[i];
      }
      coutf << endl;
    } else if( property == "R_CUT" ){
      input >> _r_cut;
      coutf << "R_CUT= " << _r_cut << endl;
    } else if( property == "DELTA" ){
      input >> delta;
      coutf << "DELTA= " << delta << endl;
      _delta = delta;
```

Particle::initialize()

```
void Particle :: initialize(){
   _spin = 1;
   _x.resize(_ndim);
   _xold.resize(_ndim);
   _v.resize(_ndim);
   return;
}
```

… continues in the next slide …

System :: initialize() 3.

```
   } else if( property == "NBLOCKS" ){
      input >> _nblocks;
      coutf << "NBLOCKS= " << _nblocks << endl;
   } else if( property == "NSTEPS" ){
      input >> _nsteps;
      coutf << "NSTEPS= " << _nsteps << endl;
   } else if( property == "ENDINPUT" ){
      coutf << "Reading input completed!" << endl;
      break;
   } else cerr << "PROBLEM: unknown input" << endl;
 }
 input.close();
 this->read_configuration();
 this->initialize_velocities();
 coutf << "System initialized!" << endl;
 coutf.close();
 return;
}
```

System::read_configuration()

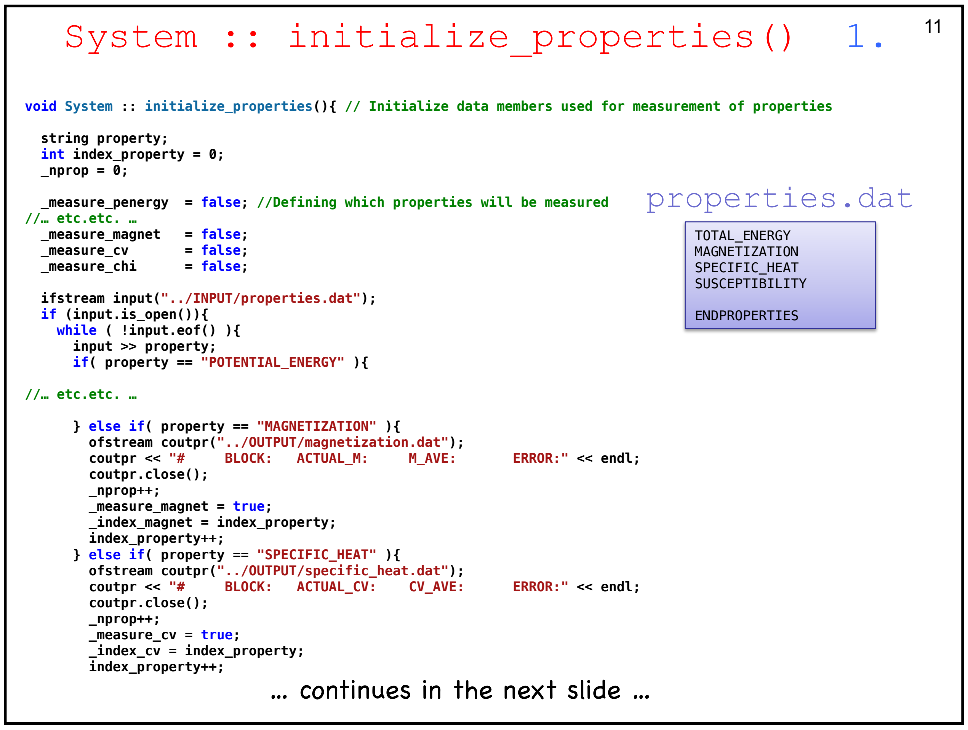
```
void System :: read_configuration(){
  ifstream cinf;
  // PARTICLE CONFIGURATION
  if(_restart and _sim_type > 1){
    int spin;
    cinf.open("../INPUT/CONFIG/config.spin");
    for(int i=0; i<_npart; i++){
      cinf >> spin;
      _particle(i).setspin(spin);
    }
    cinf.close();
  }
  return;
}
```

9

NSL Simulator code: .initialize_properties()

```
#include <iostream>
#include "system.h"

using namespace std;

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

  int nconf = 1;
  System SYS;
  SYS.initialize();
  SYS.initialize_properties();
  SYS.block_reset(0);

  for(int i=0; i < SYS.get_nbl(); i++){ //loop over blocks
    for(int j=0; j < SYS.get_nsteps(); j++){ //loop over steps in a block
      SYS.step();
      SYS.measure();
      if(j%10 == 0){
//        SYS.write_XYZ(nconf); //Write actual configuration in XYZ format //Commented to avoid "filesystem full"!
        nconf++;
      }
    }
    SYS.averages(i+1);
    SYS.block_reset(i+1);
  }
  SYS.finalize();

  return 0;
}
```

10



System :: initialize_properties()   1.

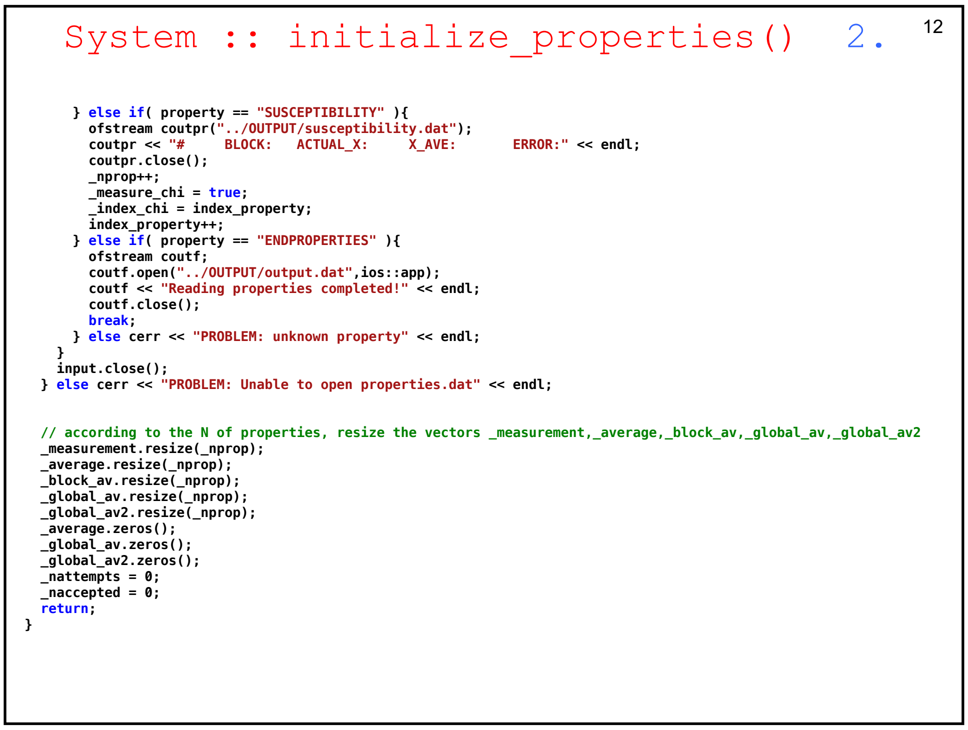
```
void System :: initialize_properties(){ // Initialize data members used for measurement of properties

  string property;
  int index_property = 0;
  _nprop = 0;

  _measure_penergy  = false; //Defining which properties will be measured
//… etc.etc. …
  _measure_magnet   = false;
  _measure_cv       = false;
  _measure_chi      = false;

  ifstream input("../INPUT/properties.dat");
  if (input.is_open()){
    while ( !input.eof() ){
      input >> property;
      if( property == "POTENTIAL_ENERGY" ){

//… etc.etc. …

      } else if( property == "MAGNETIZATION" ){
        ofstream coutpr("../OUTPUT/magnetization.dat");
        coutpr << "#     BLOCK:   ACTUAL_M:     M_AVE:       ERROR:" << endl;
        coutpr.close();
        _nprop++;
        _measure_magnet = true;
        _index_magnet = index_property;
        index_property++;
      } else if( property == "SPECIFIC_HEAT" ){
        ofstream coutpr("../OUTPUT/specific_heat.dat");
        coutpr << "#     BLOCK:   ACTUAL_CV:    CV_AVE:      ERROR:" << endl;
        coutpr.close();
        _nprop++;
        _measure_cv = true;
        _index_cv = index_property;
        index_property++;
```

properties.dat

```
TOTAL_ENERGY
MAGNETIZATION
SPECIFIC_HEAT
SUSCEPTIBILITY

ENDPROPERTIES
```

… continues in the next slide …

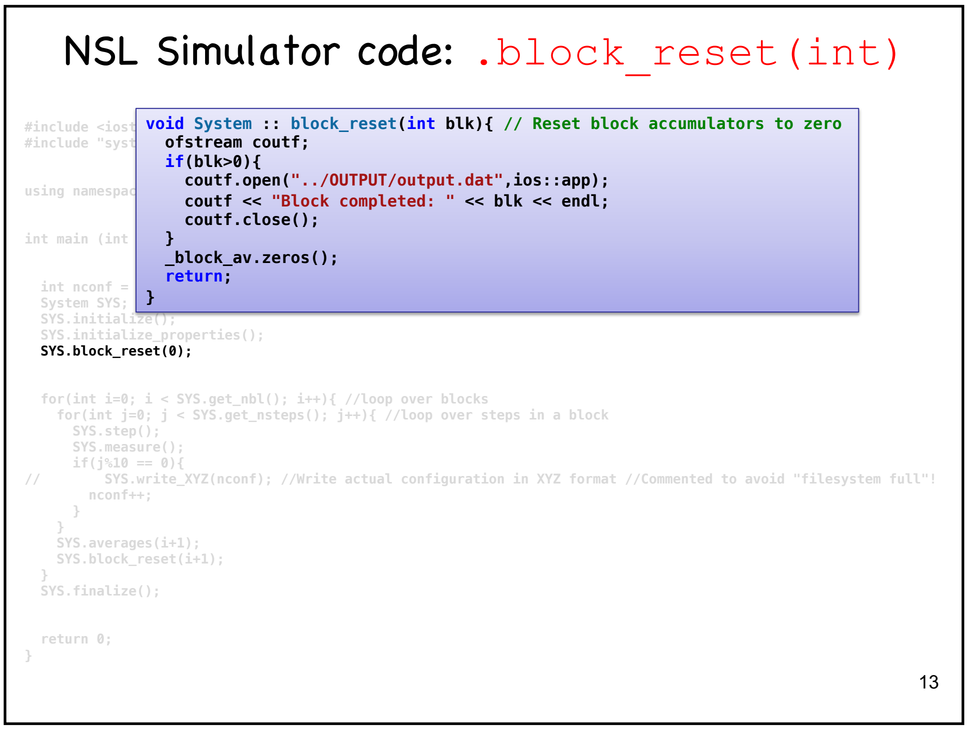
11



System :: initialize_properties()   2.

```
      } else if( property == "SUSCEPTIBILITY" ){
        ofstream coutpr("../OUTPUT/susceptibility.dat");
        coutpr << "#     BLOCK:   ACTUAL_X:     X_AVE:       ERROR:" << endl;
        coutpr.close();
        _nprop++;
        _measure_chi = true;
        _index_chi = index_property;
        index_property++;
      } else if( property == "ENDPROPERTIES" ){
        ofstream coutf;
        coutf.open("../OUTPUT/output.dat",ios::app);
        coutf << "Reading properties completed!" << endl;
        coutf.close();
        break;
      } else cerr << "PROBLEM: unknown property" << endl;
    }
    input.close();
  } else cerr << "PROBLEM: Unable to open properties.dat" << endl;

  // according to the N of properties, resize the vectors _measurement,_average,_block_av,_global_av,_global_av2
  _measurement.resize(_nprop);
  _average.resize(_nprop);
  _block_av.resize(_nprop);
  _global_av.resize(_nprop);
  _global_av2.resize(_nprop);
  _average.zeros();
  _global_av.zeros();
  _global_av2.zeros();
  _nattempts = 0;
  _naccepted = 0;
  return;
}
```

12



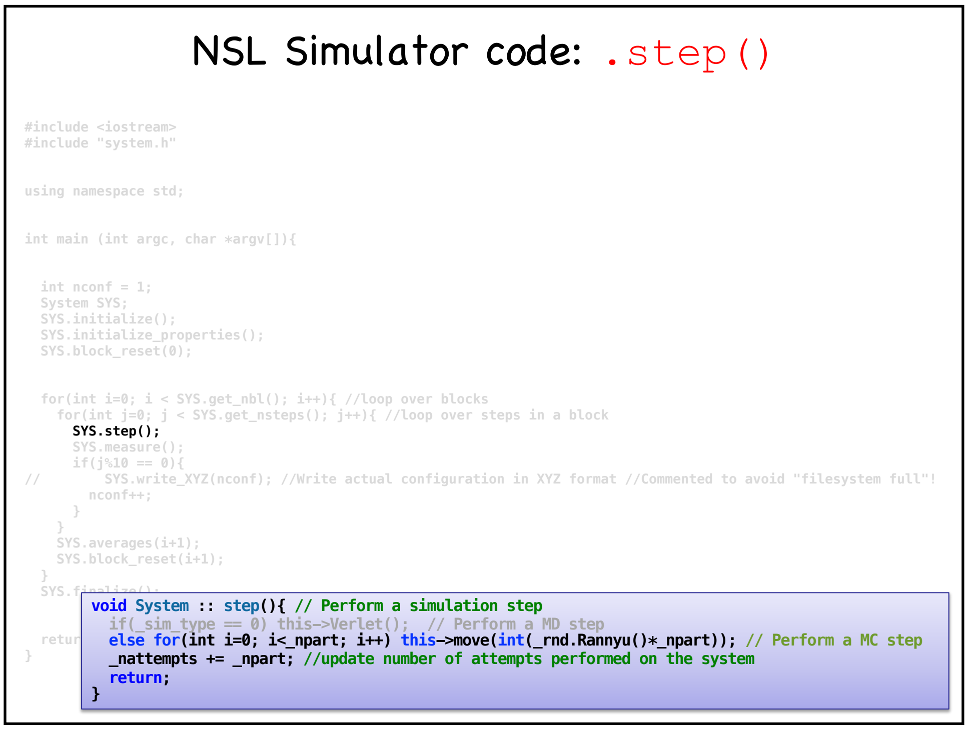
NSL Simulator code: .block_reset(int)

```
void System :: block_reset(int blk){ // Reset block accumulators to zero
  ofstream coutf;
  if(blk>0){
    coutf.open("../OUTPUT/output.dat",ios::app);
    coutf << "Block completed: " << blk << endl;
    coutf.close();
  }
  _block_av.zeros();
  return;
}
```

```
#include <iost
#include "syst

using namespac

int main (int

  int nconf =
  System SYS;
  SYS.initialize();
  SYS.initialize_properties();
  SYS.block_reset(0);

  for(int i=0; i < SYS.get_nbl(); i++){ //loop over blocks
    for(int j=0; j < SYS.get_nsteps(); j++){ //loop over steps in a block
      SYS.step();
      SYS.measure();
      if(j%10 == 0){
//        SYS.write_XYZ(nconf); //Write actual configuration in XYZ format //Commented to avoid "filesystem full"!
        nconf++;
      }
    }
    SYS.averages(i+1);
    SYS.block_reset(i+1);
  }
  SYS.finalize();

  return 0;
}
```

13

13

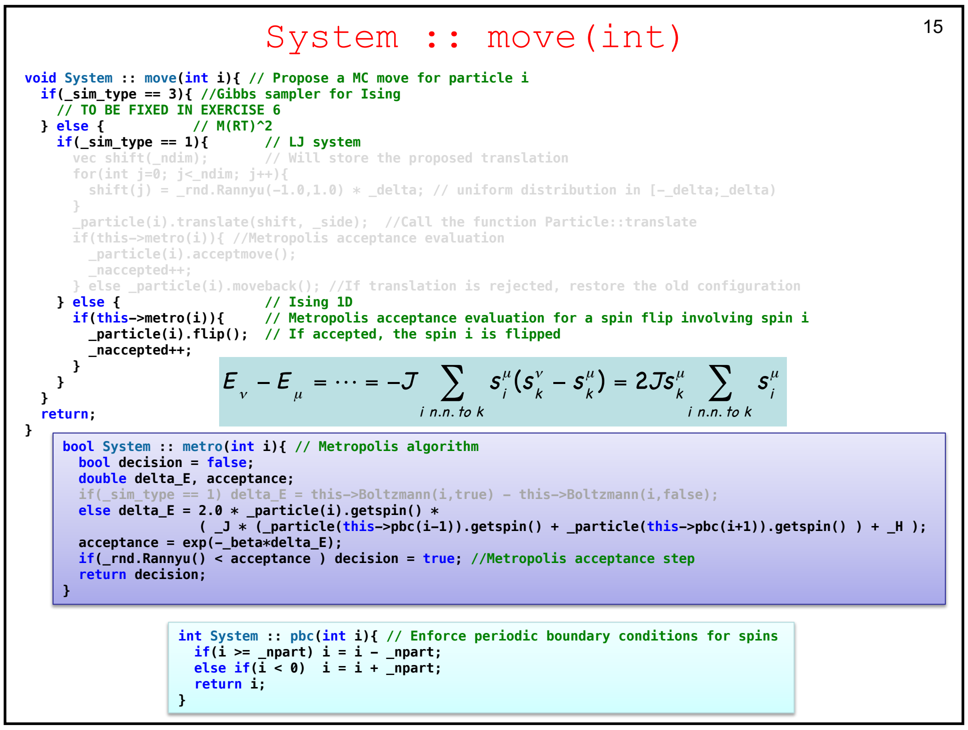


NSL Simulator code: .step()

```
#include <iostream>
#include "system.h"

using namespace std;

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

  int nconf = 1;
  System SYS;
  SYS.initialize();
  SYS.initialize_properties();
  SYS.block_reset(0);

  for(int i=0; i < SYS.get_nbl(); i++){ //loop over blocks
    for(int j=0; j < SYS.get_nsteps(); j++){ //loop over steps in a block
      SYS.step();
      SYS.measure();
      if(j%10 == 0){
//        SYS.write_XYZ(nconf); //Write actual configuration in XYZ format //Commented to avoid "filesystem full"!
        nconf++;
      }
    }
    SYS.averages(i+1);
    SYS.block_reset(i+1);
  }
  SYS.finalize();

  retur
}
```

```
void System :: step(){ // Perform a simulation step
  if(_sim_type == 0) this->Verlet();  // Perform a MD step
  else for(int i=0; i<_npart; i++) this->move(int(_rnd.Rannyu()*_npart)); // Perform a MC step
  _nattempts += _npart; //update number of attempts performed on the system
  return;
}
```

14



System :: move(int)

15

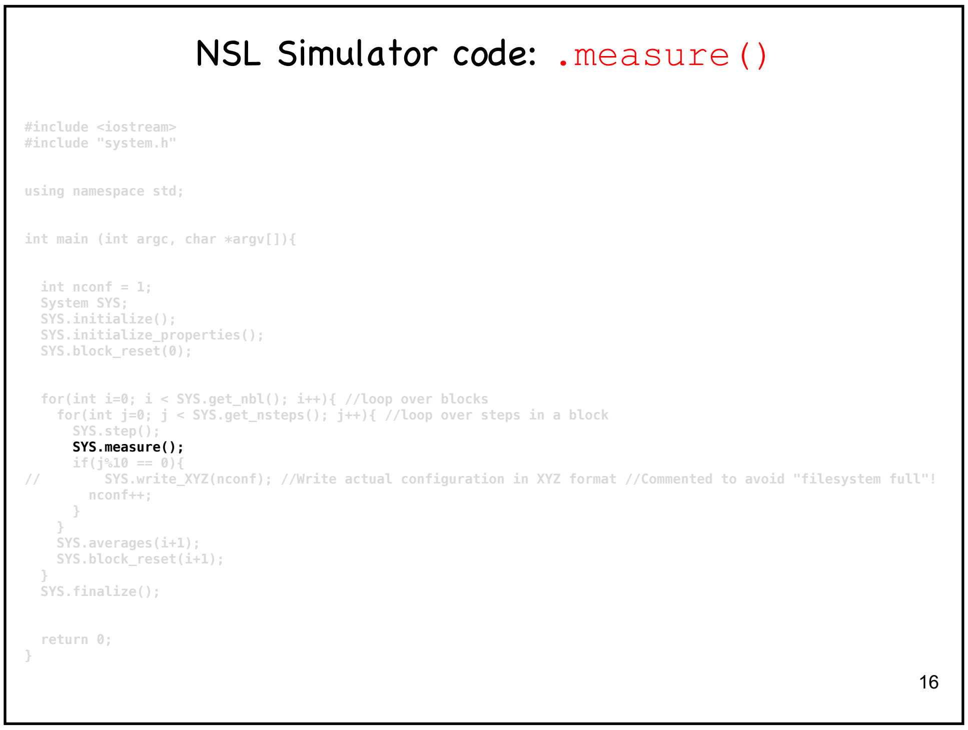
```
void System :: move(int i){ // Propose a MC move for particle i
  if(_sim_type == 3){ //Gibbs sampler for Ising
    // TO BE FIXED IN EXERCISE 6
  } else {           // M(RT)^2
    if(_sim_type == 1){       // LJ system
      vec shift(_ndim);       // Will store the proposed translation
      for(int j=0; j<_ndim; j++){
        shift(j) = _rnd.Rannyu(-1.0,1.0) * _delta; // uniform distribution in [-_delta;_delta)
      }
      _particle(i).translate(shift, _side);  //Call the function Particle::translate
      if(this->metro(i)){ //Metropolis acceptance evaluation
        _particle(i).acceptmove();
        _naccepted++;
      } else _particle(i).moveback(); //If translation is rejected, restore the old configuration
    } else {                  // Ising 1D
      if(this->metro(i)){     // Metropolis acceptance evaluation for a spin flip involving spin i
        _particle(i).flip();  // If accepted, the spin i is flipped
        _naccepted++;
      }
    }
  }
  return;
}
```

$$E_\nu - E_\mu = \cdots = -J \sum_{i \text{ n.n. to } k} s_i^\mu (s_k^\nu - s_k^\mu) = 2Js_k^\mu \sum_{i \text{ n.n. to } k} s_i^\mu$$

```
bool System :: metro(int i){ // Metropolis algorithm
  bool decision = false;
  double delta_E, acceptance;
  if(_sim_type == 1) delta_E = this->Boltzmann(i,true) - this->Boltzmann(i,false);
  else delta_E = 2.0 * _particle(i).getspin() *
                 ( _J * (_particle(this->pbc(i-1)).getspin() + _particle(this->pbc(i+1)).getspin() ) + _H );
  acceptance = exp(-_beta*delta_E);
  if(_rnd.Rannyu() < acceptance ) decision = true; //Metropolis acceptance step
  return decision;
}
```

```
int System :: pbc(int i){ // Enforce periodic boundary conditions for spins
  if(i >= _npart) i = i - _npart;
  else if(i < 0)  i = i + _npart;
  return i;
}
```

15

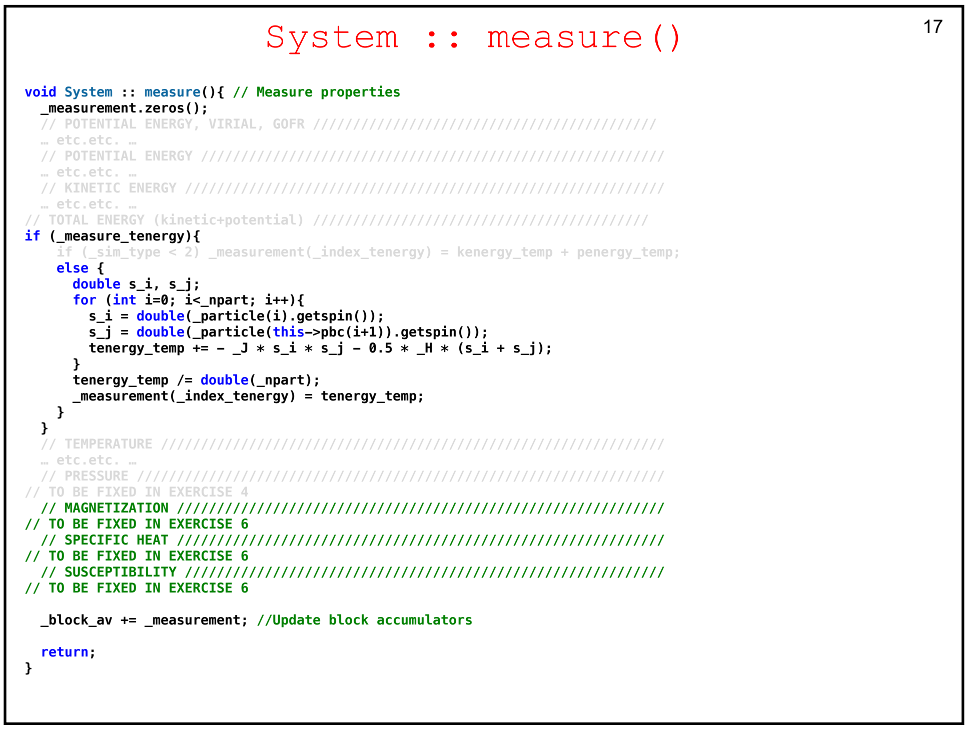


NSL Simulator code: .measure()

```
#include <iostream>
#include "system.h"


using namespace std;


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


  int nconf = 1;
  System SYS;
  SYS.initialize();
  SYS.initialize_properties();
  SYS.block_reset(0);


  for(int i=0; i < SYS.get_nbl(); i++){ //loop over blocks
    for(int j=0; j < SYS.get_nsteps(); j++){ //loop over steps in a block
      SYS.step();
      SYS.measure();
      if(j%10 == 0){
//        SYS.write_XYZ(nconf); //Write actual configuration in XYZ format //Commented to avoid "filesystem full"!
        nconf++;
      }
    }
    SYS.averages(i+1);
    SYS.block_reset(i+1);
  }
  SYS.finalize();


  return 0;
}
```

16

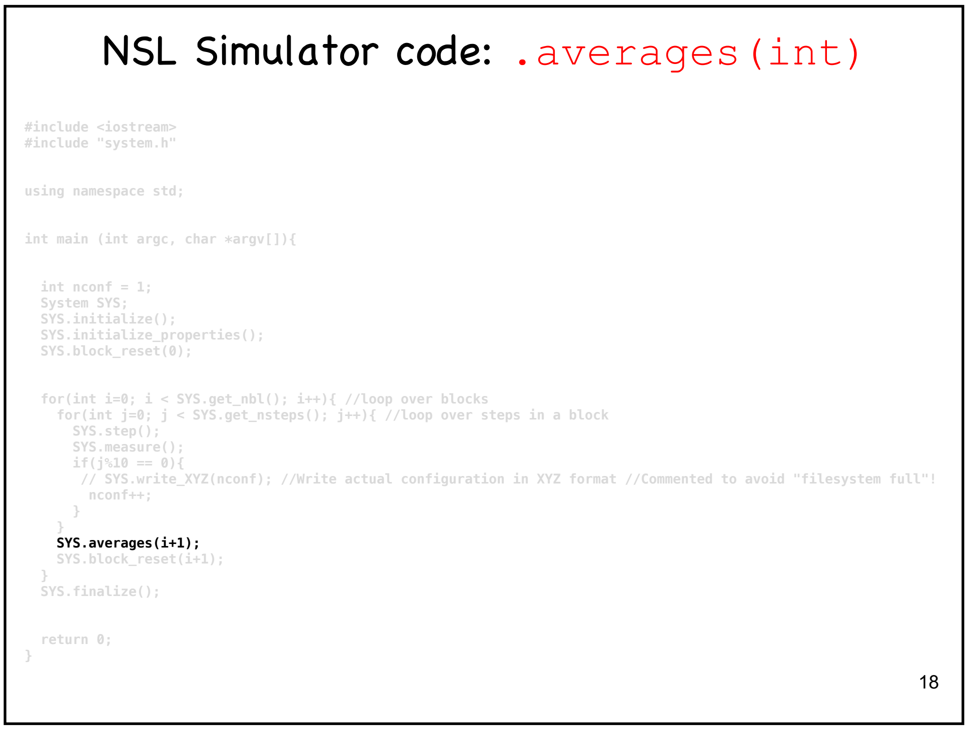
16



System :: measure()

```
void System :: measure(){ // Measure properties
  _measurement.zeros();
  // POTENTIAL ENERGY, VIRIAL, GOFR ////////////////////////////////////////////
  … etc.etc. …
  // POTENTIAL ENERGY //////////////////////////////////////////////////////////
  … etc.etc. …
  // KINETIC ENERGY ////////////////////////////////////////////////////////////
  … etc.etc. …
// TOTAL ENERGY (kinetic+potential) //////////////////////////////////////////
if (_measure_tenergy){
    if (_sim_type < 2) _measurement(_index_tenergy) = kenergy_temp + penergy_temp;
    else {
      double s_i, s_j;
      for (int i=0; i<_npart; i++){
        s_i = double(_particle(i).getspin());
        s_j = double(_particle(this->pbc(i+1)).getspin());
        tenergy_temp += - _J * s_i * s_j - 0.5 * _H * (s_i + s_j);
      }
      tenergy_temp /= double(_npart);
      _measurement(_index_tenergy) = tenergy_temp;
    }
  }
  // TEMPERATURE ///////////////////////////////////////////////////////////////
  … etc.etc. …
  // PRESSURE //////////////////////////////////////////////////////////////////
// TO BE FIXED IN EXERCISE 4
  // MAGNETIZATION /////////////////////////////////////////////////////////////
// TO BE FIXED IN EXERCISE 6
  // SPECIFIC HEAT /////////////////////////////////////////////////////////////
// TO BE FIXED IN EXERCISE 6
  // SUSCEPTIBILITY ////////////////////////////////////////////////////////////
// TO BE FIXED IN EXERCISE 6

  _block_av += _measurement; //Update block accumulators

  return;
}
```

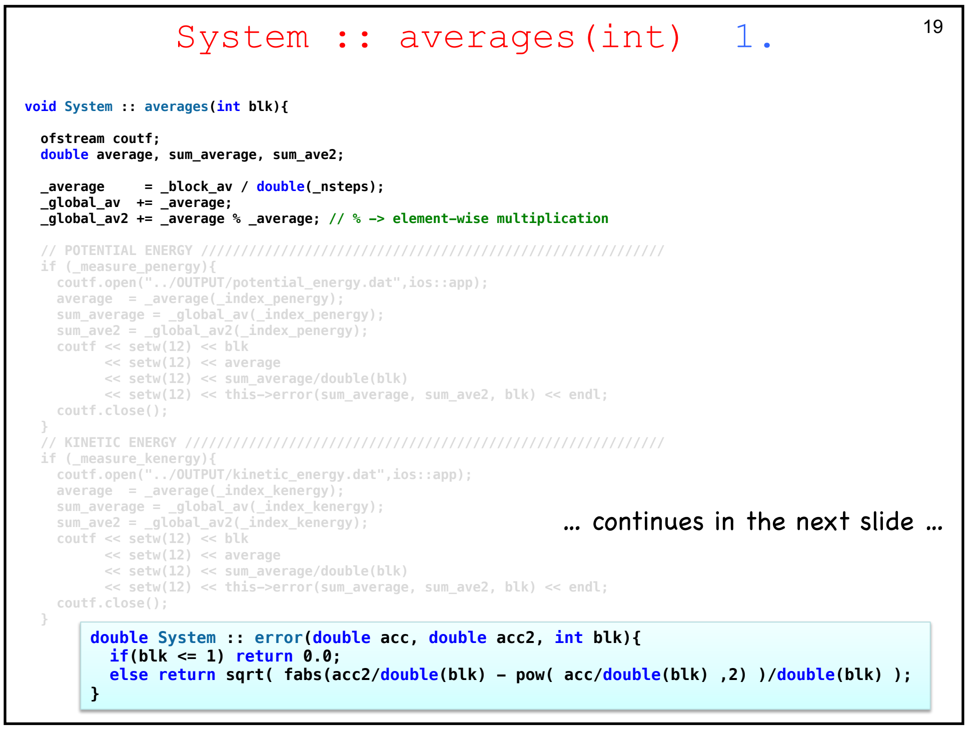
17



NSL Simulator code: .averages(int)

```
#include <iostream>
#include "system.h"

using namespace std;

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

  int nconf = 1;
  System SYS;
  SYS.initialize();
  SYS.initialize_properties();
  SYS.block_reset(0);

  for(int i=0; i < SYS.get_nbl(); i++){ //loop over blocks
    for(int j=0; j < SYS.get_nsteps(); j++){ //loop over steps in a block
      SYS.step();
      SYS.measure();
      if(j%10 == 0){
       // SYS.write_XYZ(nconf); //Write actual configuration in XYZ format //Commented to avoid "filesystem full"!
        nconf++;
      }
    }
    SYS.averages(i+1);
    SYS.block_reset(i+1);
  }
  SYS.finalize();

  return 0;
}
```

18



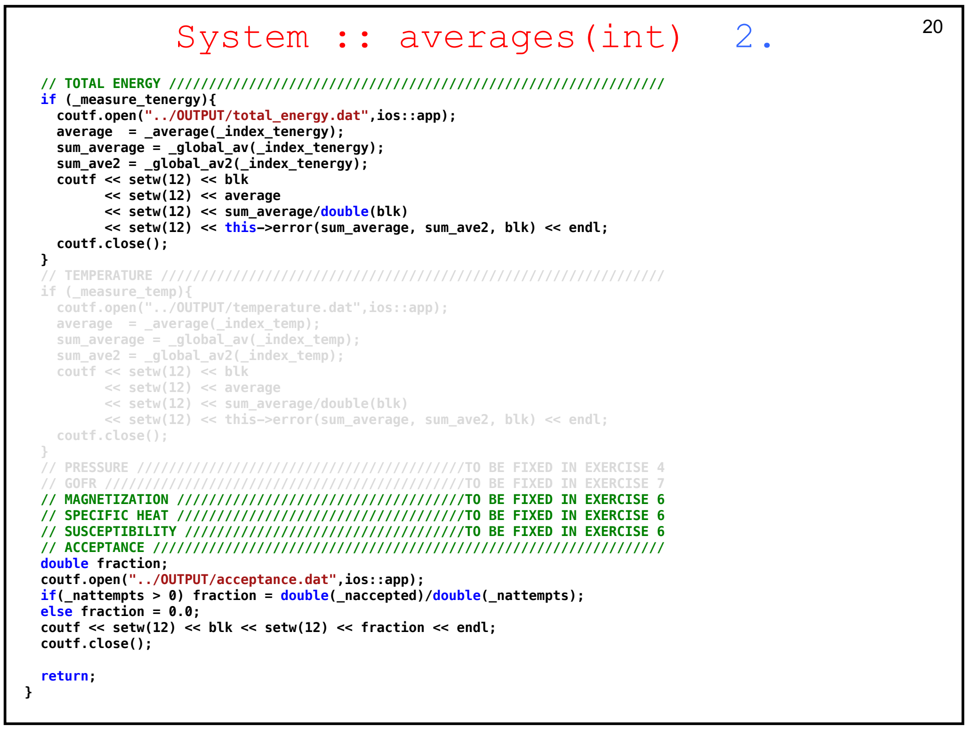
System :: averages(int)   1.

```cpp
void System :: averages(int blk){

  ofstream coutf;
  double average, sum_average, sum_ave2;

  _average     = _block_av / double(_nsteps);
  _global_av  += _average;
  _global_av2 += _average % _average; // % -> element-wise multiplication

  // POTENTIAL ENERGY //////////////////////////////////////////////////////////
  if (_measure_penergy){
    coutf.open("../OUTPUT/potential_energy.dat",ios::app);
    average  = _average(_index_penergy);
    sum_average = _global_av(_index_penergy);
    sum_ave2 = _global_av2(_index_penergy);
    coutf << setw(12) << blk
          << setw(12) << average
          << setw(12) << sum_average/double(blk)
          << setw(12) << this->error(sum_average, sum_ave2, blk) << endl;
    coutf.close();
  }
  // KINETIC ENERGY ////////////////////////////////////////////////////////////
  if (_measure_kenergy){
    coutf.open("../OUTPUT/kinetic_energy.dat",ios::app);
    average  = _average(_index_kenergy);
    sum_average = _global_av(_index_kenergy);
    sum_ave2 = _global_av2(_index_kenergy);
    coutf << setw(12) << blk
          << setw(12) << average
          << setw(12) << sum_average/double(blk)
          << setw(12) << this->error(sum_average, sum_ave2, blk) << endl;
    coutf.close();
  }
```

… continues in the next slide …

```cpp
double System :: error(double acc, double acc2, int blk){
  if(blk <= 1) return 0.0;
  else return sqrt( fabs(acc2/double(blk) - pow( acc/double(blk) ,2) )/double(blk) );
}
```

19



System :: averages(int)   2.

```cpp
  // TOTAL ENERGY //////////////////////////////////////////////////////////////
  if (_measure_tenergy){
    coutf.open("../OUTPUT/total_energy.dat",ios::app);
    average  = _average(_index_tenergy);
    sum_average = _global_av(_index_tenergy);
    sum_ave2 = _global_av2(_index_tenergy);
    coutf << setw(12) << blk
          << setw(12) << average
          << setw(12) << sum_average/double(blk)
          << setw(12) << this->error(sum_average, sum_ave2, blk) << endl;
    coutf.close();
  }
  // TEMPERATURE ///////////////////////////////////////////////////////////////
  if (_measure_temp){
    coutf.open("../OUTPUT/temperature.dat",ios::app);
    average  = _average(_index_temp);
    sum_average = _global_av(_index_temp);
    sum_ave2 = _global_av2(_index_temp);
    coutf << setw(12) << blk
          << setw(12) << average
          << setw(12) << sum_average/double(blk)
          << setw(12) << this->error(sum_average, sum_ave2, blk) << endl;
    coutf.close();
  }
  // PRESSURE /////////////////////////////////////////TO BE FIXED IN EXERCISE 4
  // GOFR /////////////////////////////////////////////TO BE FIXED IN EXERCISE 7
  // MAGNETIZATION ////////////////////////////////////TO BE FIXED IN EXERCISE 6
  // SPECIFIC HEAT ////////////////////////////////////TO BE FIXED IN EXERCISE 6
  // SUSCEPTIBILITY ///////////////////////////////////TO BE FIXED IN EXERCISE 6
  // ACCEPTANCE ////////////////////////////////////////////////////////////////
  double fraction;
  coutf.open("../OUTPUT/acceptance.dat",ios::app);
  if(_nattempts > 0) fraction = double(_naccepted)/double(_nattempts);
  else fraction = 0.0;
  coutf << setw(12) << blk << setw(12) << fraction << endl;
  coutf.close();

  return;
}
```

20