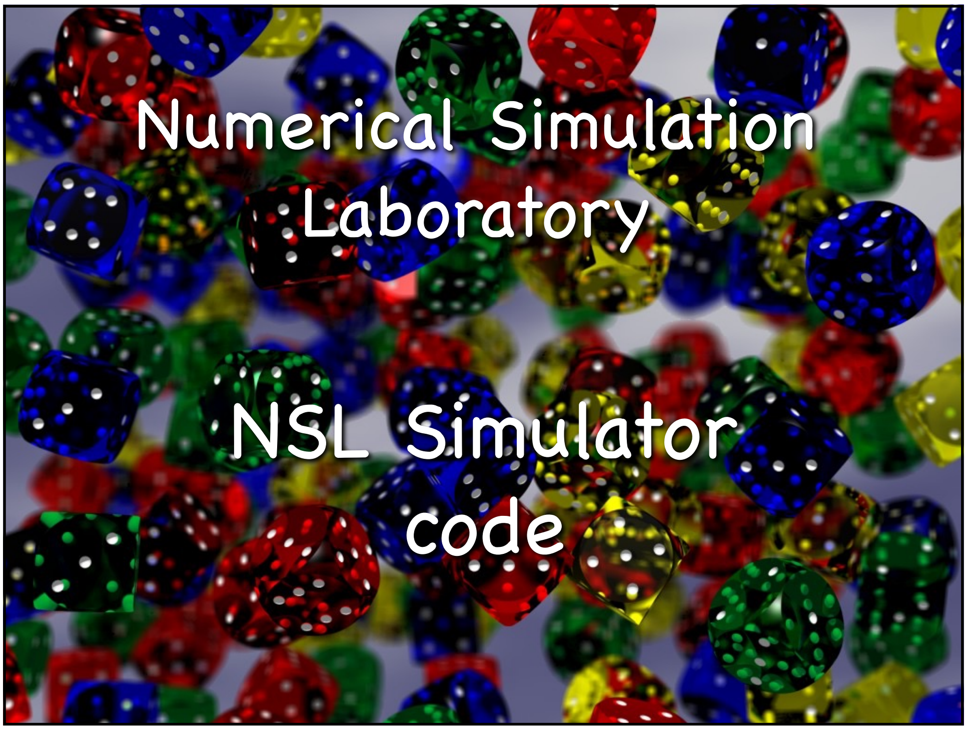
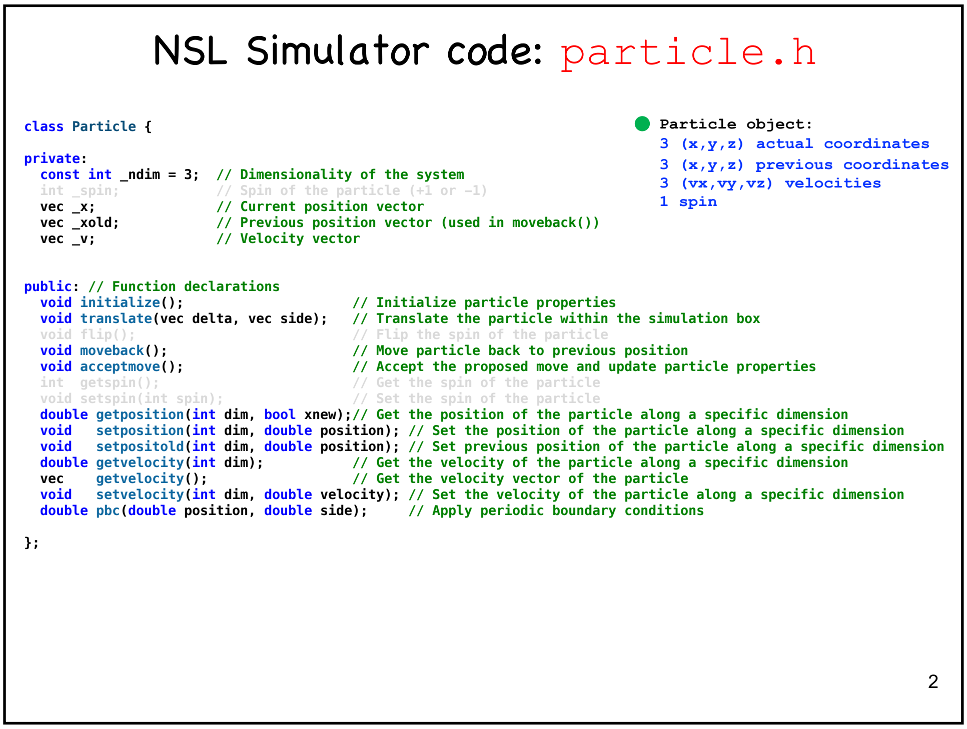


Numerical Simulation Laboratory

NSL Simulator code

1



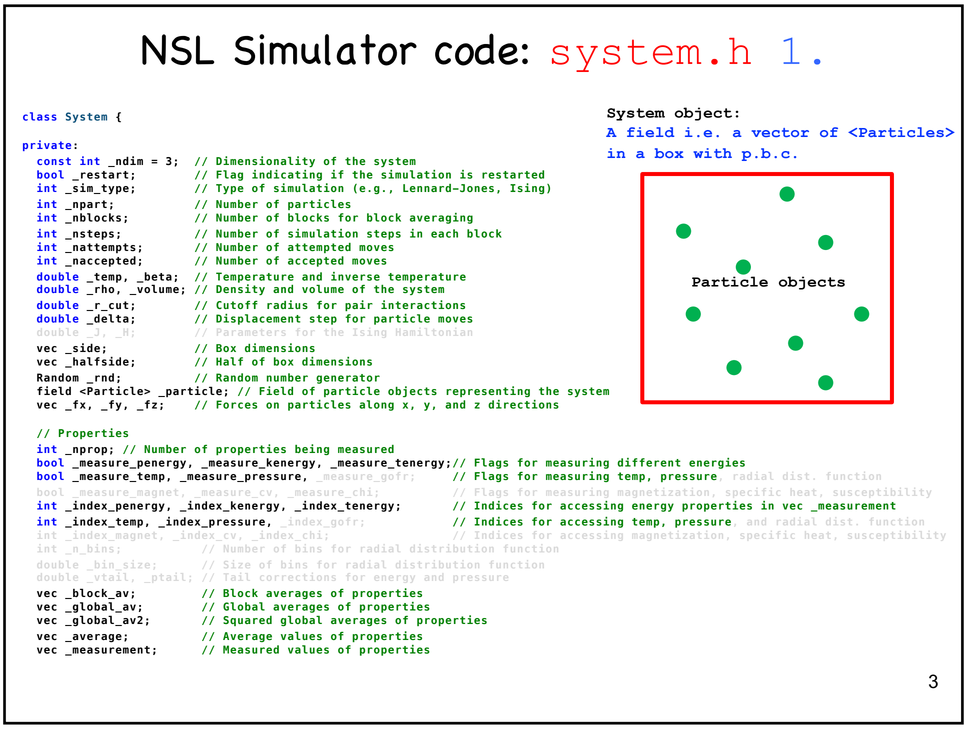
NSL Simulator code: particle.h

```
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.

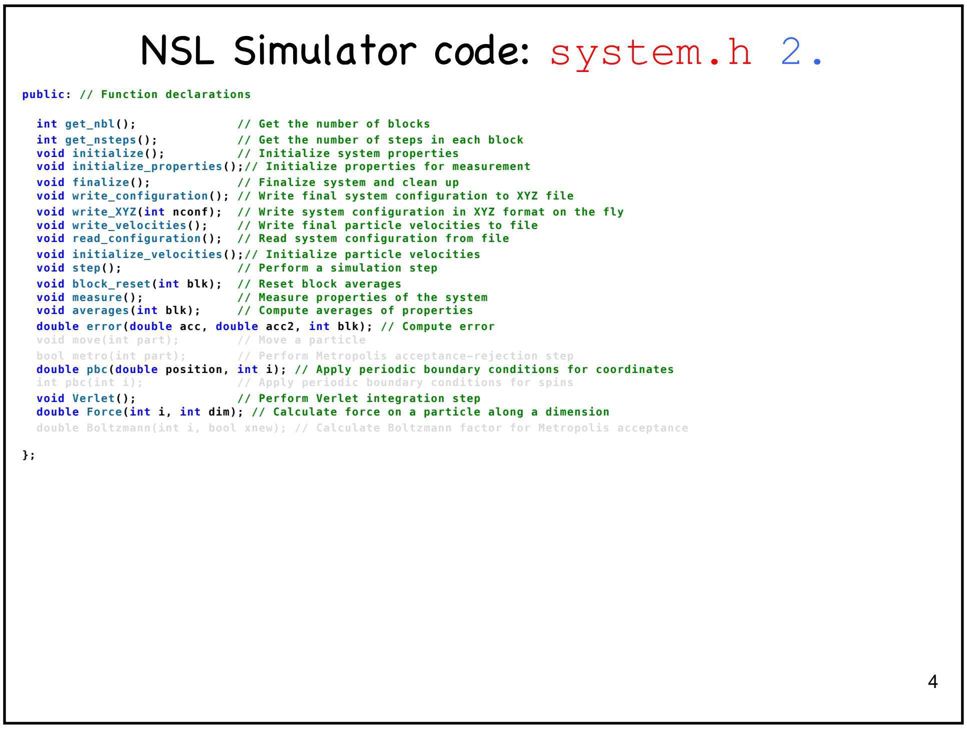
```
class System {

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

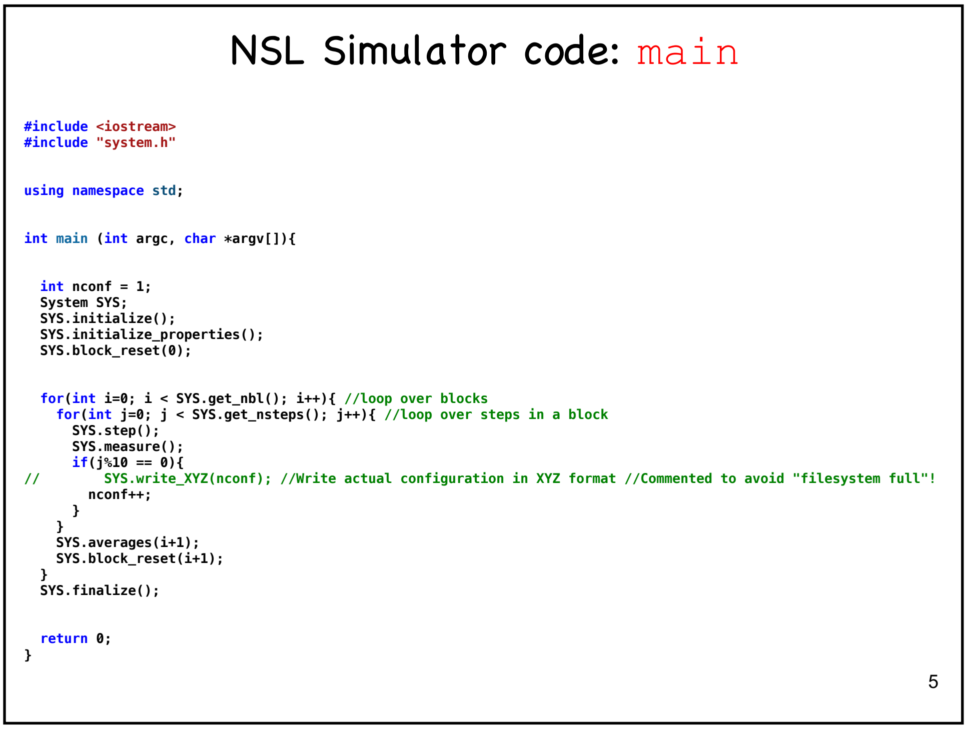


NSL Simulator code: system.h 2.

```
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

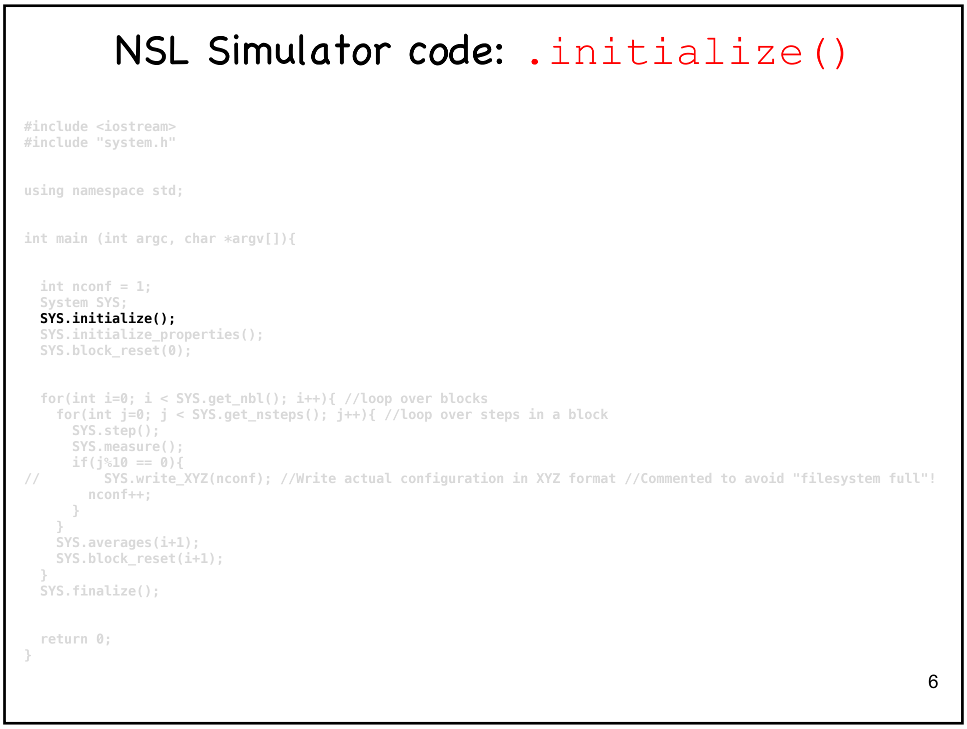


NSL Simulator code: main

```
#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

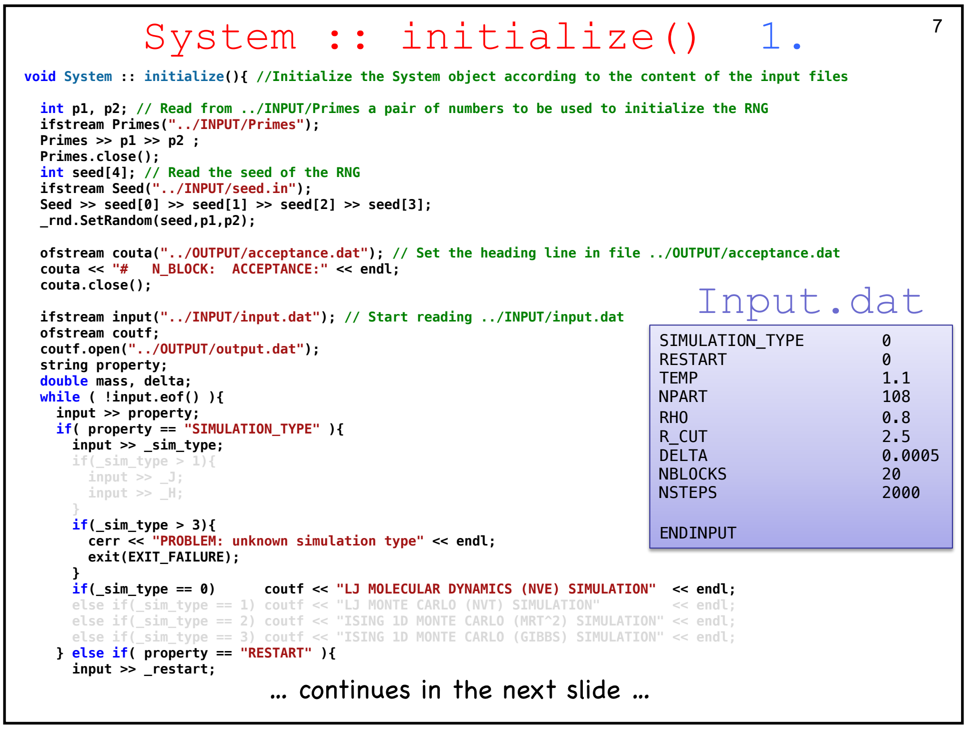


NSL Simulator code: .initialize()

```
#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



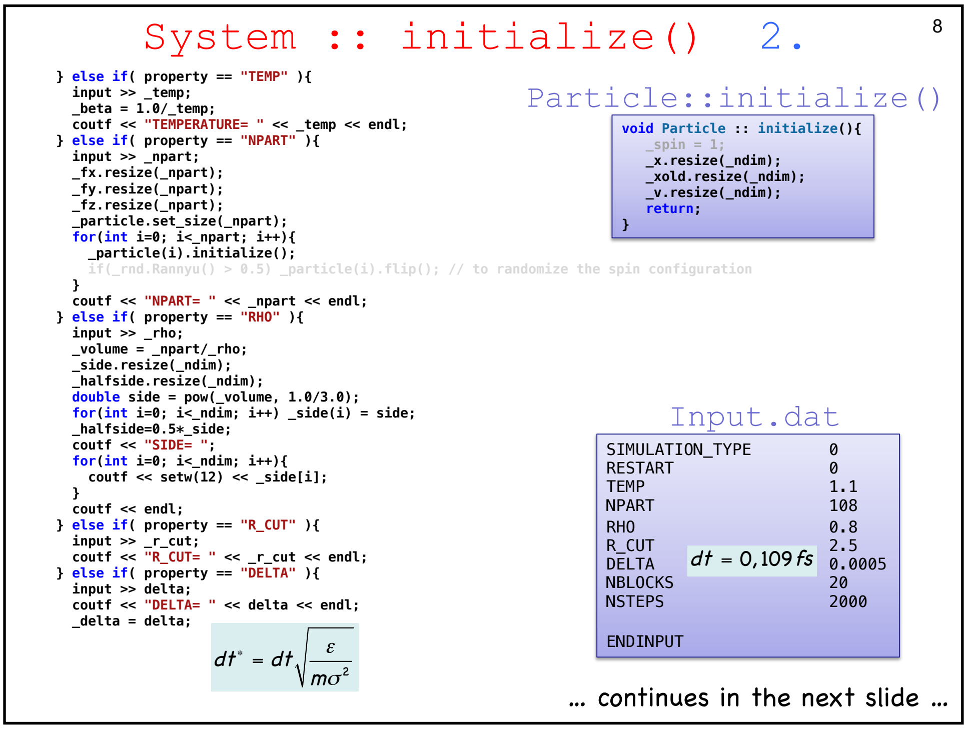
System :: initialize()   1.

```
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        0
RESTART                0
TEMP                   1.1
NPART                  108
RHO                    0.8
R_CUT                  2.5
DELTA                  0.0005
NBLOCKS                20
NSTEPS                 2000

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;
```

$$dt^* = dt\sqrt{\frac{\varepsilon}{m\sigma^2}}$$

Particle::initialize()

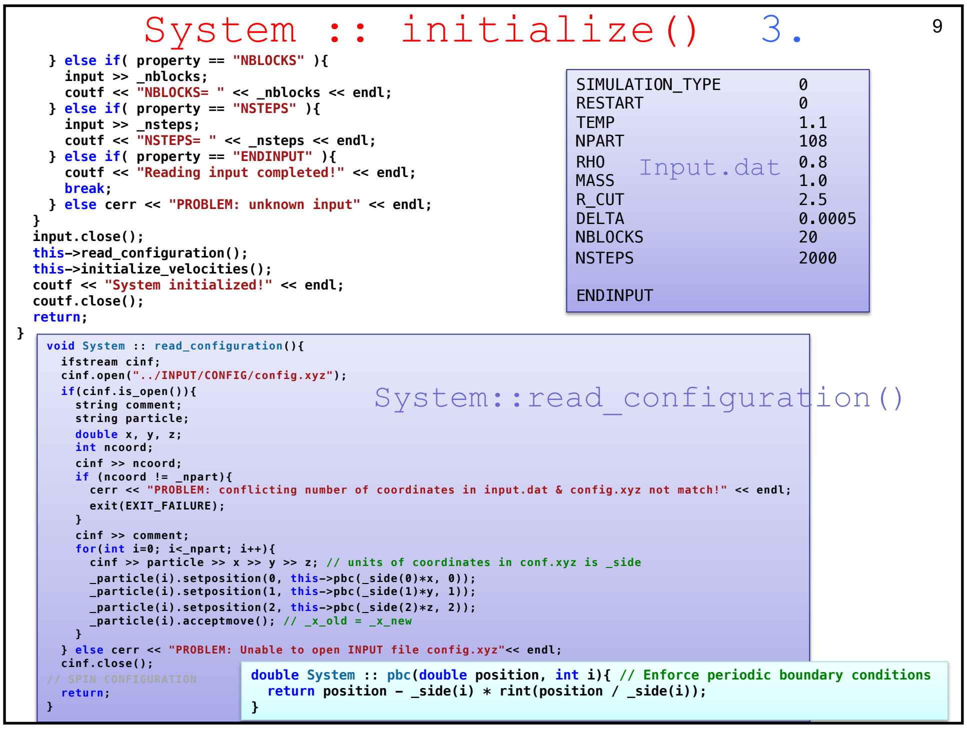
```
void Particle :: initialize(){
   _spin = 1;
   _x.resize(_ndim);
   _xold.resize(_ndim);
   _v.resize(_ndim);
   return;
}
```

Input.dat

```
SIMULATION_TYPE        0
RESTART                0
TEMP                   1.1
NPART                  108
RHO                    0.8
R_CUT                  2.5
DELTA                  0.0005
NBLOCKS                20
NSTEPS                 2000

ENDINPUT
```

$dt = 0{,}109\,fs$

… continues in the next slide …



System :: initialize() 3.

9

```
  } 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;
}
```

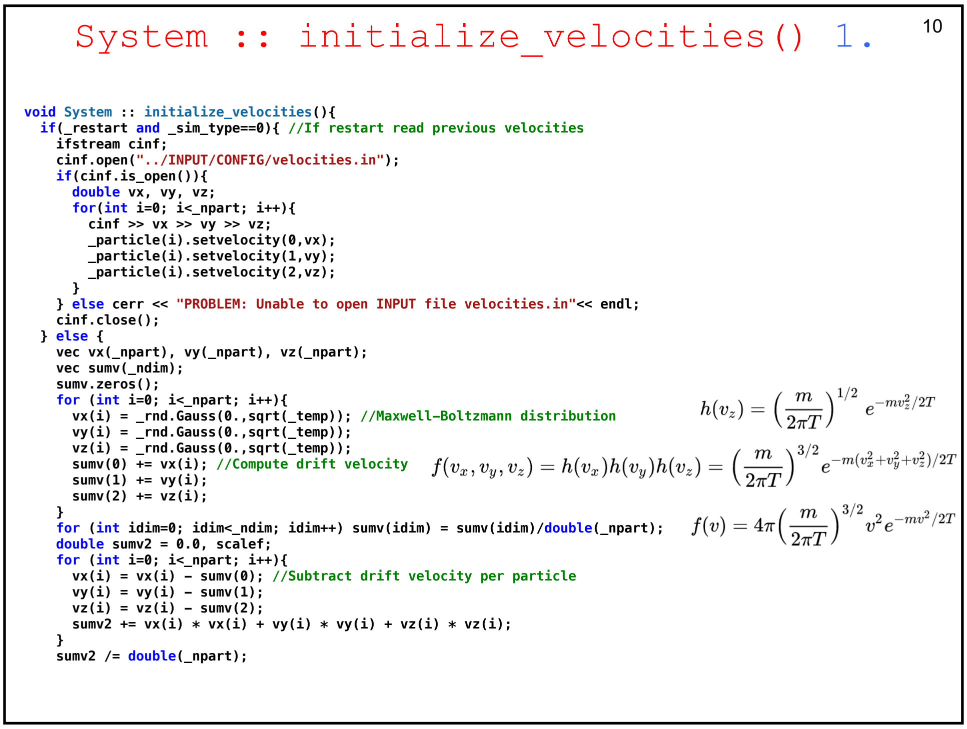
Input.dat

```
SIMULATION_TYPE     0
RESTART             0
TEMP                1.1
NPART               108
RHO                 0.8
MASS                1.0
R_CUT               2.5
DELTA               0.0005
NBLOCKS             20
NSTEPS              2000

ENDINPUT
```

System::read_configuration()

```
void System :: read_configuration(){
  ifstream cinf;
  cinf.open("../INPUT/CONFIG/config.xyz");
  if(cinf.is_open()){
    string comment;
    string particle;
    double x, y, z;
    int ncoord;
    cinf >> ncoord;
    if (ncoord != _npart){
      cerr << "PROBLEM: conflicting number of coordinates in input.dat & config.xyz not match!" << endl;
      exit(EXIT_FAILURE);
    }
    cinf >> comment;
    for(int i=0; i<_npart; i++){
      cinf >> particle >> x >> y >> z; // units of coordinates in conf.xyz is _side
      _particle(i).setposition(0, this->pbc(_side(0)*x, 0));
      _particle(i).setposition(1, this->pbc(_side(1)*y, 1));
      _particle(i).setposition(2, this->pbc(_side(2)*z, 2));
      _particle(i).acceptmove(); // _x_old = _x_new
    }
  } else cerr << "PROBLEM: Unable to open INPUT file config.xyz"<< endl;
  cinf.close();
// SPIN CONFIGURATION
  return;
}
```

```
double System :: pbc(double position, int i){ // Enforce periodic boundary conditions
  return position - _side(i) * rint(position / _side(i));
}
```



System :: initialize_velocities() 1.

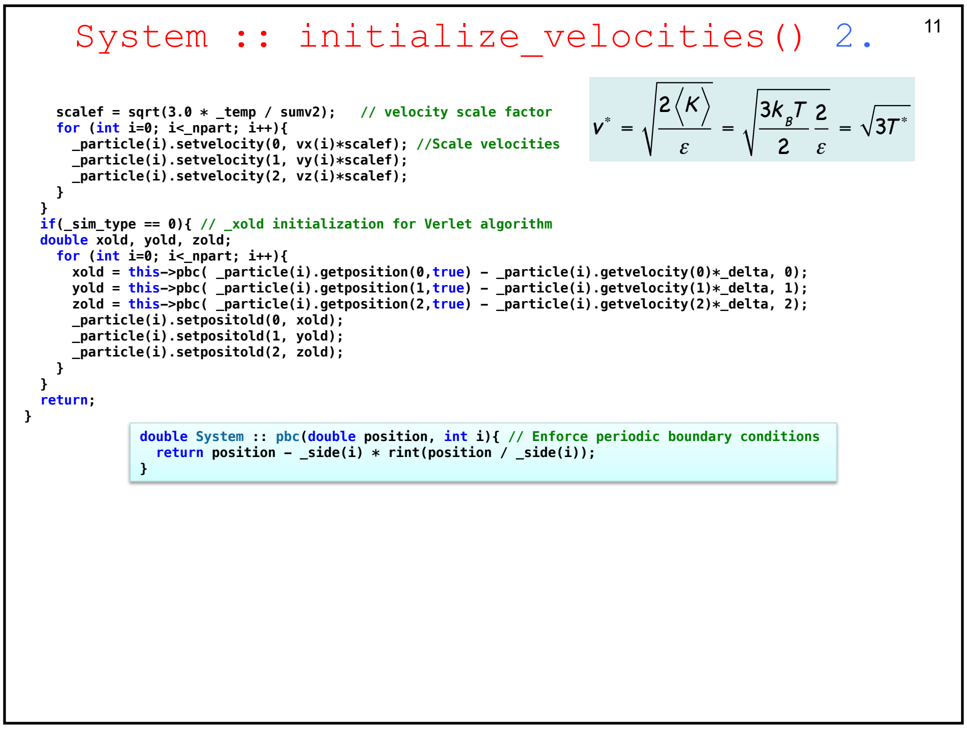
10

```
void System :: initialize_velocities(){
  if(_restart and _sim_type==0){ //If restart read previous velocities
    ifstream cinf;
    cinf.open("../INPUT/CONFIG/velocities.in");
    if(cinf.is_open()){
      double vx, vy, vz;
      for(int i=0; i<_npart; i++){
        cinf >> vx >> vy >> vz;
        _particle(i).setvelocity(0,vx);
        _particle(i).setvelocity(1,vy);
        _particle(i).setvelocity(2,vz);
      }
    } else cerr << "PROBLEM: Unable to open INPUT file velocities.in"<< endl;
    cinf.close();
  } else {
    vec vx(_npart), vy(_npart), vz(_npart);
    vec sumv(_ndim);
    sumv.zeros();
    for (int i=0; i<_npart; i++){
      vx(i) = _rnd.Gauss(0.,sqrt(_temp)); //Maxwell-Boltzmann distribution
      vy(i) = _rnd.Gauss(0.,sqrt(_temp));
      vz(i) = _rnd.Gauss(0.,sqrt(_temp));
      sumv(0) += vx(i); //Compute drift velocity
      sumv(1) += vy(i);
      sumv(2) += vz(i);
    }
    for (int idim=0; idim<_ndim; idim++) sumv(idim) = sumv(idim)/double(_npart);
    double sumv2 = 0.0, scalef;
    for (int i=0; i<_npart; i++){
      vx(i) = vx(i) - sumv(0); //Subtract drift velocity per particle
      vy(i) = vy(i) - sumv(1);
      vz(i) = vz(i) - sumv(2);
      sumv2 += vx(i) * vx(i) + vy(i) * vy(i) + vz(i) * vz(i);
    }
    sumv2 /= double(_npart);
```

$$h(v_z) = \left(\frac{m}{2\pi T}\right)^{1/2} e^{-mv_z^2/2T}$$

$$f(v_x, v_y, v_z) = h(v_x)h(v_y)h(v_z) = \left(\frac{m}{2\pi T}\right)^{3/2} e^{-m(v_x^2+v_y^2+v_z^2)/2T}$$

$$f(v) = 4\pi\left(\frac{m}{2\pi T}\right)^{3/2} v^2 e^{-mv^2/2T}$$



System :: initialize_velocities() 2.

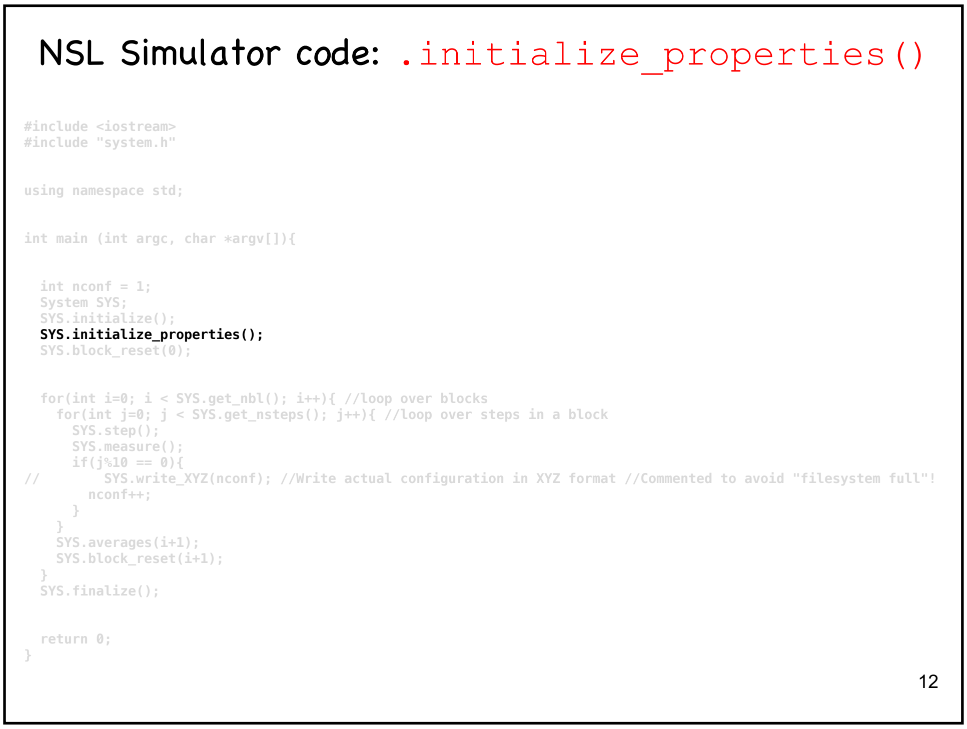
11

```
    scalef = sqrt(3.0 * _temp / sumv2);   // velocity scale factor
    for (int i=0; i<_npart; i++){
      _particle(i).setvelocity(0, vx(i)*scalef); //Scale velocities
      _particle(i).setvelocity(1, vy(i)*scalef);
      _particle(i).setvelocity(2, vz(i)*scalef);
    }
  }
  if(_sim_type == 0){ // _xold initialization for Verlet algorithm
  double xold, yold, zold;
    for (int i=0; i<_npart; i++){
      xold = this->pbc( _particle(i).getposition(0,true) - _particle(i).getvelocity(0)*_delta, 0);
      yold = this->pbc( _particle(i).getposition(1,true) - _particle(i).getvelocity(1)*_delta, 1);
      zold = this->pbc( _particle(i).getposition(2,true) - _particle(i).getvelocity(2)*_delta, 2);
      _particle(i).setpositold(0, xold);
      _particle(i).setpositold(1, yold);
      _particle(i).setpositold(2, zold);
    }
  }
  return;
}
```

$$v^* = \sqrt{\frac{2\langle K\rangle}{\varepsilon}} = \sqrt{\frac{3k_BT}{2}\frac{2}{\varepsilon}} = \sqrt{3T^*}$$

```
double System :: pbc(double position, int i){ // Enforce periodic boundary conditions
  return position - _side(i) * rint(position / _side(i));
}
```

11

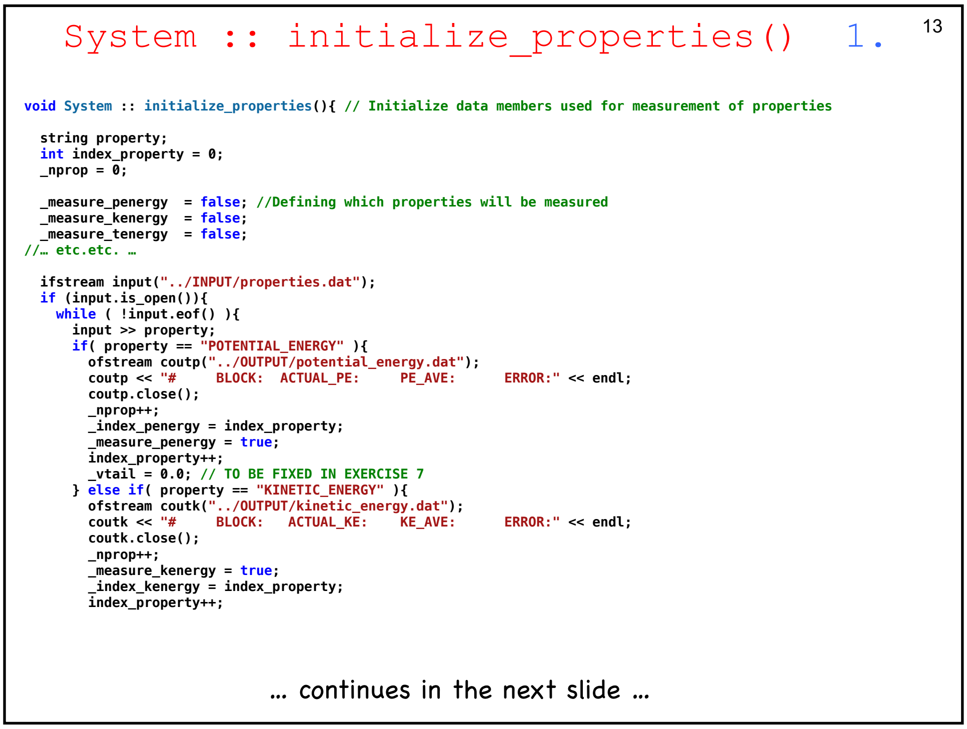


NSL Simulator code: .initialize_properties()

```
#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;
}
```

12

12



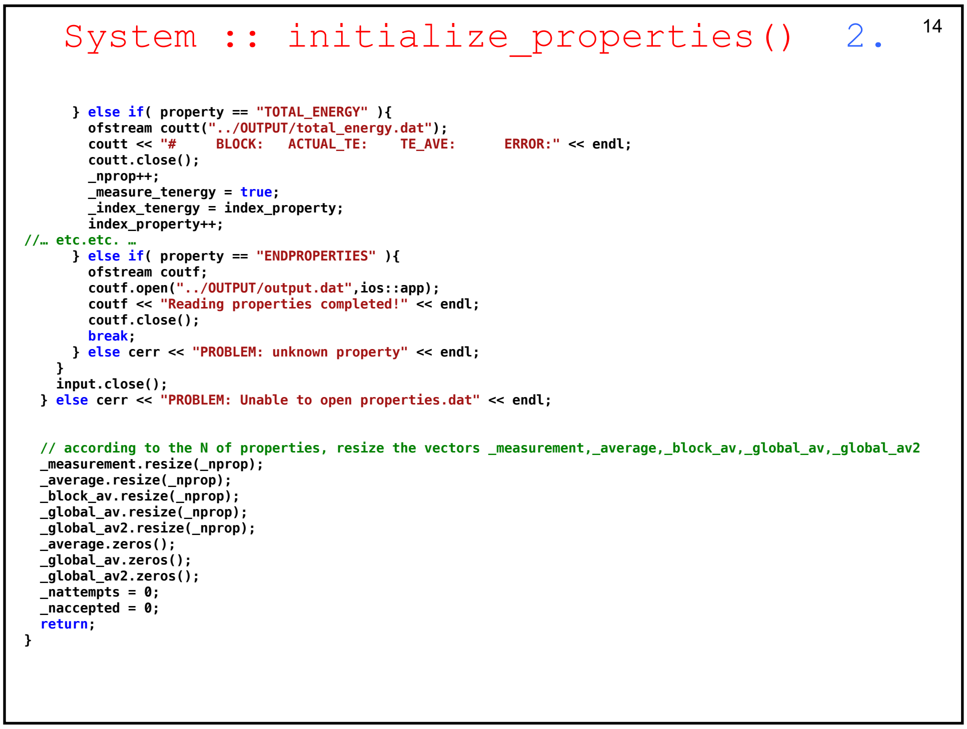
System :: initialize_properties() 1.

13

```
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
  _measure_kenergy  = false;
  _measure_tenergy  = false;
//… etc.etc. …

  ifstream input("../INPUT/properties.dat");
  if (input.is_open()){
    while ( !input.eof() ){
      input >> property;
      if( property == "POTENTIAL_ENERGY" ){
        ofstream coutp("../OUTPUT/potential_energy.dat");
        coutp << "#     BLOCK:  ACTUAL_PE:     PE_AVE:      ERROR:" << endl;
        coutp.close();
        _nprop++;
        _index_penergy = index_property;
        _measure_penergy = true;
        index_property++;
        _vtail = 0.0; // TO BE FIXED IN EXERCISE 7
      } else if( property == "KINETIC_ENERGY" ){
        ofstream coutk("../OUTPUT/kinetic_energy.dat");
        coutk << "#     BLOCK:   ACTUAL_KE:    KE_AVE:      ERROR:" << endl;
        coutk.close();
        _nprop++;
        _measure_kenergy = true;
        _index_kenergy = index_property;
        index_property++;
```

… continues in the next slide …

13

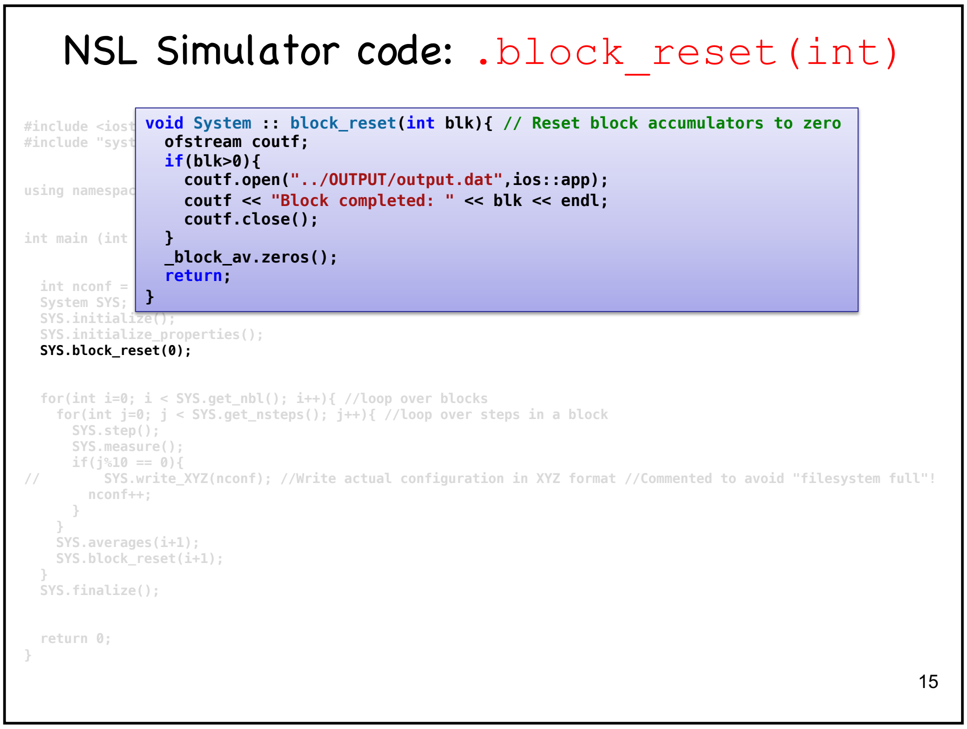


System :: initialize_properties() 2.

14

```
      } else if( property == "TOTAL_ENERGY" ){
        ofstream coutt("../OUTPUT/total_energy.dat");
        coutt << "#     BLOCK:   ACTUAL_TE:    TE_AVE:      ERROR:" << endl;
        coutt.close();
        _nprop++;
        _measure_tenergy = true;
        _index_tenergy = index_property;
        index_property++;
//… etc.etc. …
      } 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;
}
```

14



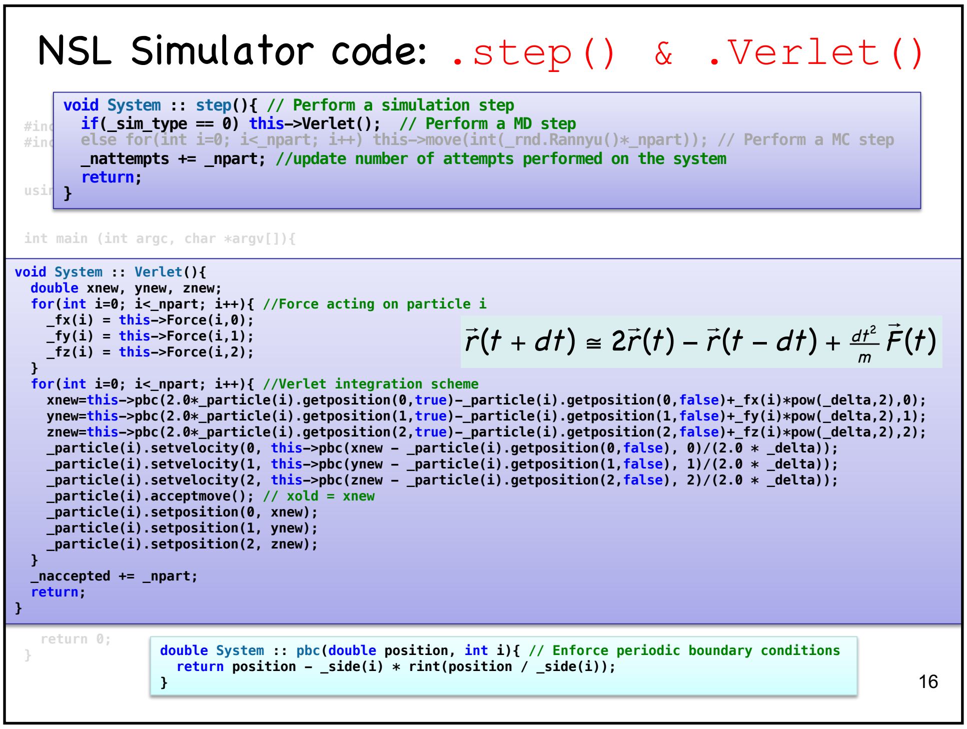
NSL Simulator code: .block_reset(int)

```
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;
}
```

15

15



NSL Simulator code: .step() & .Verlet()

```
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;
}
```

```
#inc
#inc

usin

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

```
void System :: Verlet(){
  double xnew, ynew, znew;
  for(int i=0; i<_npart; i++){ //Force acting on particle i
    _fx(i) = this->Force(i,0);
    _fy(i) = this->Force(i,1);
    _fz(i) = this->Force(i,2);
  }
  for(int i=0; i<_npart; i++){ //Verlet integration scheme
    xnew=this->pbc(2.0*_particle(i).getposition(0,true)-_particle(i).getposition(0,false)+_fx(i)*pow(_delta,2),0);
    ynew=this->pbc(2.0*_particle(i).getposition(1,true)-_particle(i).getposition(1,false)+_fy(i)*pow(_delta,2),1);
    znew=this->pbc(2.0*_particle(i).getposition(2,true)-_particle(i).getposition(2,false)+_fz(i)*pow(_delta,2),2);
    _particle(i).setvelocity(0, this->pbc(xnew - _particle(i).getposition(0,false), 0)/(2.0 * _delta));
    _particle(i).setvelocity(1, this->pbc(ynew - _particle(i).getposition(1,false), 1)/(2.0 * _delta));
    _particle(i).setvelocity(2, this->pbc(znew - _particle(i).getposition(2,false), 2)/(2.0 * _delta));
    _particle(i).acceptmove(); // xold = xnew
    _particle(i).setposition(0, xnew);
    _particle(i).setposition(1, ynew);
    _particle(i).setposition(2, znew);
  }
  _naccepted += _npart;
  return;
}
```

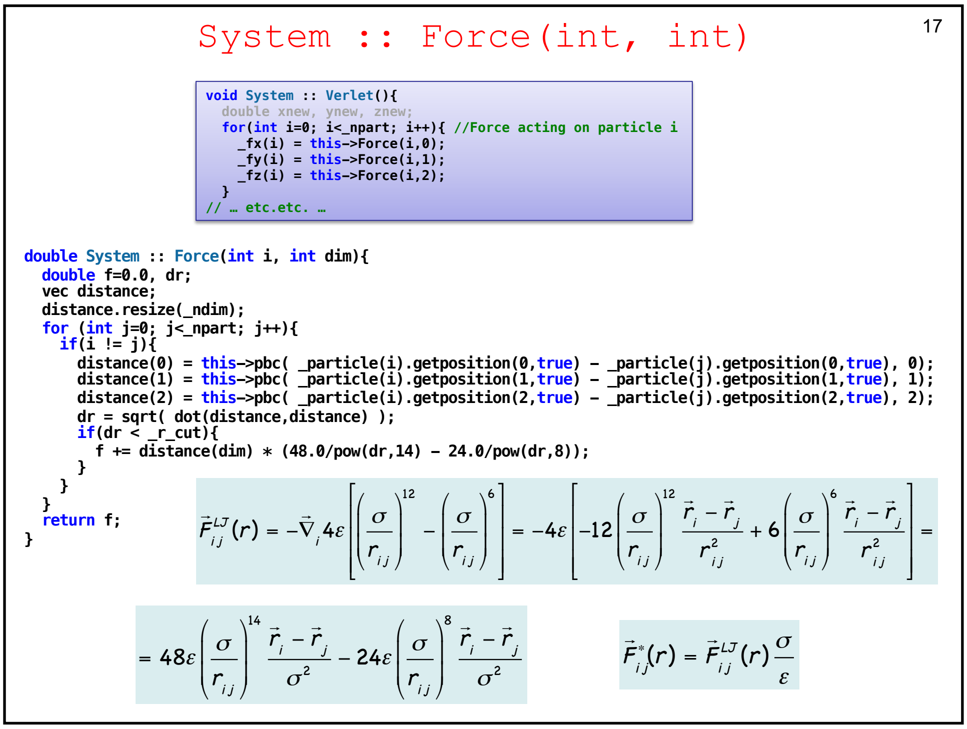
$$\vec{r}(t+dt) \cong 2\vec{r}(t) - \vec{r}(t-dt) + \frac{dt^2}{m}\vec{F}(t)$$

```
  return 0;
}
```

```
double System :: pbc(double position, int i){ // Enforce periodic boundary conditions
  return position - _side(i) * rint(position / _side(i));
}
```

16

16



System :: Force(int, int)

17

```
void System :: Verlet(){
  double xnew, ynew, znew;
  for(int i=0; i<_npart; i++){ //Force acting on particle i
    _fx(i) = this->Force(i,0);
    _fy(i) = this->Force(i,1);
    _fz(i) = this->Force(i,2);
  }
// … etc.etc. …
```

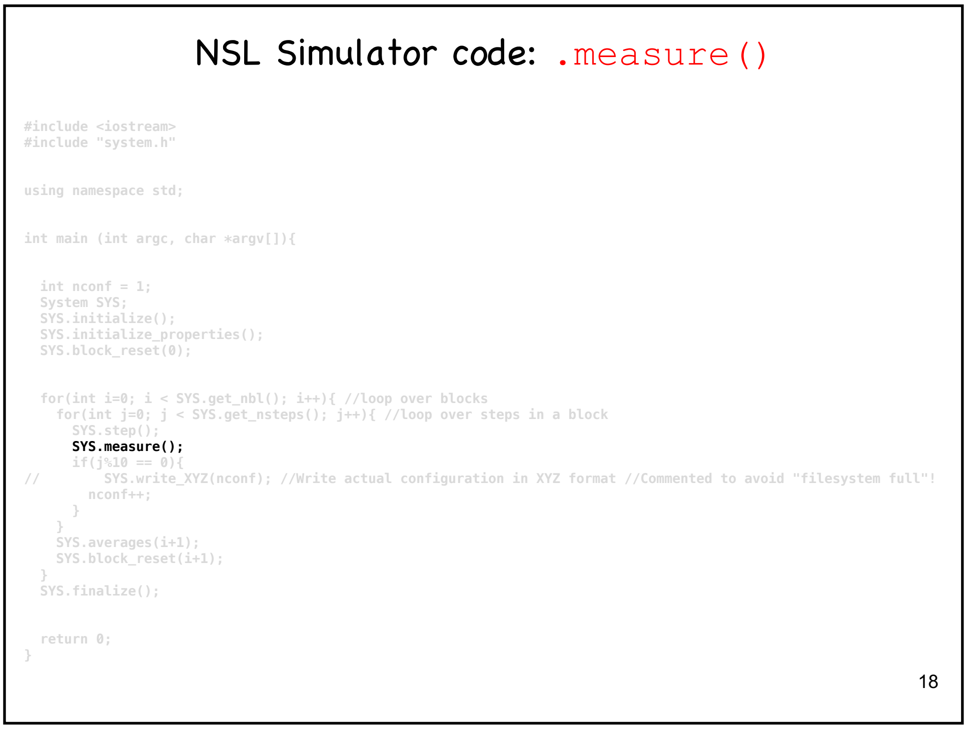
```
double System :: Force(int i, int dim){
  double f=0.0, dr;
  vec distance;
  distance.resize(_ndim);
  for (int j=0; j<_npart; j++){
    if(i != j){
      distance(0) = this->pbc( _particle(i).getposition(0,true) - _particle(j).getposition(0,true), 0);
      distance(1) = this->pbc( _particle(i).getposition(1,true) - _particle(j).getposition(1,true), 1);
      distance(2) = this->pbc( _particle(i).getposition(2,true) - _particle(j).getposition(2,true), 2);
      dr = sqrt( dot(distance,distance) );
      if(dr < _r_cut){
        f += distance(dim) * (48.0/pow(dr,14) - 24.0/pow(dr,8));
      }
    }
  }
  return f;
}
```

$$\vec{F}^{LJ}_{ij}(r) = -\vec{\nabla}_i 4\varepsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right] = -4\varepsilon\left[-12\left(\frac{\sigma}{r_{ij}}\right)^{12}\frac{\vec{r}_i-\vec{r}_j}{r_{ij}^2} + 6\left(\frac{\sigma}{r_{ij}}\right)^{6}\frac{\vec{r}_i-\vec{r}_j}{r_{ij}^2}\right] =$$

$$= 48\varepsilon\left(\frac{\sigma}{r_{ij}}\right)^{14}\frac{\vec{r}_i-\vec{r}_j}{\sigma^2} - 24\varepsilon\left(\frac{\sigma}{r_{ij}}\right)^{8}\frac{\vec{r}_i-\vec{r}_j}{\sigma^2}$$

$$\vec{F}^{*}_{ij}(r) = \vec{F}^{LJ}_{ij}(r)\frac{\sigma}{\varepsilon}$$

17

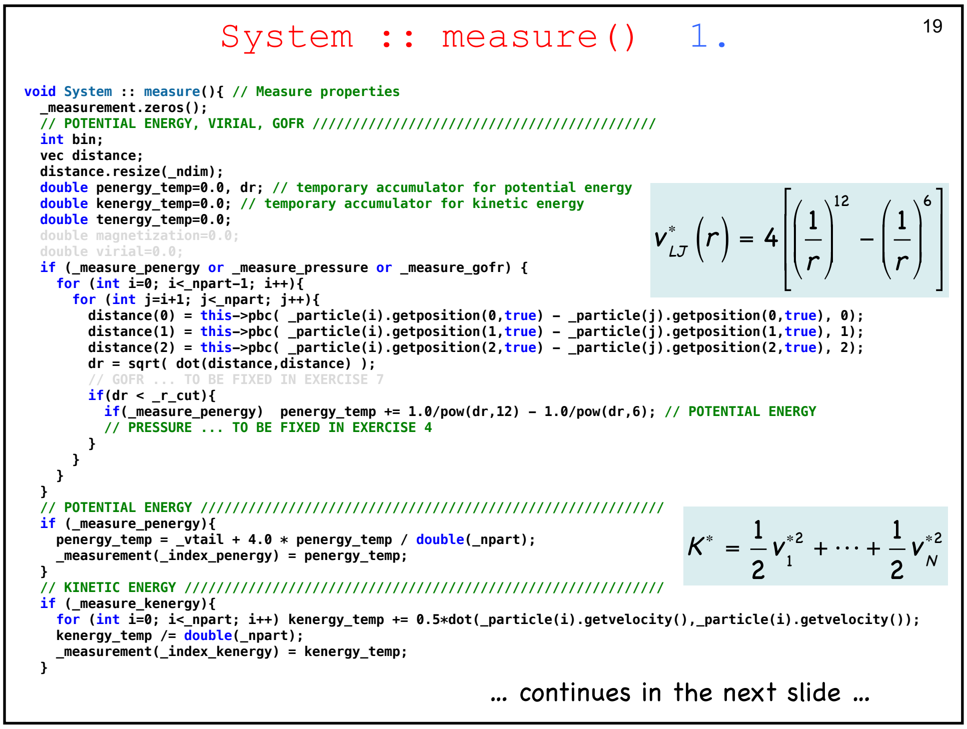


NSL Simulator code: .measure()

```
#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;
}
```

18

18



System :: measure()   1.

19

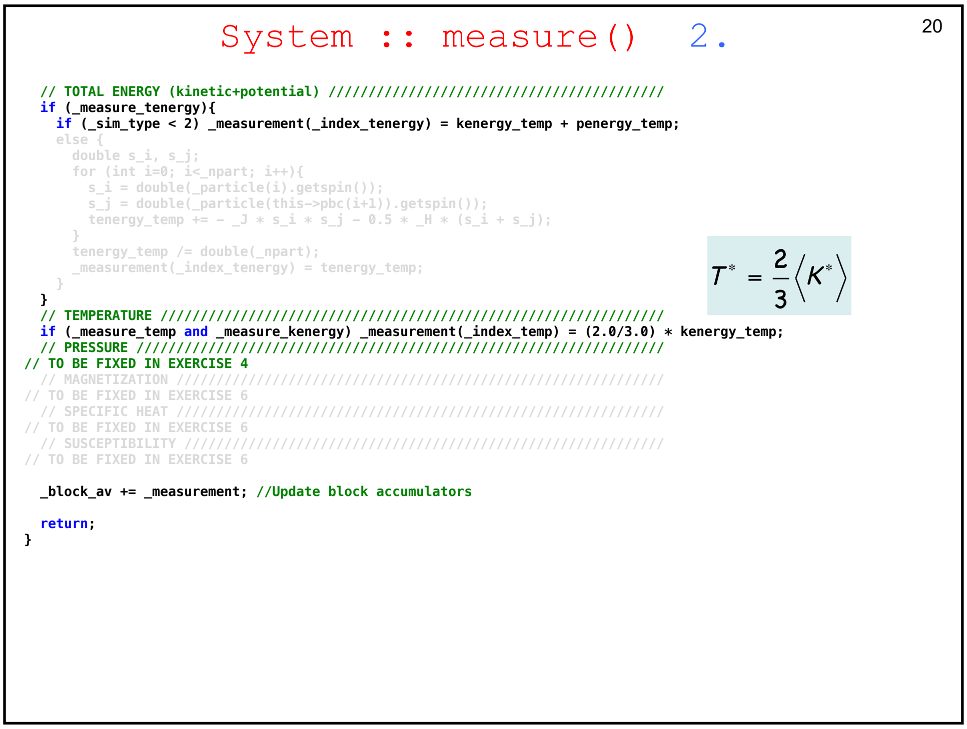
```
void System :: measure(){ // Measure properties
  _measurement.zeros();
  // POTENTIAL ENERGY, VIRIAL, GOFR ///////////////////////////////////////////
  int bin;
  vec distance;
  distance.resize(_ndim);
  double penergy_temp=0.0, dr; // temporary accumulator for potential energy
  double kenergy_temp=0.0; // temporary accumulator for kinetic energy
  double tenergy_temp=0.0;
  double magnetization=0.0;
  double virial=0.0;
  if (_measure_penergy or _measure_pressure or _measure_gofr) {
    for (int i=0; i<_npart-1; i++){
      for (int j=i+1; j<_npart; j++){
        distance(0) = this->pbc( _particle(i).getposition(0,true) - _particle(j).getposition(0,true), 0);
        distance(1) = this->pbc( _particle(i).getposition(1,true) - _particle(j).getposition(1,true), 1);
        distance(2) = this->pbc( _particle(i).getposition(2,true) - _particle(j).getposition(2,true), 2);
        dr = sqrt( dot(distance,distance) );
        // GOFR ... TO BE FIXED IN EXERCISE 7
        if(dr < _r_cut){
          if(_measure_penergy)  penergy_temp += 1.0/pow(dr,12) - 1.0/pow(dr,6); // POTENTIAL ENERGY
          // PRESSURE ... TO BE FIXED IN EXERCISE 4
        }
      }
    }
  }
  // POTENTIAL ENERGY ///////////////////////////////////////////////////////
  if (_measure_penergy){
    penergy_temp = _vtail + 4.0 * penergy_temp / double(_npart);
    _measurement(_index_penergy) = penergy_temp;
  }
  // KINETIC ENERGY /////////////////////////////////////////////////////////
  if (_measure_kenergy){
    for (int i=0; i<_npart; i++) kenergy_temp += 0.5*dot(_particle(i).getvelocity(),_particle(i).getvelocity());
    kenergy_temp /= double(_npart);
    _measurement(_index_kenergy) = kenergy_temp;
  }
```

$$v^*_{LJ}(r) = 4\left[\left(\frac{1}{r}\right)^{12} - \left(\frac{1}{r}\right)^{6}\right]$$

$$K^* = \frac{1}{2}v_1^{*2} + \cdots + \frac{1}{2}v_N^{*2}$$

… continues in the next slide …

19



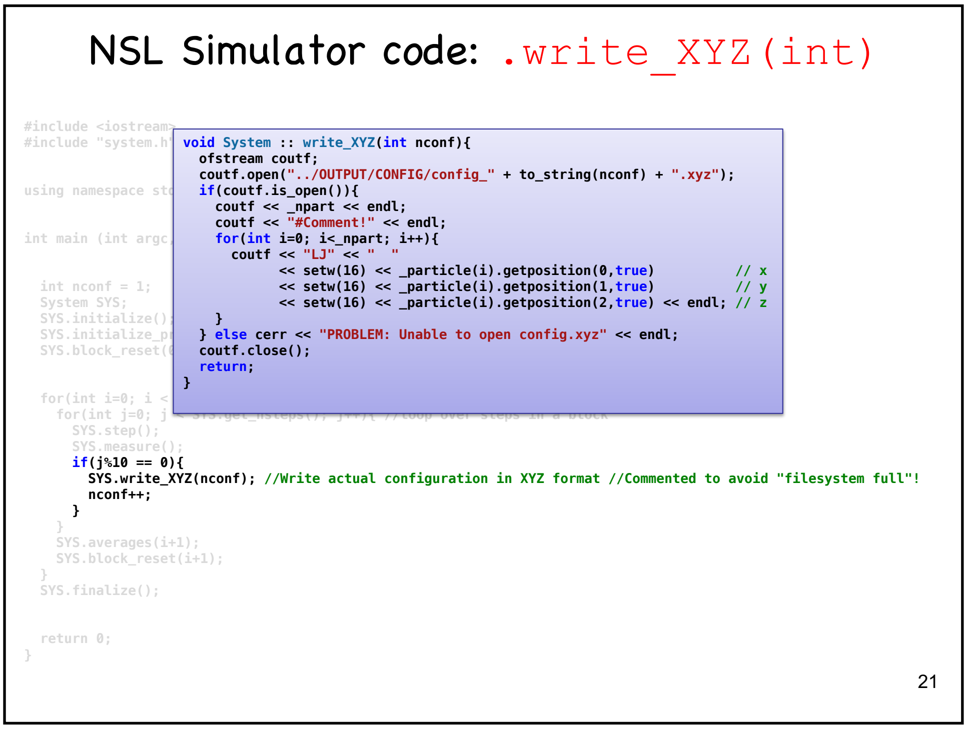
System :: measure()   2.

20

```
  // 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 ////////////////////////////////////////////////////////////
  if (_measure_temp and _measure_kenergy) _measurement(_index_temp) = (2.0/3.0) * kenergy_temp;
  // 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;
}
```

$$T^* = \frac{2}{3}\left\langle K^* \right\rangle$$

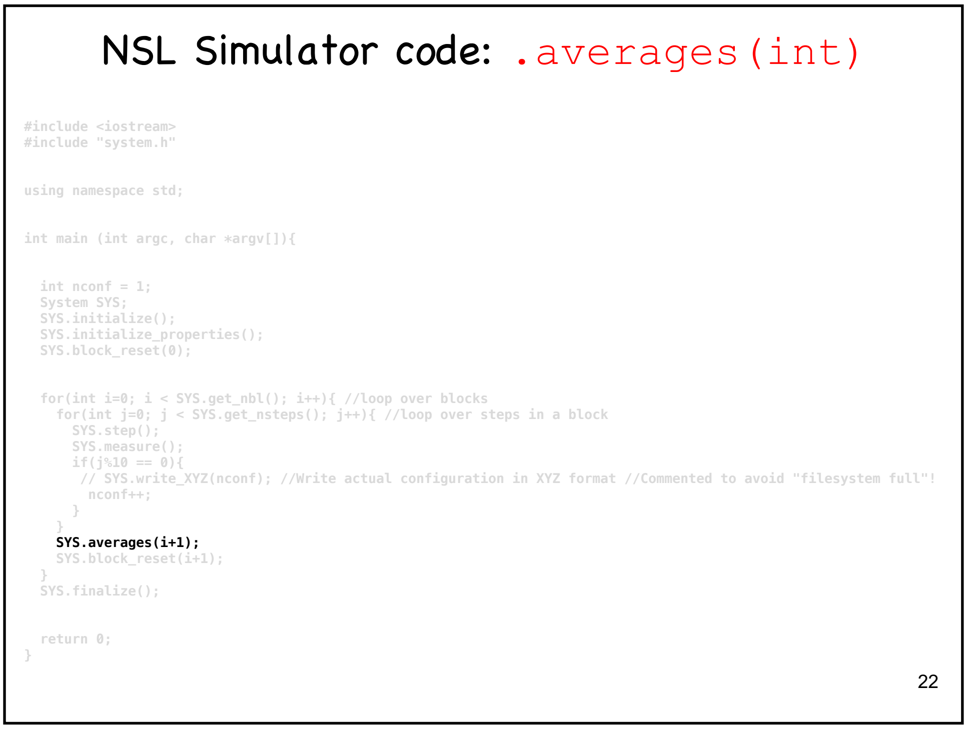
20



NSL Simulator code: .write_XYZ(int)

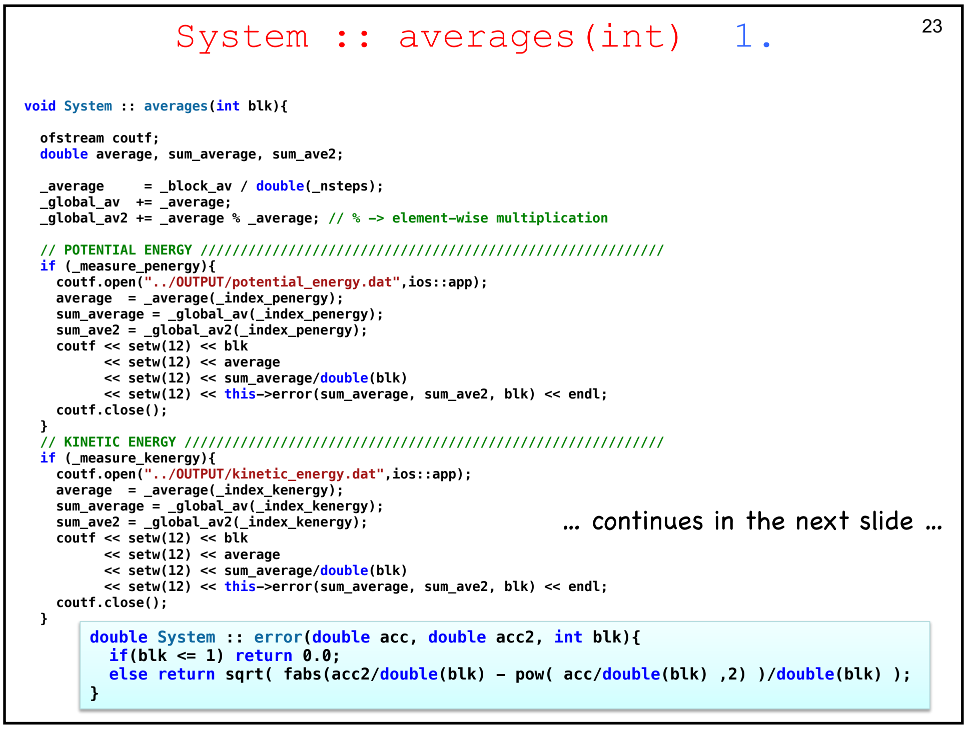
```
#include <iostream>
#include "system.h"

using namespace std

int main (int argc,

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

  for(int i=0; i <
    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;
}
```

```
void System :: write_XYZ(int nconf){
  ofstream coutf;
  coutf.open("../OUTPUT/CONFIG/config_" + to_string(nconf) + ".xyz");
  if(coutf.is_open()){
    coutf << _npart << endl;
    coutf << "#Comment!" << endl;
    for(int i=0; i<_npart; i++){
      coutf << "LJ" << "  "
            << setw(16) << _particle(i).getposition(0,true)          // x
            << setw(16) << _particle(i).getposition(1,true)          // y
            << setw(16) << _particle(i).getposition(2,true) << endl; // z
    }
  } else cerr << "PROBLEM: Unable to open config.xyz" << endl;
  coutf.close();
  return;
}
```



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;
}
```



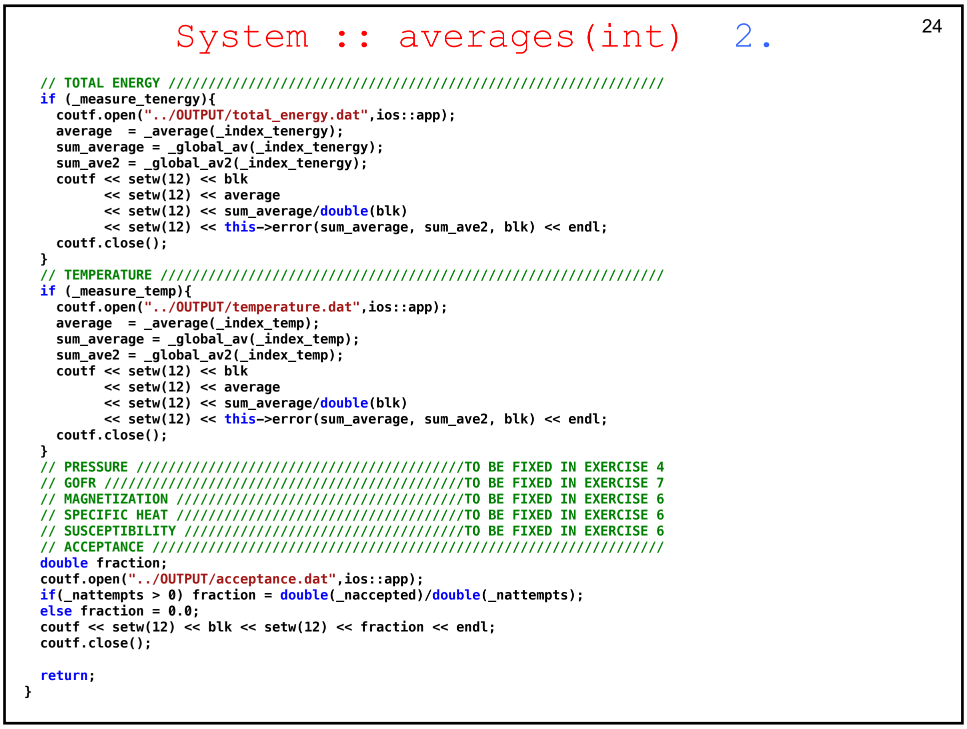
System :: averages(int) 1.

```
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 …

```
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) );
}
```

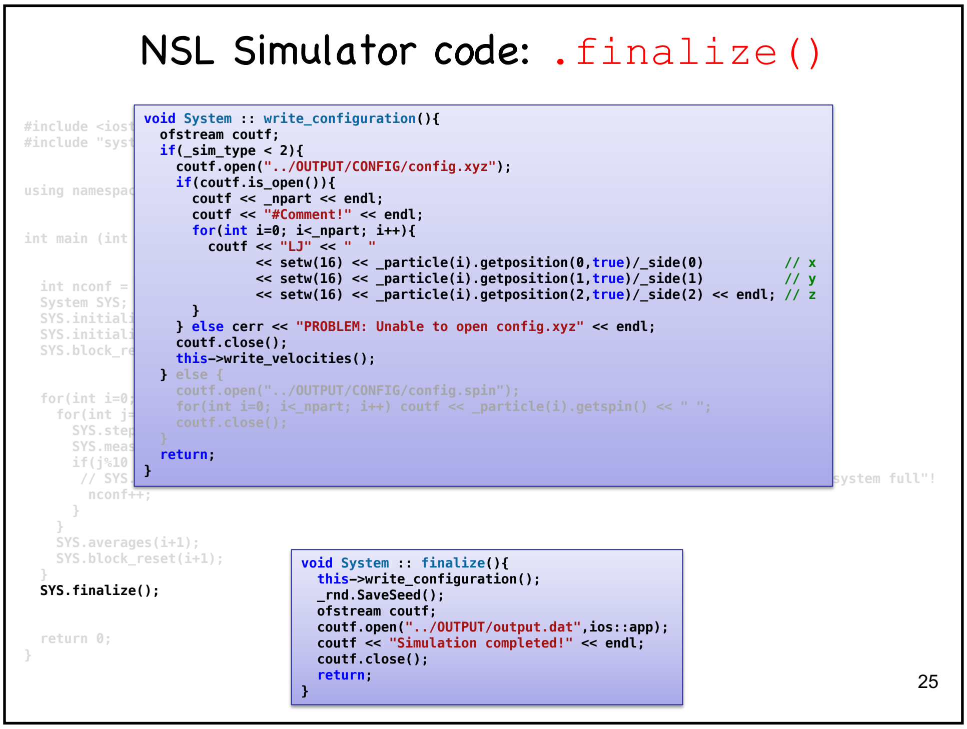
23



System :: averages(int) 2.

```
  // 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;
}
```

24



NSL Simulator code: .finalize()

```
#include <iost
#include "syst

using namespac

int main (int

  int nconf =
  System SYS;
  SYS.initiali
  SYS.initiali
  SYS.block_re

  for(int i=0;
    for(int j=
      SYS.step
      SYS.meas
      if(j%10
       // SYS.                                                               system full"!
        nconf++;
      }
    }
    SYS.averages(i+1);
    SYS.block_reset(i+1);
  }
  SYS.finalize();

  return 0;
}
```

```
void System :: write_configuration(){
  ofstream coutf;
  if(_sim_type < 2){
    coutf.open("../OUTPUT/CONFIG/config.xyz");
    if(coutf.is_open()){
      coutf << _npart << endl;
      coutf << "#Comment!" << endl;
      for(int i=0; i<_npart; i++){
        coutf << "LJ" << "  "
              << setw(16) << _particle(i).getposition(0,true)/_side(0)          // x
              << setw(16) << _particle(i).getposition(1,true)/_side(1)          // y
              << setw(16) << _particle(i).getposition(2,true)/_side(2) << endl; // z
      }
    } else cerr << "PROBLEM: Unable to open config.xyz" << endl;
    coutf.close();
    this->write_velocities();
  } else {
    coutf.open("../OUTPUT/CONFIG/config.spin");
    for(int i=0; i<_npart; i++) coutf << _particle(i).getspin() << " ";
    coutf.close();
  }
  return;
}
```

```
void System :: finalize(){
  this->write_configuration();
  _rnd.SaveSeed();
  ofstream coutf;
  coutf.open("../OUTPUT/output.dat",ios::app);
  coutf << "Simulation completed!" << endl;
  coutf.close();
  return;
}
```

25

25