Eric Brower
Abi Emmanuel
Cameron McMillan

The following pdf will be sorted based on the person responsible for writing the code.

------------------ Eric Brower ------------------
These functions are basically random unrelated functions that have to be assigned to somebody.

double* GetBrownian();
GetBrownian takes no parameters. Since it's random, it is independent of any particle and any simulation
This function returns an array of doubles. There are three doubles in this array because there are three dimensions in xyz!
This function is defined in Particle.h

Simulation* CreateSimulation(double l, int b);
CreateSimulation takes two parameters. l is a double which defines the length, width, and height of the box. b is an integer which is how many particles are in the box.
This thing returns a pointer pointing to a Simulation struct. That's how we know where the Simulation ended up in memory!
This function is defined in Simulation.h

int CreateParticles(Simulation sim);
CreateParticles takes one parameter. The only parameter you need is a pointer to the simulation! That way, we populate the simulation with enough particles that are randomly dispersed. The simulation knows how many particles to exist because of the "b" parameter we send in when we create it.
This function returns an int, just to let you know if everything worked out all right. If it's negative that's bad! If it's not, then it's *probably good!
This function is defined in Simulation.h

------------------ Abi Emmanuel ------------------
All these functions have to do with reading and setting positions of some particular particle in some simulation.

double* GetPosition(Simulation sim, int partIndex);
GetPosition takes two parameters, it's the simulation pointer and the index to find the particle in that particular simulation.
This thing returns an array of doubles. There are three! There are three because there are three dimensions in our dimension, or at least in this simulation probably!
This function is defined in Particle.h

void WriteToFile(Simulation sim);
This function takes in the simulation, and writes out the xyz file so we can look at it.
Returns nothing, so we just gotta hope it worked once we see the file.
This function is defined in Simulation.h

------------------- Cameron McMillan -------------------
All these functions have to do with the update step in the simulation

void Update(Simulation sim);
Update takes in one param, it's the sim pointer again. The timestamp is defined in the H file, so we don't have to worry about that. This function will iterate through all the particles in the particle array and do some math, also update positions and whatnot.
Returns nothing because we are going to be calling this function (potentially) a lot.
This function is defined in Simulation.h

double GetParticleDistance(Simulation sim, int partIndexA, int partIndexB);
GetParticleDistance takes in three params. It's the sim (of course) and the indexes (indices?) of the two particles we care about at this particular moment. Using some math, we will figure out the closest distance between the particles.
This function returns a double which is the distance!
This function is defined in Simulation.h

double* SimulationNetForce(Simulation sim, int partIndex);
SimulationNetForce takes in two params: a simulation pointer and a particular index of a particle in the simulation. This function will find the net force acting on this particle based on proximity to other particles.
This thing returns an array of doubles. This will correspond to the three-part force vector so we can update the particle position later.
This function is defined in Simulation.h

double* SetPosition(Simulation sim, int partIndex, double x, double y, double z);
SetPosition takes in five parameters! You will need a pointer for the simulation, and an integer for the particle index. This way, the user doesn't have to handle any weird Particle structs, we don't want to confuse them. The simulation will (probably) be able to find the particle you are looking for, and then you can update the three coordinates, which are all doubles.
This function also does all the arithmetic to make sure the particle stays inside the simulated box. We have to make sure that the particle remains in the box following the rules in the instructions. So if we give this function a particle with an invalid position (outside the box) we can update the particle to the correct position (inside the box) based on the work of this function.
This function returns the vector that you set the particle position to, we do this just so it's convenient for the programming (good practice).
This function is defined in Simulation.h