//

// main.cpp

// Struct1

//

// Created by Zahra Khatami on 4/1/15.

// Copyright (c) 2015 Zahra Khatami. All rights reserved.

#include <iostream>

#include <string>

#include <sstream>

#include <vector>

#include <math.h>

#include <fstream>

#include <cmath>

#include <array>

#include <boost/timer/timer.hpp>

using std::vector;

using namespace std;

typedef basic\_ofstream<char> ofstream;

//Storing Data

struct Parameters{

int number;

};

struct Body {

int ID1,parent;

double m1,r1[3],v1[3],force=0.0;

vector<int> list\_node1, list\_node2;

};

struct Cell {

int ID2, parent2,NumNodes,level,neighbors[6],Ncell;

vector<int> members, child,scell;

double r2[3],rd[3],m2,boundary[6];

};

//Read input from txt file

void read\_Input(Body body[]){

int N; string temp,temp1;

fstream textfile;

string::size\_type sz;

textfile.open("ex10000.txt");

textfile>>temp;

N = atoi(temp.c\_str());

for (int i=0; i<N; ++i){

textfile>>temp>>temp1;

body[i].ID1=atoi(temp.c\_str()); body[i].m1=stof(temp1,&sz);

for (int j=2; j<5; ++j){

textfile>>temp; body[i].r1[j-2]=stof(temp,&sz);}

for (int j=5; j<8; ++j){

textfile>>temp; body[i].v1[j-5]=stof(temp,&sz);}

body[i].parent=0;}}

//Detremining max and min in the boundary cells

void apply\_changes(Cell cell[],int n, Body body[]){

for(int j=0; j<cell[n].NumNodes; ++j){

int i=cell[n].members[j];

if (body[i].r1[0]<cell[n].boundary[0]) cell[n].boundary[0]=body[i].r1[0];

if (body[i].r1[0]>cell[n].boundary[1]) cell[n].boundary[1]=body[i].r1[0];

if (body[i].r1[1]<cell[n].boundary[2]) cell[n].boundary[2]=body[i].r1[1];

if (body[i].r1[1]>cell[n].boundary[3]) cell[n].boundary[3]=body[i].r1[1];

if (body[i].r1[2]<cell[n].boundary[4]) cell[n].boundary[4]=body[i].r1[2];

if (body[i].r1[2]>cell[n].boundary[5]) cell[n].boundary[5]=body[i].r1[2];}}

//Finding out if the node is in that subcubic

bool InCube(Body body[], Cell cell[], int node, int p){

for (int i=0; i<3; ++i)

if(body[node].r1[i]<cell[p].boundary[2\*i] || body[node].r1[i]>cell[p].boundary[2\*i+1])

return false;

return true;

}

//enlarging cell

/\*int \*extend\_cell(Cell cell[], int size){

int newsize = size+8;

int \*newvalues = new int[newsize];

for( int i = 0; i < newsize; i++ )

newvalues[i] = cell[i];

delete[] cell; //deallocate arrays with delete[]

return newvalues;

delete[] newvalues; //Don't forget newvalues too.

//}

}\*/

//Determining the boundry of the each subcube

void det\_boundary\_subcube(Cell cell[],int n, Body body[]){

double a1,a2,a3,b1,b2,b3,c1,c2,c3;

int p=(cell[n].Ncell)\*8;

a1=cell[n].boundary[0]; a2=(cell[n].boundary[0]+cell[n].boundary[1])/2; a3=cell[n].boundary[1];

b1=cell[n].boundary[2]; b2=(cell[n].boundary[2]+cell[n].boundary[3])/2; b3=cell[n].boundary[3];

c1=cell[n].boundary[4]; c2=(cell[n].boundary[4]+cell[n].boundary[5])/2; c3=cell[n].boundary[5];

double A1[]={a1,a2,b1,b2,c1,c2},A2[]={a1,a2,b1,b2,c2,c3},A3[]={a1,a2,b2,b3,c1,c2}, A4[]={a1,a2,b2,b3,c2,c3}, A5[]={a2,a3,b1,b2,c1,c2}, A6[]={a2,a3,b1,b2,c2,c3}, A7[]={a2,a3,b2,b3,c1,c2}, A8[]={a2,a3,b2,b3,c2,c3};

copy(begin(A1),end(A1),begin(cell[1+p].boundary)); copy(begin(A2),end(A2),begin(cell[2+p].boundary));

copy(begin(A3),end(A3),begin(cell[3+p].boundary)); copy(begin(A4),end(A4),begin(cell[4+p].boundary));

copy(begin(A5),end(A5),begin(cell[5+p].boundary)); copy(begin(A6),end(A6),begin(cell[6+p].boundary));

copy(begin(A7),end(A7),begin(cell[7+p].boundary)); copy(begin(A8),end(A8),begin(cell[8+p].boundary));

//neighbor

}

//creating octree and determining its parameters

template<class type> class octree {

public:

octree() {};

void root(Cell cell[], Body body[], int n);

void strc(Cell cell[], Body body[], vector<int> &cell\_arrange, int n);

void traverse\_tree(Cell cell[], Body body[], int root,int node);

float compute\_force(Cell cell[],Body body[],int node, double force);

};

template <class type> void octree<type>::root(Cell cell[], Body body[], int n){

vector<float> CM(3,0);

for (int j=0; j<cell[n].NumNodes; ++j){

cell[n].m2=cell[n].m2+body[cell[n].members[j]].m1;

for (int i=0; i<3; ++i)

CM[i]=CM[i]+body[cell[n].members[j]].r1[i] \* body[cell[n].members[j]].m1;}

for (int i=0; i<3;++i){

cell[n].r2[i]=CM[i]/cell[n].m2;

cell[n].rd[i]=cell[n].boundary[2\*i+1]-cell[n].boundary[2\*i];}

}

template <class type> void octree<type>::strc(Cell cell[], Body body[], vector<int> &cell\_arrange, int n){

int i,j,p; octree<float> tree;

p=(cell[n].Ncell)\*8;

//extend\_cell(cell,p+1);

// cout<<"\n-----"<<n<<"\n";

det\_boundary\_subcube(cell,n,body);

tree.root(cell,body,n);

for(i=1; i<9; ++i){//

cell[i+p].level=cell[n].level+1; cell[i+p].ID2=i+p;

cell[i+p].parent2=cell[n].ID2; cell[i+p].NumNodes=0;

for(j=0; j<cell[n].NumNodes; ++j){

if(InCube(body,cell,cell[n].members[j],i+p)==true){

cell[i+p].members.push\_back(cell[n].members[j]);

cell[i+p].NumNodes=cell[i+p].NumNodes+1;

body[cell[n].members[j]].parent=i+p;}}

if(cell[i+p].NumNodes>=1 && cell[i+p].NumNodes<3){

if(cell[i+p].NumNodes==1){

cell[n].child.push\_back(cell[i+p].members[0]);

cell[i+p].child.push\_back(cell[i+p].members[0]);

body[cell[i+p].child[0]].parent=n;}

else{

cell\_arrange.push\_back(cell[i+p].ID2);

cell[n].scell.push\_back(cell[i+p].ID2);

for(j=0; j<cell[i+p].NumNodes; ++j){

cell[i+p].child.push\_back(cell[i+p].members[j]);

body[cell[i+p].child[j]].parent=i+p;}}}

if(cell[i+p].NumNodes>=3){

cell[i+p].Ncell=(int)cell\_arrange.size();

cell\_arrange.push\_back(cell[i+p].ID2);

cell[n].scell.push\_back(cell[i+p].ID2);

cell[i+p].parent2=i+p;}}

for(i=1; i<9; ++i)

if(cell[i+p].NumNodes>=3)

strc(cell,body,cell\_arrange,i+p);

}

template <class type> void octree<type>::traverse\_tree(Cell cell[], Body body[],int root,int node){

int i,theta=3; ///////Get from input params

float D=sqrt(pow((body[node].r1[0]-cell[root].r2[0]),2.0)+pow((body[node].r1[1]-cell[root].r2[1]),2.0)+pow((body[node].r1[2]-cell[root].r2[2]),2.0));

float r=sqrt(pow(cell[root].rd[0],2.0)+pow(cell[root].rd[1],2.0)+pow(cell[root].rd[2],2.0));

if (D<(r/theta)){

if (cell[root].child.size()!=0)

for(i=0; i<cell[root].child.size(); ++i)

body[node].list\_node1.push\_back(cell[root].child[i]);

if(cell[root].scell.size()!=0)

for(i=0; i<cell[root].scell.size(); ++i)

traverse\_tree(cell,body,cell[root].scell[i],node);}

else

body[node].list\_node2.push\_back(root);

}

template <class type> float octree<type>::compute\_force(Cell cell[],Body body[],int node, double force){

float G=6.673\*pow(10.0,-11.0); //////Get it from input

if(body[node].list\_node1.size()!=0)

for (int i=0; i<body[node].list\_node1.size(); ++i)

force=force+ G\* body[body[node].list\_node1[i]].m1 \* body[node].m1 / (1+(pow((body[body[node].list\_node1[i]].r1[0]-body[node].r1[0]),2.0)+pow((body[body[node].list\_node1[i]].r1[1]-body[node].r1[1]),2.0)+pow((body[body[node].list\_node1[i]].r1[2]-body[node].r1[2]),2.0)));

if(body[node].list\_node2.size()!=0)

for (int i=0; i<body[node].list\_node2.size()!=0; ++i)

force=force+ G\* cell[body[node].list\_node2[i]].m2 \* body[node].m1 / (1+(pow((cell[body[node].list\_node2[i]].r2[0]-body[node].r1[0]),2.0)+pow((cell[body[node].list\_node2[i]].r2[1]-body[node].r1[1]),2.0)+pow((cell[body[node].list\_node2[i]].r2[2]-body[node].r1[2]),2.0)));

return force;

}

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

boost::timer::cpu\_timer timer;

int i;

vector<int> cell\_arrange; octree<float> tree;

vector<int> m, list\_cell1, list\_cell2;

int N; string temp; fstream textfile;

textfile.open("ex10000.txt");

textfile>>temp;

N = atoi(temp.c\_str());

Body \*body=new Body[N];

Cell \*cell=new Cell[N\*10];

//Cell \*cell=new Cell[1];///////

read\_Input(body);

cell[0].NumNodes=N; cell[0].ID2=0; cell[0].level=0;

cell\_arrange.push\_back(0);

cell[0].Ncell=0;

for(i=0; i<N; ++i)

cell[0].members.push\_back(i);

apply\_changes(cell,0,body);

tree.strc(cell,body,cell\_arrange,0);

for(i=0;i<N; ++i)

tree.traverse\_tree(cell, body, 0, i);

for (i=0; i<N; ++i){

body[i].force=tree.compute\_force(cell,body,i, body[i].force);

}

boost::timer::cpu\_times elapsed = timer.elapsed();

std::cout << " CPU TIME: " << (elapsed.user + elapsed.system) / 1e9 << " seconds"<< " WALLCLOCK TIME: " << elapsed.wall / 1e9 << " seconds"<< std::endl;

return 0;

}