```cpp
using namespace std;

/*
#include <utility>
#include <set>
#include <memory>
#include <exception>
#include <EXCPT.H>
*/
#include <cstdio>
#include <iostream>
#include <fstream>
#include <vector>
#define _USE_MATH_DEFINES
#include <cmath>

#include "Vector3D.cpp"
#include "Charge.cpp"
#include "View3DScriptGenerator.cpp"


//_____________________________________________________________________________________________________________________
//_____________________________________________________________________________________________________________________
//
int main()
{
    vector<Charge> chargesList;
    chargesList.push_back(Vector3D{0.0D  * M_PI, 0.1D  * M_PI});
    chargesList.push_back(Vector3D{0.1D  * M_PI, 0.22D * M_PI});
    chargesList.push_back(Vector3D{0.2D  * M_PI, 0.32D * M_PI});
    chargesList.push_back(Vector3D{0.15D * M_PI, 0.39D * M_PI});
    chargesList.push_back(Vector3D{0.11D * M_PI, 0.45D * M_PI});
    chargesList.push_back(Vector3D{0.4D  * M_PI, 0.4D  * M_PI});

    chargesList.push_back(Vector3D{-0.10D  * M_PI, -0.1D  * M_PI});
    chargesList.push_back(Vector3D{-0.21D  * M_PI, -0.22D * M_PI});
    chargesList.push_back(Vector3D{-0.32D  * M_PI, -0.32D * M_PI});
    chargesList.push_back(Vector3D{-0.215D * M_PI, -0.39D * M_PI});
    chargesList.push_back(Vector3D{-0.211D * M_PI, -0.45D * M_PI});
    chargesList.push_back(Vector3D{-0.5D  * M_PI, -0.5D  * M_PI});

    View3DScriptGenerator view3DScriptWriter{chargesList};

    for(long long timeIndex = 0; timeIndex < 20000 ; ++timeIndex)
    {
        for(Charge& c: chargesList)
        {
            const Vector3D forceToApply = c.getSphereProjectedForce(chargesList);
            c.setOuterForceToApplyOn(forceToApply);
        }
        for(Charge& c: chargesList)
        {
            c.applyOuterForce();
            c.onSphereAdjustment();
        }
        if(timeIndex % 100 == 0)
            view3DScriptWriter.writeNextTimeStep(chargesList);
```

```
    }


    /*
    //Vector3D v1{1,22,9};
    //Vector3D v2{3,12,4};

    Vector3D v1{1,0,0};
    Vector3D v2{0,1,0};


    //Vector3D v1{1,2,3};
    //Vector3D v2{4,5,6};

    Vector3D v3 = v1- v2;
    //Vector3D v3 = v1.getOrthogonalPartOf(v2);
    //Vector3D v3 = (v1^v2);//.getNormalized();

    //cout << (string)v1 << endl;
    //cout << (string)v2 << endl;
    cout << (string)v3 << endl;

    */
}

//_____________________________________________________________________________________________________________________________
//_____________________________________________________________________________________________________________________________
//
```

```
class Vector3D
{
private:
    double positions[3];

public:
    Vector3D(const double x,
             const double y,
             const double z):
        positions{x,y,z}
    {}

    Vector3D():
        positions{0.0D,0.0D,0.0D}
    {}

    Vector3D(const double phi, const double theta):
        positions{::cos(phi),
                  ::sin(phi) * ::sin(theta),
                  ::sin(phi) * ::cos(theta)}
    {}

public:
    //_____________________________________________________________________________________________________________________________
    //_____________________________________________________________________________________________________________________________
    //
    double operator[](int index) const
    {
        return this->positions[index];
    }

    //_____________________________________________________________________________________________________________________________
    //_____________________________________________________________________________________________________________________________
    //
    void normalize(const double toLength = 1.0D)
    {
        const double length = this->length();

        this->positions[0] /= length * toLength;
        this->positions[1] /= length * toLength;
        this->positions[2] /= length * toLength;
    }

    //_____________________________________________________________________________________________________________________________
    //_____________________________________________________________________________________________________________________________
    //
    Vector3D getNormalized(const double toLength = 1.0D) const
    {
        Vector3D returnValue = *this;
        returnValue.normalize(toLength);

        return returnValue;
    }

    //_____________________________________________________________________________________________________________________________
    //_____________________________________________________________________________________________________________________________
    //
```

```
double length() const
{
    double  returnValue  = this->positions[0]*this->positions[0];
            returnValue += this->positions[1]*this->positions[1];
            returnValue += this->positions[2]*this->positions[2];

    return ::pow(returnValue, 0.5D);
}

//_____________________________________________________________________________________________________________________________
//_____________________________________________________________________________________________________________________________
//
double getAngle(Vector3D v2) const
{
    Vector3D v1 = this->getNormalized();
             v2 = v2.   getNormalized();

    return ::acos(v1*v2);
}

//_____________________________________________________________________________________________________________________________
//_____________________________________________________________________________________________________________________________
//
Vector3D operator+(const Vector3D& v) const
{
    return {this->positions[0] + v.positions[0],
            this->positions[1] + v.positions[1],
            this->positions[2] + v.positions[2]};
}

//_____________________________________________________________________________________________________________________________
//_____________________________________________________________________________________________________________________________
//
void operator+=(const Vector3D& v)
{
    this->positions[0] += v.positions[0];
    this->positions[1] += v.positions[1];
    this->positions[2] += v.positions[2];
}

//_____________________________________________________________________________________________________________________________
//_____________________________________________________________________________________________________________________________
//
Vector3D operator-(const Vector3D& v) const
{
    return *this + v*-1.0D;
}

//_____________________________________________________________________________________________________________________________
//_____________________________________________________________________________________________________________________________
//
double operator*(const Vector3D& v) const
{
    double returnValue  = this->positions[0] * v.positions[0];
           returnValue += this->positions[1] * v.positions[1];
           returnValue += this->positions[2] * v.positions[2];
```

```
    return returnValue;
}

//_____________________________________________________________________________________________________________________________
//_____________________________________________________________________________________________________________________________
//
Vector3D operator*(const double factor) const
{
    return {factor * this->positions[0],
            factor * this->positions[1],
            factor * this->positions[2]};
}

//_____________________________________________________________________________________________________________________________
//_____________________________________________________________________________________________________________________________
//
void operator*=(const double factor) // was buggy
{
    *this = *this * factor;

    //return *this * factor;
}

//_____________________________________________________________________________________________________________________________
//_____________________________________________________________________________________________________________________________
//
Vector3D operator/(const double divider) const
{
    return *this * (1/divider);
}

//_____________________________________________________________________________________________________________________________
//_____________________________________________________________________________________________________________________________
//
Vector3D operator^(const Vector3D& v) const //crossProduct //getNormalVector
{
    return {this->positions[1]*v.positions[2] - this->positions[2]*v.positions[1],
            this->positions[2]*v.positions[0] - this->positions[0]*v.positions[2],
            this->positions[0]*v.positions[1] - this->positions[1]*v.positions[0]};
}

//_____________________________________________________________________________________________________________________________
//_____________________________________________________________________________________________________________________________
//
Vector3D getOrthogonalPartOf(const Vector3D& v) const
{
    return (*this^v)^*this;
}

//_____________________________________________________________________________________________________________________________
//_____________________________________________________________________________________________________________________________
//
operator string() const
{
    return    "["
            + std::to_string(this->positions[0]) + ", "
            + std::to_string(this->positions[1]) + ", "
```

```
                + std::to_string(this->positions[2])
                + "]";
    }

    //_____________________________________________________________________________________________________________________________
    //_____________________________________________________________________________________________________________________________
    //
};
```

```
class Charge
{
private:
    constexpr static double resistanceFactor = 0.5D;

private:
    const double chargeStrength;
    const double onSphereWithRadius;

private:
    Vector3D    position;
    Vector3D    speed;
    Vector3D    outerForceToApply;

public:
    Charge(const Vector3D& position):
        chargeStrength      {1.0D           },
        onSphereWithRadius  {1.0D           },
        position            {position       },
        speed               {0.0D,0.0D,0.0D },
        outerForceToApply   {}
    {}

public:
    //_____________________________________________________________________________________________________________________________
    //_____________________________________________________________________________________________________________________________
    //
    Vector3D getDirectForceVectorFrom(const Charge& otherCharge) const
    {
        Vector3D returnValue = this->position - otherCharge.position;
        double        length = returnValue.length();

        return returnValue.getNormalized(otherCharge.chargeStrength) / (length*length);
    }

//_____________________________________________________________________________________________________________________________
//_____________________________________________________________________________________________________________________________
//
Vector3D getSphereProjectedForce(const Charge& otherCharge) const
{
    return this->position.getOrthogonalPartOf(this->getDirectForceVectorFrom(otherCharge));
}

//_____________________________________________________________________________________________________________________________
//_____________________________________________________________________________________________________________________________
//
Vector3D getSphereProjectedForce(const vector<Charge>& charges) const
{
    Vector3D returnValue{};
    for(const Charge& c: charges)
    {
        if(&c == this)
            continue;

        returnValue += this->getSphereProjectedForce(c);
    }
```

```
        return returnValue;
    }

    //_____________________________________________________________________________________________________________________________________
    //_____________________________________________________________________________________________________________________________________
    //
    void setOuterForceToApplyOn(const Vector3D& outerForceVector)
    {
        this->outerForceToApply = outerForceVector;
    }

    //_____________________________________________________________________________________________________________________________________
    //_____________________________________________________________________________________________________________________________________
    //
    void applyOuterForce(const double timeDelta = 0.0001)
    {
        this->speed += this->outerForceToApply;
        this->speed *= this->resistanceFactor;

        this->position += this->speed*timeDelta;
    }

    //_____________________________________________________________________________________________________________________________________
    //_____________________________________________________________________________________________________________________________________
    //
    void onSphereAdjustment()
    {
        this->position.normalize(this->onSphereWithRadius);
        const double skalarSpeed    = this->speed.length();
        this->speed                 = this->position.getOrthogonalPartOf(this->speed).getNormalized(skalarSpeed);
    }

    //_____________________________________________________________________________________________________________________________________
    //_____________________________________________________________________________________________________________________________________
    //
    Vector3D getPosition() const
    {
        return this->position;
    }

    //_____________________________________________________________________________________________________________________________________
    //_____________________________________________________________________________________________________________________________________
    //
};
```

```
class View3DScriptGenerator
{
    const string    fileName    = "charges.ms.txt";
    long long       timeIndex   = 0LL;

public:
    View3DScriptGenerator(const vector<Charge>& charges)
    {
        std::ofstream file{this->fileName, std::ofstream::out | std::ofstream::trunc};

        file << "sphere radius:1.0 pos:[0,0,0] segments:64" << endl;
        file << "sphereArray = #()" << endl << endl;

        for(auto c : charges)
        {
            file << "s = sphere radius:0.05 pos:[";
            file << c.getPosition()[0] << ", ";
            file << c.getPosition()[1] << ", ";
            file << c.getPosition()[2] << "] "  << endl;
            file << "append sphereArray s"      << endl << endl;
        }
        file.close();
    }

public:
    //_____________________________________________________________________________________________________________________________________
    //_____________________________________________________________________________________________________________________________________
    //
    void writeNextTimeStep(const vector<Charge>& charges)
    {
        std::ofstream file{this->fileName, std::ofstream::out | std::ofstream::app};

        file << endl << endl;
        file << "animate on"                                << endl;
        file << "("                                         << endl;
        file << "at time(" << this->timeIndex++ << ")"      << endl;
        file << "("                                         << endl;

        int scriptIndex = 1;
        for(auto c : charges)
        {
            file << this->getSpherePositionIndex(scriptIndex++) << (string)c.getPosition() << endl;
        }
        file << ")" << endl;
        file << ")" << endl;
        file.close();
    }

    //_____________________________________________________________________________________________________________________________________
    //_____________________________________________________________________________________________________________________________________
    //

private:
    string getSpherePositionIndex(const int index)
    {
        return "sphereArray[" + to_string(index) + "].position = ";
    }
```

```
    //_____________________________________________________________________________________________________________________________
    //_____________________________________________________________________________________________________________________________
    //
};
```