Changes made to main_Solar_HW.cpp

```
double const G = 6.674*pow(10,-11);
double const pi = 3.14159265358979323846;
```

Set conts for math later

```
ostream& operator<<(ostream& os, const Vec3d& v) {
    string output = "(" + to_string(v.x) + ", " + to_string(v.y) + ", " + to_string(v.z) + ")";
    os << output;
    return os;
}
```

Overloaded << operator for Vec3d struct, important to send entire output to OS at once for proper formatting

```
class Body {
// define the Body variable here
// name, orbit, mass (double), pos (Vec3d overload), v (Vec3d overload), a(Vec3d overload)
    // default body constructor set all variables to zero and string to "none"
    // name it Body()
    // write your code here
    string name, orbit;
    double mass;
    Vec3d pos, v, a;
    Body(){
        mass = 0;
        name = "none";
        orbit = "none";
        pos.x = 0;
        pos.y = 0;
        pos.z = 0;
        v.x = 0;
        v.y = 0;
        v.z = 0;
        a.x = 0;
        a.y = 0;
        a.z = 0;
    }
```

global variables and default constructor for Body class

```
Body(string name, string orbit, double mass, Vec3d pos, Vec3d v, Vec3d a):name(name), orbit(orbit), mass(mass), pos(pos), v(v), a(a){};
```

Simple body constructor sets class variables to what is passed through.

```
friend ostream& operator<<(ostream& os, const Body& body) {
    os << left << setw(14) << body.name
       << setw(10) << body.orbit
       << setw(15) << body.mass
       << setw(15) << body.pos
       << setw(35) << body.v
       << setw(35) << body.a << endl;
    return os;
}
```

Overload << operator for Body class, aligns left and sets width for each data type for uniform formatting and readability

```
// Todo
static auto setAccelerations(vector<Body>& bodies, int acceleration) {
    for (auto& body : bodies) {
        if(body.orbit == "Sun"){
            body.a.x += acceleration;
            body.a.y += acceleration;
        }
    }
}
// make sure that SolarSystem class is a friend to Body()
friend class SolarSystem;
```

Set accelerations function adds passed in acc value to each body that orbits the sun in the Solar System bodies vector

Also sets SolarSystem as a friend to Body

```
void stepForward(int acc){
    Body::setAccelerations(bodies, acc);
}
```

Step forward function calls setAccelerations and passes acc value to be added and bodies vector

```
class SolarSystem {
    // Solar have one variable that is vector of type Body called bodies
    // all planet Body you read should save in this vector
    vector<Body> bodies;

    // SolarSystem main function should take the location of .dat file
    // and read the complete file
    public:
    SolarSystem(string path){

    // open ifstream file
    ifstream solarfile(path);

    //check if file exist or not
    if (solarfile.is_open()){

        cout << "File is open and ready" << '\n';
        cout << " " << '\n';
        //    return; // use this for program testing to check file is open only without reading the data
    }else{
        cout << "File not found!" << '\n';
        cout << "check if path is .././src/***" << '\n';
        cout << " " << '\n';
        return;
    }
```

Creates bodies vector, as well as the beginning of the solarSystem constructor that checks if the solar system data file can be read from

```
    // define variable you want to read some info will be saved and some will be used in calculation
    // don't forget to skip first line since it contain title only
    string line, name, orbit, mass, diam, perihelion, aphelion, orbPeriod, rotationalPeriod, axialtilt, orbinclin;
    double solarMass;

    while(getline(solarfile, line)){
        Body newBody = Body();
        stringstream ss(line);
        ss >> name >> orbit >> mass >> diam >> perihelion >> aphelion >> orbPeriod >> rotationalPeriod >> axialtilt >> orbinclin;

        // Skips first label line
        if (name == "Name"){
            continue;
        }

        newBody.name = name;
        newBody.orbit = orbit;
        newBody.mass = stod(mass);

        if(name == "Sun"){
            solarMass = stod(mass);
        }
```

Variables for reading solar system data and doing initial calculations. Beginning of a while loop that runs once for each line in the file, reads the line then uses a string stream to split each column in the line into its own string. Checks for the header line so it can skip it. If it's not the header the name orbit and mass are saved into the body and if the body is the sun its mass is saved for later calculations.

```
if(orbit == "Sun"){
    double radius = (stod(perihelion)+stod(aphelion))/2;
    double orbVel = sqrt(G * (solarMass+stod(mass))/radius);
    // cout << "orbVel: " << orbVel << endl;
    // cout << "radius: " << radius << endl;

    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<> distr(0, 2 * pi);

    double ang = distr(gen);
    // cout << "ang: " << ang << endl;
    // cout << "Sin(ang): " << sin(ang) << endl;
    // cout << "Cos(ang): " << cos(ang) << endl;
    newBody.v.x = radius * cos(ang) * orbVel;
    // cout << "v.x: " << newBody.v.x << endl;
    newBody.v.y=radius*sin(ang)*orbVel;
    // cout << "v.y: " << newBody.v.y << endl;
    newBody.v.z=0;

    double orbAcc = (orbVel * orbVel)/radius;
    // ang = distr(gen);
    newBody.a.x=radius*cos(ang)*orbAcc;
    newBody.a.y=radius*sin(ang)*orbAcc;
    newBody.a.z=0;

    cout << "Body Name: " << newBody.name << endl;
    cout << "Orbit: " <<  newBody.orbit << endl;
    cout << "Orbital Velocity: " << orbVel << endl;
    cout << "Centripetal acceleration: " << orbAcc << endl;
    cout << "=============" << endl;
}
```

If the body orbits around the sun the orbital velocity and acceleration calculations are done and displayed in the console as well as saved to their respective bodies. Decided to also take into account the mass of the smaller body when calculating the orbital velocity.

```
        // srand(time(0));
        random_device rd;
        mt19937 gen(rd());
        uniform_real_distribution<> distr(0, 11);
        newBody.pos.x = distr(gen);
        newBody.pos.y = distr(gen);
        newBody.pos.z = distr(gen);
        bodies.push_back(newBody);
    }

    solarfile.close();

    }
```

Finally the position is randomly generated for all bodies, the body is added to the bodies vector, and the file is closed once the while loop is complete.

```
friend ostream& operator<<(ostream& os, const SolarSystem& solarSystem) {
    os << left << setw(14) << "Name" << setw(10) << "Orbit" << setw(15) << "Mass" << setw(15) << "Position"
    << setw(35) << "Velocity" << setw(35) << "Acceleration" << endl;
    os << string(110, '=') << endl;
    for(int i = 0; i < solarSystem.bodies.size(); i++){
        os << solarSystem.bodies.at(i)<< endl;
    }
    return os;
}
```

Overloaded << operator for SolarSystem class, creates formatted header that goes with the formatted << overload for the body class, steps through the bodies vector and prints one on each line.