PS2: N-Body Simulation

Part b: N-Body Simulation

Average time to complete assignment ~5 hours.

In **Part B,** we are adding physics simulation and animation to the program created in Part A.

- Your `CelestialBody` class should be extended with mutators so that the physics simulation can modify the velocities of each object
- You should implement a method named `step` in the class `Universe` which takes a time parameter (`double seconds`) and moves the `CelestialBody` object given its internal velocity for that much time
- You may internally represent forces, or you may calculate these from outside the Body
- Your main routine should keep track of elapsed time and terminate the simulation when time has elapsed beyond the time limit provided at the command line
- You must use smart pointers to manage the lifetimes of your `CelestialBody` objects.
- You may optionally (+.5 pt extra credit) display elapsed time on the main screen

Here are the particular assignment requirements:

- You should build a command-line app which accepts the same parameters as the one specified, and reads the universe file from `stdin`. Name your executable `NBody`, so you would run it with e.g:

```
./NBody 157788000.0 25000.0 < planets.txt
```

  - *Note*, "`157788000.0 25000.0`" are two double command-line arguments *T* and Δt
  - Reads in the universe from standard input
  - Simulates the universe, starting at time t = 0.0, and continuing as long as t < T, using the *leapfrog* scheme described below.
  - ***After the animation stops, your program should output the final state of the universe in the same format as the input.***
  - Please submit all files needed to build your project: `.cpp`'s, any header files, and a `Makefile`
  - Include the `planets.txt` file and all associated GIF images with your submission, in the proper directory structure as required by your code
  - Fill out and include this `ps2b-readme.txt` file with your work
  - Make sure to do `make clean` before tarring up your code
  - Submit your work in a directory named `ps2b`

Reading in the universe

The input format is a text file that contains the information for a particular universe (in SI units). The first value is an integer *N* which represents the number of particles. The second value is a real number *R* which represents the *radius* of the universe, used to determine the scaling of the drawing window.

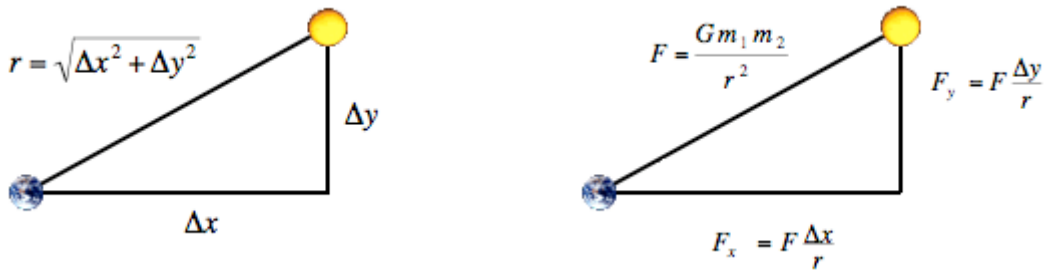
Finally, there are *N* rows, and each row contains 6 values. The first two values are the *x*- and *y*-coordinates of the initial position; the next pair of values are the *x*- and *y*-components of the initial velocity; the fifth value is the mass; the last value is a string that is the name of an image file used to display the particle.

You should read in **exactly** as many rows of body information as are indicated by *N*, the first value in the file. You may assume there are not more than 1000 planets, and that all of the filenames are less than 30 characters long.

Simulating the universe: the physics

We review the equations governing the motion of the particles, according to Newton's laws of motion and gravitation. Don't worry if your physics is a bit rusty; all of the necessary formulas are included below. We'll assume for now that the position (p_x, p_y) and velocity (v_x, v_y) of each particle is known. In order to model the dynamics of the system, we must know the net force exerted on each particle.

- **Pairwise force.** *Newton's law of universal gravitation* asserts that the strength of the gravitational force between two particles is given by the product of their masses divided by the square of the distance between them, scaled by the gravitational constant G (6.67×10^{-11} N m^2 / kg^2). The pull of one particle towards another acts on the line between them. Since we are using Cartesian coordinates to represent the position of a particle, it is convenient to break up the force into its *x*- and *y*-components (F_x, F_y) as illustrated below.



- **Net force.** The *principle of superposition* says that the net force acting on a particle in the *x*- or *y*-direction is the sum of the pairwise forces acting on the particle in that direction.
- **Acceleration.** *Newton's second law of motion* postulates that the accelerations in the *x*- and *y*-directions are given by: $a_x = F_x / m$, $a_y = F_y / m$.

Simulating the universe: the numerics

We use the *leapfrog finite difference approximation scheme* to numerically integrate the above equations: this is the basis for most astrophysical simulations of gravitational systems. In the leapfrog scheme, we discretize time, and update the time variable t in increments of the *time quantum* Δt (measured in seconds). We maintain the position (p_x, p_y) and velocity (v_x, v_y) of each particle at each time step. The steps below illustrate how to evolve the positions and velocities of the particles.

- **Step 1**. For each particle: Calculate the net force (F_x, F_y) at the current time t acting on that particle using Newton's law of gravitation and the principle of superposition. Note that force is a vector (i.e., it has direction). In particular, be aware that Δx and Δy are signed (positive or negative). In the diagram above, when you compute the force the sun exerts on the earth, the sun is pulling the earth up (Δy positive) and to the right (Δx positive).
- **Step 2**. For each particle:

a. Calculate its acceleration (a_x, a_y) at time t using the net force computed in Step 1 and Newton's second law of motion: $a_x = F_x / m$, $a_y = F_y / m$.

b. Calculate its new velocity (v_x, v_y) at the next time step by using the acceleration computed in Step 2a and the *old velocity*: assuming the acceleration remains constant in this interval, the new velocity is ($v_x + \Delta t\, a_x$, $v_y + \Delta t\, a_y$).

c. Calculate its new position (p_x, p_y) at time $t + \Delta t$ by using the *new velocity* computed in Step 2b and its *old position*: assuming the velocity remains constant in this interval, the new position is ($p_x + \Delta t\, v_x$, $p_y + \Delta t\, v_y$).

- **Step 3**. For each particle: Draw it using the position computed in Step 2.

The simulation is more accurate when Δt is very small, but this comes at the price of more computation.

The TA should be able to type the following to run your code:

```
make
./NBody 157788000.0 25000.0 < planets.txt
```

How to turn it in

Submit your work on Blackboard

The executable file that your `Makefile` builds should be called `NBody` (the grading script checks that this executable builds successfully.)

Grading rubric

Feature	Value	Comment
core implementation	12	full & correct implementation = 9 pts; nearly complete= 6pts; part way = 4 pts; started = 2 pt
		1 pt Prints the state of the universe at the end of the simulation
		1 pt planets revolve counter-clockwise
		1 pt using smart pointers
Makefile	1	Makefile included targets `all` and `clean` must exist `all` should build `NBody` must have dependencies correct
ps2b-readme.txt	2	
Total	**15**	
extra credit	+.5	display elapsed time
	+1	create new universe (and describe in readme)
	+.5	play sound file

Testing and Debugging

Below are the outputs your program should print after running on the planets.txt for various lengths of time. **Make sure you are printing your output at the very end of the program before comparing to these.** The exact numbers you get may be slightly different than the ones

below if you do your computations in a slightly different order than our solution. **This is fine**, and is caused by rounding errors that affect the results differently depending on the precise order of statements. **The biggest errors will appear to be in the sun's position.** Don't worry; all the values are printed in scientific notation. The planets' positions all end in e+10 or e+11, meaning you should multiple the numbers by 10^{10} or 10^{11}. The exponents on the sun's position are on the scale of 10^5. An error in the leading digit of the sun's position is therefore miniscule compared to the planet positions. Another way to think about this is that the window is 512 × 512 pixels, representing coordinates ranging from -2.5×10^{11} to 2.5×10^{11}. Even if the leading digit of the sun's position is off by one, the error is on the order of 10000, which is far less that one pixel in the window!

Inserting print statements is a good way to trace what your program is doing. Here are results for a few sample inputs.

```
// zero steps
5
2.50e+11
1.4960e+11  0.0000e+00  0.0000e+00  2.9800e+04  5.9740e+24    earth.gif
2.2790e+11  0.0000e+00  0.0000e+00  2.4100e+04  6.4190e+23     mars.gif
5.7900e+10  0.0000e+00  0.0000e+00  4.7900e+04  3.3020e+23  mercury.gif
0.0000e+00  0.0000e+00  0.0000e+00  0.0000e+00  1.9890e+30      sun.gif
1.0820e+11  0.0000e+00  0.0000e+00  3.5000e+04  4.8690e+24    venus.gif

// one step
5
2.50e+11
1.4960e+11  7.4500e+08 -1.4820e+02  2.9800e+04  5.9740e+24    earth.gif
2.2790e+11  6.0250e+08 -6.3860e+01  2.4100e+04  6.4190e+23     mars.gif
5.7875e+10  1.1975e+09 -9.8933e+02  4.7900e+04  3.3020e+23  mercury.gif
3.3087e+01  0.0000e+00  1.3235e-03  0.0000e+00  1.9890e+30      sun.gif
1.0819e+11  8.7500e+08 -2.8329e+02  3.5000e+04  4.8690e+24    venus.gif

// two steps
5
2.50e+11
1.4959e+11  1.4900e+09 -2.9640e+02  2.9799e+04  5.9740e+24    earth.gif
2.2790e+11  1.2050e+09 -1.2772e+02  2.4100e+04  6.4190e+23     mars.gif
5.7826e+10  2.3945e+09 -1.9789e+03  4.7880e+04  3.3020e+23  mercury.gif
9.9262e+01  2.8198e-01  2.6470e-03  1.1279e-05  1.9890e+30      sun.gif
1.0818e+11  1.7499e+09 -5.6660e+02  3.4998e+04  4.8690e+24    venus.gif

// three steps
5
2.50e+11
1.4958e+11  2.2349e+09 -4.4460e+02  2.9798e+04  5.9740e+24    earth.gif
2.2789e+11  1.8075e+09 -1.9158e+02  2.4099e+04  6.4190e+23     mars.gif
5.7752e+10  3.5905e+09 -2.9682e+03  4.7839e+04  3.3020e+23  mercury.gif
1.9852e+02  1.1280e+00  3.9705e-03  3.3841e-05  1.9890e+30      sun.gif
1.0816e+11  2.6248e+09 -8.4989e+02  3.4993e+04  4.8690e+24    venus.gif

// one year
5
2.50e+11
1.4959e+11 -1.6531e+09  3.2949e+02  2.9798e+04  5.9740e+24    earth.gif
-2.2153e+11 -4.9263e+10  5.1805e+03 -2.3640e+04  6.4190e+23     mars.gif
3.4771e+10  4.5752e+10 -3.8269e+04  2.9415e+04  3.3020e+23  mercury.gif
```

```
 5.9426e+05  6.2357e+06 -5.8569e-02  1.6285e-01  1.9890e+30      sun.gif
-7.3731e+10 -7.9391e+10  2.5433e+04 -2.3973e+04  4.8690e+24    venus.gif
```