```
//Authors: Joseph Calles and Tharith Sovann

#include "Body.hpp" // include header file

istream& operator>>(istream& input, Body& celestial_body)
{
  // get input from stdin
  input >> celestial_body._x_position >> celestial_body._y_position
        >> celestial_body._x_velocity >> celestial_body._y_velocity
        >> celestial_body._mass >> celestial_body._filename;

  // get and set object image
  if(!celestial_body._image.loadFromFile(celestial_body._filename))
    { cout << "Error: could not load sprite from file"
           << ’\’’ << celestial_body._filename << ’\’’ << endl; }
  celestial_body._texture.loadFromImage(celestial_body._image);
  celestial_body._sprite.setTexture(celestial_body._texture);

  sf::Vector2u size = celestial_body._window_size; // get data variables
  sf::Vector2u image_size = celestial_body._texture.getSize();

  double rad = *celestial_body.get_radius(); // calculate position
  double x = (celestial_body._x_position * size.x) / ( 2.f * (rad) ) + (size.x / 2.0);
  double y = (celestial_body._y_position * size.y) / ( 2.f * (rad) ) + (size.y / 2.0);

  x -= (image_size.x / 2.f); // center position over self
  y -= (image_size.y / 2.f);

  celestial_body._sprite.setPosition(x, y); // set position

  return input; // return istream
}

// extraction operator overloader for debugging and comparing position results after x steps
ostream& operator<<(ostream& out, Body& celestial_body)
{
  out << celestial_body._x_position << ’ ’
      << celestial_body._y_position << ’ ’
      << celestial_body._x_velocity << ’ ’
      << celestial_body._y_velocity << ’ ’
      << celestial_body._mass       << ’ ’
      << celestial_body._filename   << ’ ’
      << endl;

  return out;
}

void Body::set_new_position(void)
{
  sf::Vector2u size = this->_window_size; // get data variables
  sf::Vector2u image_size = this->_texture.getSize();

  double rad = *this->get_radius(); // calculate position
  double x = (this->_x_position * size.x) / ( 2.f * (rad) ) + (size.x / 2.0);
  double y = (this->_y_position * size.y) / ( 2.f * (rad) ) + (size.y / 2.0);
/*
```

```
  x -= (image_size.x / 2.f);
  y -= (image_size.y / 2.f);
*/
  this->_sprite.setOrigin((image_size.x / 2.f), (image_size.y / 2.f));//center position in middle of planet
  this->_sprite.setPosition(x, y); // set position
}

double Body::calc_x_force(Body& other_planet)//when I say total, I mean between the two bodies
{
        double delta_x = other_planet._x_position - this->_x_position;
        double delta_y = other_planet._y_position - this->_y_position;
        double total_distance = sqrt( (delta_x)*(delta_x) + (delta_y)*(delta_y) );
        double total_force = ( (*this->_big_G) * this->_mass * other_planet._mass ) / (total_distance * total_distance);
        double x_force = total_force * (delta_x / total_distance);
        return x_force;
}


double Body::calc_y_force(Body& other_planet)
{
        double delta_x = other_planet._x_position - this->_x_position;
        double delta_y = other_planet._y_position - this->_y_position;
        double total_distance = sqrt( (delta_x)*(delta_x) + (delta_y)*(delta_y) );
        double total_force = ( (*this->_big_G) * this->_mass * other_planet._mass ) / (total_distance*total_distance);

        double y_force = total_force * (delta_y / total_distance);
        return y_force;
}

void Body::step(double delta_time){

        //acceleration first
        this->_x_accel = this->_x_force / this->_mass;
        this->_y_accel = this->_y_force / this->_mass;

        //using acceleration to calculate new velocty
        this->_x_velocity -= (delta_time * _x_accel);
        this->_y_velocity -= (delta_time * _y_accel);

        //using velocity to calculate new position
        this->_x_position -= (delta_time * (_x_velocity*10));
        this->_y_position -= (delta_time * (_y_velocity*10));

        this->set_new_position();
}
```