```cpp
#include "Body.hpp"
#include <vector>

int main(int argc, char* argv[])
{
  sf::Image backdrop;

  if(!backdrop.loadFromFile("starfield.jpg"))
    {
      return -1;
    }

  //Set up the universe
  std::vector<Body*> v_bodies;
  int number_of_bodies;
  double universe_size;
  sf::Vector2u size = backdrop.getSize();

  std::cin >> number_of_bodies;
  std::cin >> universe_size;


  //Create the celestial bodies and put them into the vector
  for(int i = 0; i < number_of_bodies; i++)
    {
      Body* body = new Body();
      v_bodies.push_back(body);
    }

  for(int i = 0; i < number_of_bodies; i++)
    {
      std::cin >> (*(v_bodies.at(i)));
      v_bodies.at(i)->set_radius(universe_size);
      v_bodies.at(i)->set_window(size.x);
      v_bodies.at(i)->update_pixel_pos();
    }

  sf::Texture texture_drop;
  texture_drop.loadFromImage(backdrop);

  sf::Sprite sprite_drop;
  sprite_drop.setTexture(texture_drop);


  sf::RenderWindow window(sf::VideoMode(size.x, size.y),
                            "NBody Program");

  while(window.isOpen())
    {
      sf::Event event;
      while(window.pollEvent(event))
        {
          if(event.type == sf::Event::Closed)
            {
              window.close();
            }
        }

      window.clear();
      window.draw(sprite_drop);
      for(int i = 0; i < number_of_bodies; i++)
```

```
        {
          window.draw(*(v_bodies.at(i)));
        }
      window.display();
    }

  return 0;
}
```

```cpp
#include "Body.hpp"
#include <cstdlib>

Body::Body()
{

}

Body::Body(double x, double y, double vel_x, double vel_y,
           double user_mass, std::string u_filename)
{
  //initialize variables
  xpos = x;
  ypos = y;
  velocity_x = vel_x;
  velocity_y = vel_y;
  mass = user_mass;
  filename = u_filename;

  //Load image
  sf::Image image;
  if(!image.loadFromFile(filename))
    {
      exit(1);
    }

  texture.loadFromImage(image);
  sprite.setTexture(texture);
}

Body::~Body()
{


}

double Body::get_xpos()
{
  return xpos;
}

double Body::get_ypos()
{
  return ypos;
}

void Body::set_radius(double rad)
{
  univ_rad = rad;
}

void Body::set_window(int size)
{
  window_size = size;
}

void Body::update_pixel_pos()
{
  double percent_x, percent_y;

  percent_x = (xpos + univ_rad) / (2 * univ_rad);
  percent_y = (ypos + univ_rad) / (2 * univ_rad);

  sprite.setPosition((window_size * percent_x),
                     ((window_size * percent_y)));
}

std::istream& operator >> (std::istream &input, Body &B)
{
  input >> B.xpos >> B.ypos >> B.velocity_x >> B.velocity_y >>
    B.mass >> B.filename;

  //Load image
  sf::Image image;
  if(!image.loadFromFile(B.filename))
    {
      exit(1);
    }

  B.texture.loadFromImage(image);
  B.sprite.setTexture(B.texture);



  return input;
}

void Body::draw(sf::RenderTarget &target, sf::RenderStates states) const
{
  target.draw(sprite, states);
}
```

```cpp
#ifndef BODY_HPP
#define BODY_HPP

#include <iostream>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <SFML/System.hpp>
#include <string>

class Body : public sf::Drawable
{
public:

  Body();
  Body(double x, double y, double vel_x, double vel_y, double user_mass,
       std::string u_filename);
  ~Body();

  //Accessors
  double get_xpos();
  double get_ypos();

  //Mutators
  void set_radius(double rad);
  void set_window(int size);

  void update_pixel_pos();

  friend std::istream& operator >> (std::istream &input, Body &B);

private:

  virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const
;

  double xpos, ypos, velocity_x, velocity_y, mass, univ_rad;
  sf::Texture texture;
  sf::Sprite sprite;
  int window_size;
  std::string filename;

};



#endif
```

```
C=g++ -g -Wall --std=c++98 -Werror
E=.cpp
O=main.o Body.o
P=NBody
SFML= -lsfml-graphics -lsfml-window -lsfml-system
all: $(P)
$(P):$(O)
	$(C) -o $(P) $(O) $(SFML)

$(E).o:
	$(C) -c $<  -o $@

clean:
	rm $(O) $(P)
```