```
 1: //Steven (Canhua) Tan
 2: #include <SFML/Graphics.hpp>
 3: #include <SFML/Window.hpp>
 4:
 5: int main()
 6: {
 7:     sf::RenderWindow window(sf::VideoMode(800, 600), "SFML works!");
 8:     window.setVerticalSyncEnabled(true);
 9:     // Load a sprite to display
10:     sf::Texture texture;
11:     if (!texture.loadFromFile("sprite.png"))
12:         return EXIT_FAILURE;
13:     sf::Sprite sprite(texture);
14:
15:     sprite.setPosition(0,300);
16:
17:     while (window.isOpen())
18:     {
19:         sf::Event event;
20:
21:         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)){
22:             sprite.rotate(10.f);
23:         }
24:         while (window.pollEvent(event))
25:         {
26:             if (event.type == sf::Event::Closed)
27:                 window.close();
28:             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)){
29:                 sprite.setPosition(-100.f, 300);
30:             }
31:             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)){
32:                 sprite.setPosition(200.f, 300);
33:             }
34:         }
35:         window.clear();
36:         window.draw(sprite);
37:         window.display();
38:     }
39:
40:     return 0;
41: }
```

```
 1: CC = g++
 2: CFLAGS = -std=c++11 -c -g -Og -Wall -Werror -pedantic
 3: LIBS =-lboost_unit_test_framework
 4: EXE = boosttest
 5:
 6: all: ps1a
 7:         ./ps1a
 8:         $(CC) $(OBJ) -o $(EXE) $(LIBS)
 9:
10: ps1a:   test.o FibLFSR.o
11:         $(CC) $^ -o $@ $(LIBS)
12:
13: test.o: test.cpp FibLFSR.h
14:         $(CC) $(CFLAGS) -c $^ $(LIBS)
15:
16: FibLFSR.o:      FibLFSR.cpp FibLFSR.h
17:         $(CC) $(CFLAGS) -c $<
18:
19: clean:
20:         rm *.o
21:         rm ps1a
```

```
 1: #include <string>
 2: #include <iostream>
 3: #include "FibLFSR.h"
 4: using namespace std;
 5:
 6: // constructor to create LFSR with the given initial seed and tap.
 7: FibLFSR::FibLFSR(string seed) {this->seed=seed;}
 8:
 9: // simulate one step and return the new bit as 0 or 1.
10: int FibLFSR::step () {
11:     char first;
12:     first = this->seed[0];
13:
14:     char pos1, pos2, pos3;
15:     pos1 = seed[tab13];
16:     pos2 = seed[tab12];
17:     pos3 = seed[tab10];
18:
19:     int res1, res2, res3;
20:     res1 = XOR(first, pos1);
21:     res2 = XOR(res1, pos2);
22:     res3 = XOR(res2, pos3);
23:
24:     this->seed.erase(0,1);
25:     this->seed.push_back(res3);
26:
27:     if (res3 == '0'){
28:       return 0;
29:     }
30:     else{
31:       return 1;
32:     }
33:   }
34:
35: // simulate k steps and return k-bits integer.
36: int FibLFSR::generate (int k)
37:   {
38:     int i;
39:     int val=0;
40:     for(i=0;i<k;i++){
41:       val = val*2 + this->step();
42:     }
43:     return val;
44:   }
45:
46: char FibLFSR::XOR (char first, char second){
47:    if(first=='0' && second=='0'){
48:      return '0';
49:    }
50:    else if(first=='0' && second=='1'){
51:      return '1';
52:    }
53:    else if(first=='1' && second=='0'){
54:      return '1';
55:    }
56:    else{
57:      return '0';
58:    }
59:  }
60:
61: std::ostream& operator<< (std::ostream &out, const FibLFSR &lfsr){
```

```
   62:   out<< lfsr.seed;
   63:   return out;
   64: }
   65:
```

```
 1: #include <string>
 2: #include <iostream>
 3: using namespace std;
 4:
 5: class FibLFSR {
 6: public:
 7:   FibLFSR(string seed);
 8:   // constructor to create LFSR with the given initial seed and tap.
 9:   int step ();
10:   // simulate one step and return the new bit as 0 or 1.
11:   int generate (int k);
12:   // simulate k steps and return k-bits integer.
13:   friend std::ostream& operator<< (std::ostream &out, const FibLFSR &lfsr);
14: private:
15:   int tab13 = 2;
16:   int tab12 = 3;
17:   int tab10 = 5;
18:   string seed;
19:   char XOR (char first, char second);
20: };
```

```
 1: #include "FibLFSR.cpp"
 2: using namespace std;
 3:
 4: int main(){
 5:   int i;
 6:   FibLFSR gen ("1100011011000011");
 7:   for(i=0;i<10;i++){
 8:     int j;
 9:     j = gen.generate(5);
10:     cout<< gen << ' ' << j << endl;
11:   }
12: }
```

```
 1: // Dr. Rykalova
 2: // test.cpp for PS1a
 3: // updated 1/31/2020
 4:
 5: #include <iostream>
 6: #include <string>
 7:
 8: #include "FibLFSR.h"
 9:
10: #define BOOST_TEST_DYN_LINK
11: #define BOOST_TEST_MODULE Main
12: #include <boost/test/unit_test.hpp>
13:
14: BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {
15:
16:   FibLFSR l("1011011000110110");
17:   BOOST_REQUIRE(l.step() == 0);
18:   BOOST_REQUIRE(l.step() == 0);
19:   BOOST_REQUIRE(l.step() == 0);
20:   BOOST_REQUIRE(l.step() == 1);
21:   BOOST_REQUIRE(l.step() == 1);
22:   BOOST_REQUIRE(l.step() == 0);
23:   BOOST_REQUIRE(l.step() == 0);
24:   BOOST_REQUIRE(l.step() == 1);
25:
26:   FibLFSR l2("1011011000110110");
27:   BOOST_REQUIRE(l2.generate(9) == 51);
28: }
29:
30: BOOST_AUTO_TEST_CASE (test2)
31: {
32:   FibLFSR lfsr("1101101101000010");
33:   BOOST_CHECK_EQUAL(lfsr.generate(4),0);
34: }
35:
36: BOOST_AUTO_TEST_CASE (test3)
37: {
38:   FibLFSR lfsr("0110101111000000");
39:   BOOST_CHECK_EQUAL(lfsr.step(),1);
40: }
```

```
 1: CC = g++
 2: CFLAGS =-g -c -Werror -Wall -pedantic -ansi -std=c++11
 3: LIBS = -lsfml-graphics -lsfml-window -lsfml-system
 4: SOURCES =FibLFSR.cpp PhotoMagic.cpp
 5: OBJECTS =$(SOURCES:.cpp=.o)
 6: EXE =PhotoMagic
 7:
 8: all:    $(SOURCES) $(EXE)
 9:
10: $(EXE): $(OBJECTS)
11:         $(CC) $(OBJECTS) -o $@ $(LIBS)
12:
13: .cpp.o:
14:         $(CC) $(CFLAGS) $< -o $@
15:
16: clean:
17:         rm $(EXE) $(OBJ)
```

```
 1: #include <string>
 2: #include <iostream>
 3: #include "FibLFSR.h"
 4: using namespace std;
 5:
 6: // constructor to create LFSR with the given initial seed and tap.
 7: FibLFSR::FibLFSR(string seed) {this->seed=seed;}
 8:
 9: // simulate one step and return the new bit as 0 or 1.
10: int FibLFSR::step () {
11:     char first;
12:     first = this->seed[0];
13:
14:     char pos1, pos2, pos3;
15:     pos1 = seed[tab13];
16:     pos2 = seed[tab12];
17:     pos3 = seed[tab10];
18:
19:     int res1, res2, res3;
20:     res1 = XOR(first, pos1);
21:     res2 = XOR(res1, pos2);
22:     res3 = XOR(res2, pos3);
23:
24:     this->seed.erase(0,1);
25:     this->seed.push_back(res3);
26:
27:     if (res3 == '0'){
28:       return 0;
29:     }
30:     else{
31:       return 1;
32:     }
33:   }
34:
35: // simulate k steps and return k-bits integer.
36: int FibLFSR::generate (int k)
37:   {
38:     int i;
39:     int val=0;
40:     for(i=0;i<k;i++){
41:       val = val*2 + this->step();
42:     }
43:     return val;
44:   }
45:
46: char FibLFSR::XOR (char first, char second){
47:    if(first=='0' && second=='0'){
48:      return '0';
49:    }
50:    else if(first=='0' && second=='1'){
51:      return '1';
52:    }
53:    else if(first=='1' && second=='0'){
54:      return '1';
55:    }
56:    else{
57:      return '0';
58:    }
59:  }
60:
61: std::ostream& operator<< (std::ostream &out, const FibLFSR &lfsr){
```

```
   62:   out<< lfsr.seed;
   63:   return out;
   64: }
   65:
```

```
 1: #include <string>
 2: #include <iostream>
 3: using namespace std;
 4:
 5: class FibLFSR {
 6: public:
 7:   FibLFSR(string seed);
 8:   // constructor to create LFSR with the given initial seed and tap.
 9:   int step ();
10:   // simulate one step and return the new bit as 0 or 1.
11:   int generate (int k);
12:   // simulate k steps and return k-bits integer.
13:   friend std::ostream& operator<< (std::ostream &out, const FibLFSR &lfsr);
14: private:
15:   int tab13 = 2;
16:   int tab12 = 3;
17:   int tab10 = 5;
18:   string seed;
19:   char XOR (char first, char second);
20: };
```

```
 1: // pixels.cpp:
 2: // using SFML to load a file, manipulate its pixels, write it to disk
 3:
 4:
 5: // g++ -o pixels pixels.cpp -lsfml-graphics -lsfml-window
 6: #include <iostream>
 7: #include <string>
 8: #include <sstream>
 9: #include <SFML/System.hpp>
10: #include <SFML/Window.hpp>
11: #include <SFML/Graphics.hpp>
12: #include "FibLFSR.h"
13: void transform (sf::Image& , FibLFSR*);
14:
15: using namespace std;
16: int main(int argc, char* argv[])
17: {
18:         string in_name(argv[1]);
19:         string out_name(argv[2]);
20:         string seed(argv[3]);
21:
22:         FibLFSR scramble(seed);
23:
24:         sf::Image image;
25:         if (!image.loadFromFile(in_name)){
26:                 return -1;
27:         }
28:
29:         sf::Image output_image;
30:         if (!image.loadFromFile(out_name)){
31:                 return -1;
32:         }
33:
34:         sf::Image outImage = image;
35:         // creates 2 separate windows for input and output images
36:         sf::Vector2u size = image.getSize();
37:         sf::RenderWindow window(sf::VideoMode(size.x, size.y), "input");
38:         sf::RenderWindow outWindow(sf::VideoMode(size.x, size.y), "output");
39:
40:         transform (outImage, &scramble);
41:
42:         sf::Texture texture, outTex;
43:         texture.loadFromImage(image);
44:         outTex.loadFromImage(outImage);
45:
46:         sf::Sprite sprite, outSprite;
47:         sprite.setTexture(texture);
48:         outSprite.setTexture(outTex);
49:
50:         while (window.isOpen() && outWindow.isOpen())
51:         {
52:                 sf::Event event;
53:                 while (window.pollEvent(event))
54:                 {
55:                         if (event.type == sf::Event::Closed)
56:                                 window.close();
57:                 }
58:
59:                 while (outWindow.pollEvent(event))
60:                 {
61:                         if (event.type == sf::Event::Closed)
```

```
62:                                     outWindow.close();
63:                 }
64:
65:                 window.clear(sf::Color::White);
66:                 window.draw(sprite);
67:                 window.display();
68:
69:                 outWindow.clear(sf::Color::White);
70:                 outWindow.draw(outSprite);
71:                 outWindow.display();
72:         }
73:
74:         // fredm: saving a PNG segfaults for me, though it does properly
75:         //   write the file
76:         if (!outImage.saveToFile(out_name))
77:                 return -1;
78:
79:         return 0;
80: }
81:
82: void transform(sf::Image& outImage, FibLFSR* scramble){
83:         // p is a pixelimage.getPixel(x, y);
84:         FibLFSR reorder = *scramble;
85:
86:         sf::Color p;
87:         // create photographic negative image of upper-left 200 px square
88:         sf::Vector2u size = outImage.getSize();
89:         for(int x= 0; x < (signed)size.x; x++){
90:                 for(int y = 0; y < (signed)size.y; y++){
91:                         p = outImage.getPixel(x,y);
92:                         p.r = p.r ^ reorder.generate(8);
93:                         p.g = p.g ^ reorder.generate(8);
94:                         p.b = p.b ^ reorder.generate(8);
95:                         outImage.setPixel(x, y, p);
96:                 }
97:         }
98: }
```

```
 1: CC = g++
 2: CFLAGS =-g -c -Werror -Wall -pedantic -ansi -std=c++11
 3: LIBS = -lsfml-graphics -lsfml-window -lsfml-system
 4: SOURCES =main.cpp ps2a.cpp
 5: OBJECTS = $(SOURCES:.cpp=.o)
 6: EXE = NBody
 7:
 8: all: $(SOURCES) $(EXE)
 9:
10: $(EXE): $(OBJECTS)
11:         $(CC) $(OBJECTS) -o $@ $(LIBS)
12:
13: .cpp.o:
14:         $(CC) $(CFLAGS) $< -o $@
15:
16: clean:
17:         rm $(EXE) $(OBJECTS)
```

```
 1: #include <iostream>
 2: #include <sstream>
 3: #include <iomanip>
 4: #include <cstdlib>
 5: #include <vector>
 6: #include <cmath>
 7: #include <cstring>
 8: #include <SFML/Graphics.hpp>
 9: #include <SFML/Window.hpp>
10: #include <SFML/System.hpp>
11: #include "ps2a.hpp"
12:
13: using namespace std;
14:
15: body::body(){}
16: body::~body(){}
17: //setting the size of the universe and window
18: void body::setWindowSize(int size){
19:     windowSize = size;
20: }
21: void body::setUniSize(double size){
22:     uniSize = size;
23: }
24: //getter funcitons
25: double body::getx(){
26:     return xpos;
27: }
28: double body::gety(){
29:     return ypos;
30: }
31: //sets the location of each sprite according to the window size
32: void body::set_position(){
33:     sf::Sprite sprite_temep = sprite;
34:     xpos = ((xpos/xradius) * (window_width/2) + (window_width/2));
35:     ypos = ((ypos/xradius) * (window_width/2) + (window_height/2));
36:     sprite.setPosition(xpos, ypos);
37: }
38:
39: void body::setRadius(float radius){
40:     xradius = radius;
41:     return;
42: }
43:
44: void body::draw(sf::RenderTarget& target, sf::RenderStates states)const{
45:     target.draw(sprite);
46: }
47: //input and output operator overload
48: istream &operator >> (istream &input, body &spaceObj){
49:     input >> spaceObj.xpos;
50:     input >> spaceObj.ypos;
51:     input >> spaceObj.xvel;
52:     input >> spaceObj.yvel;
53:     input >> spaceObj.mass;
54:     input >> spaceObj.filename;
55:     spaceObj.texture.loadFromFile(spaceObj.filename);
56:     spaceObj.sprite.setTexture(spaceObj.texture);
57:     return input;
58: }
59:
60: ostream &operator<<(ostream &output, body &spaceObj){
61:     output << setw(14) << spaceObj.xpos;
```

```
   62:     output << setw(14) << spaceObj.ypos;
   63:     output << setw(14) << spaceObj.xvel;
   64:     output << setw(14) << spaceObj.yvel;
   65:     output << setw(14) << spaceObj.mass;
   66:     output << setw(14) << spaceObj.filename;
   67:     return output;
   68: }
```

```
 1: #ifndef PS2a_HPP
 2: #define PS2a_HPP
 3:
 4: #include <SFML/Window.hpp>
 5: #include <iostream>
 6: #include <cstring>
 7: #include <vector>
 8: #include <memory>
 9: #include <string>
10: #include <SFML/Graphics.hpp>
11:
12: const int window_height = 500;
13: const int window_width = 500;
14:
15: class body: public sf::Drawable{
16: public:
17:     body();
18:     ~body();
19:
20:     void setWindowSize(int size);
21:     void setUniSize(double size);
22:
23:     double getx();
24:     double gety();
25:     double getMass();
26:
27:     void xVel(double newVel);
28:     void yVel(double newVel);
29:
30:     void setRadius(float radius);
31:     void set_position();
32:
33:     void draw(sf::RenderTarget& target, sf::RenderStates states) const;
34:
35:     friend std::istream &operator >> (std::istream &input, body &spaceObj);
36:     friend std::ostream &operator << (std::ostream &output, body &spaceObj);
37:
38: private:
39:     double xpos, ypos;
40:     double xvel, yvel;
41:     double mass;
42:     double uniSize;
43:     int windowSize;
44:     double xradius;
45:     std::string filename;
46:     sf::Sprite sprite;
47:     sf::Texture texture;
48: };
49:
50: class Universe{
51: public:
52:     Universe(){}
53:     Universe(int r):radius(r){}
54:     ~Universe(){}
55:
56: private:
57:     double radius = 0.0;
58: };
59: #endif
```

```
    1: #include <iostream>
    2: #include <sstream>
    3: #include <cstdlib>
    4: #include <vector>
    5: #include <cmath>
    6: #include <cstring>
    7: #include "ps2a.hpp"
    8:
    9: #include <SFML/Graphics.hpp>
   10: #include <SFML/Window.hpp>
   11: #include <SFML/System.hpp>
   12:
   13: using namespace std;
   14:
   15: int main(int argc, char* argv[]){
   16:     string planetCount, radius;
   17:     //takes in the information of the planets
   18:     cin >> planetCount;
   19:     cin >> radius;
   20:     //Converts the planets' information to int and stores them to a vector t
o be displayed
   21:     int numPlanets = atoi(planetCount.c_str());
   22:     float universeRadius = atof(radius.c_str());
   23:     vector <body> uniVector;
   24:     for(int i = 0; i < numPlanets; i++){
   25:         body* temp = new body();
   26:         cin >> *temp;
   27:         temp -> setRadius(universeRadius);
   28:         temp -> set_position();
   29:         uniVector.push_back(*temp);
   30:     }
   31:     sf::RenderWindow window(sf::VideoMode(window_width, window_height), "Sol
ar System");
   32:
   33:     sf::Image bgImage;
   34:     bgImage.loadFromFile("starfield.jpg");
   35:
   36:     sf::Texture bgTexture;
   37:     bgTexture.loadFromImage(bgImage);
   38:
   39:     sf::Sprite bgSprite;
   40:     bgSprite.setTexture(bgTexture);
   41:
   42:     //displays the items in the vector
   43:     while (window.isOpen()){
   44:         sf::Event event;
   45:         while(window.pollEvent(event)){
   46:             if(event.type == sf::Event::Closed){
   47:                 window.close();
   48:             }
   49:         }
   50:         window.clear();
   51:
   52:         window.draw(bgSprite);
   53:         vector<body>::iterator itr;
   54:         for (itr = uniVector.begin(); itr != uniVector.end(); ++itr){
   55:             window.draw(*itr);
   56:         }
   57:         window.display();
   58:     }
   59:     return 0;
```

```
60: }
```

```
 1: CC = g++
 2: CFLAGS =-g -c -Werror -Wall -pedantic -ansi -std=c++11
 3: LIBS = -lsfml-graphics -lsfml-window -lsfml-system
 4: SOURCES =main.cpp ps2b.cpp
 5: OBJECTS = $(SOURCES:.cpp=.o)
 6: EXE = NBody
 7:
 8: all: $(SOURCES) $(EXE)
 9:
10: $(EXE): $(OBJECTS)
11:         $(CC) $(OBJECTS) -o $@ $(LIBS)
12:
13: .cpp.o:
14:         $(CC) $(CFLAGS) $< -o $@
15:
16: clean:
17:         rm $(EXE) $(OBJECTS)
```

```
 1: #include <iostream>
 2: #include <sstream>
 3: #include <iomanip>
 4: #include <cstdlib>
 5: #include <vector>
 6: #include <cmath>
 7: #include <cstring>
 8: #include <SFML/Graphics.hpp>
 9: #include <SFML/Window.hpp>
10: #include <SFML/System.hpp>
11: #include "ps2b.hpp"
12:
13: using namespace std;
14:
15: body::body() {
16:     xpos = 0;
17:     ypos = 0;
18:     xvel = 0;
19:     yvel = 0;
20:     mass = 0;
21: }
22:
23: body::body(int _windowSize, double _uniSize) {
24:     xpos = 0;
25:     ypos = 0;
26:     xvel = 0;
27:     yvel = 0;
28:     mass = 0;
29:     windowSize = _windowSize;
30:     uniSize = _uniSize;
31: }
32:
33: body::~body() { }
34: //Setters
35: void body::setWindowSize(int _size){
36:     windowSize = _size;
37: }
38:
39: void body::setUniSize(double _size){
40:     uniSize = _size;
41: }
42: //Getters
43: double body::getx(){
44:     return xpos;
45: }
46:
47: double body::gety(){
48:     return ypos;
49: }
50:
51: double body::getMass(){
52:     return mass;
53: }
54:
55: void body::xVel(double newVel){
56:     xvel -= newVel;
57: }
58:
59: void body::yVel(double newVel){
60:     yvel -= newVel;
61: }
```

```
   62: //step function to change the x y positions of each increment of time
   63: void body::step(double seconds){
   64:     xpos = xpos - seconds * xvel;
   65:     ypos = ypos - seconds * yvel;
   66: }
   67:
   68: void body::draw(sf::RenderTarget& target, sf::RenderStates states) const{
   69:     sf::Sprite sprite_temp = sprite;
   70:     double ratio = (windowSize / 2) / uniSize;
   71:     double rxpos = xpos * ratio + (windowSize / 2);
   72:     double rypos = ypos * ratio + (windowSize / 2);
   73:     sprite_temp.setPosition(rxpos, rypos);
   74:     target.draw(sprite_temp);
   75: }
   76: //Overload funcitons
   77: istream &operator>>(istream &input, body &arg){
   78:     input >> arg.xpos;
   79:     input >> arg.ypos;
   80:     input >> arg.xvel;
   81:     input >> arg.yvel;
   82:     input >> arg.mass;
   83:     input >> arg.filename;
   84:
   85:     arg.texture.loadFromFile(arg.filename);
   86:     arg.sprite.setTexture(arg.texture);
   87:
   88:     return input;
   89: }
   90:
   91: ostream &operator<<(ostream &output, body &arg){
   92:     output << setw(14) << arg.xpos;
   93:     output << setw(14) << arg.ypos;
   94:     output << setw(14) << arg.xvel;
   95:     output << setw(14) << arg.yvel;
   96:     output << setw(14) << arg.mass;
   97:     output << setw(14) << arg.filename;
   98:
   99:     return output;
  100: }
```

```
 1: #ifndef PS2a_HPP
 2: #define PS2a_HPP
 3:
 4: #include <SFML/Window.hpp>
 5: #include <iostream>
 6: #include <cstring>
 7: #include <vector>
 8: #include <memory>
 9: #include <string>
10: #include <SFML/Graphics.hpp>
11: #include <SFML/System.hpp>
12:
13: const int window_height = 500;
14: const int window_width = 500;
15:
16: class body: public sf::Drawable{
17: public:
18:     body();
19:     body(int windowSize, double uniSize);
20:     ~body();
21:
22:     void setWindowSize(int size);
23:     void setUniSize(double size);
24:
25:     double getx();
26:     double gety();
27:     double getMass();
28:
29:     void xVel(double newVel);
30:     void yVel(double newVel);
31:
32:     void setRadius(float radius);
33:     void set_position();
34:     void step(double second);
35:     void draw(sf::RenderTarget& target, sf::RenderStates states) const;
36:
37:     friend std::istream &operator >> (std::istream &input, body &spaceObj);
38:     friend std::ostream &operator << (std::ostream &output, body &spaceObj);
39:
40: private:
41:     double xpos, ypos;
42:     double xvel, yvel;
43:     double mass;
44:     double uniSize;
45:     int windowSize;
46:     double xradius;
47:     std::string filename;
48:     sf::Sprite sprite;
49:     sf::Texture texture;
50: };
51:
52: class Universe{
53: public:
54:     Universe(){}
55:     Universe(int r):radius(r){}
56:     ~Universe(){}
57:
58: private:
59:     double radius = 0.0;
60: };
61: #endif
```

```
 1: #include <iostream>
 2: #include <sstream>
 3: #include <iomanip>
 4: #include <cstdlib>
 5: #include <vector>
 6: #include <cmath>
 7: #include <cstring>
 8: #include <SFML/Graphics.hpp>
 9: #include <SFML/Window.hpp>
10: #include <SFML/System.hpp>
11: #include "ps2b.hpp"
12:
13: using namespace std;
14:
15: int main(int argc, char* argv[]){
16:     int windowSize = 500;
17:
18:     double G = 6.67e-11;
19:     double uniSize;
20:
21:     string input;
22:     int num_planets;
23: //Making sure the amount of arguments are correct
24:     if(argc < 3){
25:         cout << "Too few arguments" << endl;
26:         return 1;
27:     }
28:     else if(argc > 3){
29:         cout << "Too many arguments" << endl;
30:     }
31: //Variables to keep track of time
32:     double timeTot = (atof(argv[1]));
33:     double timeStep = (atof(argv[2]));
34:     double timePassed = 0.0;
35:
36:     sf::Font time_font;
37:     time_font.loadFromFile("arial.ttf");
38:
39:     sf::Text timeText;
40:     timeText.setFont(time_font);
41:     timeText.setCharacterSize(20);
42:
43:     sf::Image bgImage;;
44:     bgImage.loadFromFile("starfield.jpg");
45:
46:     sf::Texture bgTexture;
47:     bgTexture.loadFromImage(bgImage);
48:
49:     sf::Sprite sprite(bgTexture);
50:
51:     getline(cin, input);
52:     num_planets = atoi(input.c_str());
53:
54:     getline(cin, input);
55:     uniSize = atof(input.c_str());
56:
57:     body temp(windowSize, uniSize);
58:
59:     vector<body> newUni;
60:     for(int i = 0; i < num_planets; i++){
61:         newUni.push_back(temp);
```

```
    }

    for(int i = 0; i < num_planets; i++){
        getline(cin, input);
        istringstream iss(input);
        iss >> newUni[i];
    }

    sf::RenderWindow window(sf::VideoMode(windowSize, windowSize), "The Sola
r System");
    while(window.isOpen()){
        sf::Event event;
        while(window.pollEvent(event)){
            if(event.type == sf::Event::Closed){
                window.close();
            }
        }

        vector<double> forceX;
        vector<double> forceY;
        for(int i = 0; i < num_planets; i++){
            forceX.push_back(0.0);
            forceY.push_back(0.0);
        }
//Calculating forces
        for (unsigned i = 0; i < newUni.size(); i++){
            for(unsigned j = 0; j < newUni.size(); j++){
                if(i == j){
                    forceX[i] = forceX[i];
                    forceY[i] = forceY[i];
                }
                else{
                    double dX = newUni[j].getx() - newUni[i].getx();
                    double dY = newUni[j].gety() - newUni[i].gety();
                    double r = sqrt(dX * dX + dY * dY);
                    double F = (G * newUni[i].getMass() * newUni[j].getMass(
)) / (r * r);
                    double Fx = F * (dX / r);
                    double Fy = F * (dY / r);
                    forceY[i] += Fy;
                    forceX[i] += Fx;
                }
            }
        }

        for(unsigned i = 0; i < newUni.size(); i++){
            double Ax = forceX[i] / newUni[i].getMass();
            double Ay = forceY[i] / newUni[i].getMass();
            newUni[i].yVel(timeStep * Ay);
            newUni[i].xVel(timeStep * Ax);
            newUni[i].step(timeStep);
        }
        window.clear();
        window.draw(sprite);
//Draws positions of planets after increments of time
        for(unsigned i = 0; i < newUni.size(); ++i){
            window.draw(newUni[i]);
        }
        timePassed = timePassed + timeStep;
        if (timePassed > timeTot)
        window.close();
```

```
121:
122:         stringstream ss;
123:         ss << timePassed;
124:         timeText.setString(ss.str());
125:         window.draw(timeText);
126:         window.display();
127:     }
128:     for(unsigned i = 0; i < newUni.size(); i++){
129:         cout << newUni[i] << endl;
130:     }
131: }
```

```
 1: CC= g++
 2: #CFLAGS= -Wall -Werror -ansi -pedantic
 3: SFMLFLAGS= -lsfml-graphics -lsfml-window -lsfml-system
 4:
 5: all: Tree
 6:         make clean
 7:
 8: Tree: TFractal.o
 9:         $(CC) TFractal.o -o TFractal $(SFMLFLAGS)
10:
11: TFractal.o: TFractal.cpp TFractal.hpp
12:         $(CC) -c TFractal.cpp $(SFMLFLAGS)
13:
14: clean:
15:         rm *.o
```

```
    1: //including library
    2: #include <cmath>
    3: #include <cassert>
    4: #include <iostream>
    5: #include <fstream>
    6: #include <vector>
    7: #include <array>
    8: #include <algorithm>
    9: #include <SFML/System.hpp>
   10: #include <SFML/Window.hpp>
   11: #include <SFML/Graphics.hpp>
   12: #include <sstream>
   13: #include "TFractal.hpp"
   14:
   15: //namespace (not for sfml)
   16: using namespace std;
   17:
   18: //class constructor
   19: FTree::FTree(int length, int depth, int angle){
   20:     //get depth
   21:     this->depth = depth;
   22:     this->length = length;
   23:     triangle.setPointCount(3);
   24:     triangle.rotate(angle);
   25:     triangle.setRadius(length);
   26:     triangle.setPosition(250, 250);
   27:     triangle.setFillColor(sf::Color::Transparent);
   28:     triangle.setOutlineColor(sf::Color::Cyan);
   29:     triangle.setOutlineThickness(1);
   30: }
   31: //recursive function
   32: void FTree::fTree(const sf::CircleShape triangle, int depth, int angle, sf::
RenderTarget& point) const{
   33:     if (depth <= 0){
   34:         return;
   35:     }
   36:     auto tf = triangle.getTransform();
   37:     sf::CircleShape lTriangle (triangle.getRadius() / 2, 3);
   38:     lTriangle.setFillColor(sf::Color::Transparent);
   39:     lTriangle.setOutlineColor(sf::Color::Cyan);
   40:     lTriangle.setOutlineThickness(1);
   41:
   42:     lTriangle.setPosition(tf.transformPoint({lTriangle.getRadius() * 3 - lTr
iangle.getRadius() / 4, lTriangle.getRadius() * 3}));
   43:     lTriangle.rotate(angle);
   44:     point.draw(lTriangle);
   45:     fTree(lTriangle, depth - 1, angle, point);
   46:
   47:     sf::CircleShape rTriangle (triangle.getRadius() / 2, 3);
   48:     rTriangle.setFillColor(sf::Color::Transparent);
   49:     rTriangle.setOutlineColor(sf::Color::Cyan);
   50:     rTriangle.setOutlineThickness(1);
   51:
   52:     rTriangle.setPosition(tf.transformPoint({-rTriangle.getRadius()* float(1
.65), rTriangle.getRadius() + rTriangle.getRadius() / 2}));
   53:     rTriangle.rotate(angle);
   54:     point.draw(rTriangle);
   55:     fTree(rTriangle, depth - 1, angle, point);
   56:
   57:     sf::CircleShape fTriangle (triangle.getRadius() / 2, 3);
   58:     fTriangle.setFillColor(sf::Color::Transparent);
```

```
   59:     fTriangle.setOutlineColor(sf::Color::Cyan);
   60:     fTriangle.setOutlineThickness(1);
   61:
   62:     fTriangle.setPosition(tf.transformPoint({fTriangle.getRadius() * 2 - fTr
iangle.getRadius() / 16, -fTriangle.getRadius()*2 + fTriangle.getRadius() / 2}));
   63:     fTriangle.rotate(angle);
   64:     point.draw(fTriangle);
   65:     fTree(fTriangle, depth - 1, angle, point);
   66: }
   67:
   68: //virtual draw
   69: void FTree::draw(sf::RenderTarget& point, sf::RenderStates states) const
   70: {
   71:     //draw tree
   72:     point.draw(triangle, states);
   73:     //start recursion
   74:     int angle = 180;
   75:     fTree(triangle, depth, angle, point);
   76: }
   77:
   78:
   79: int main(int argc, char* argv[])
   80: {
   81:     //checks for correct arguements in command line
   82:     if(argc != 3){
   83:         cout << "Failed to provide correct number of arguements\n";
   84:         return -1;
   85:     }
   86:
   87:     //assign command line arguements to variables
   88:     string strLen(argv[1]); // L length of triangle
   89:     string strdepth(argv[2]); // N the depth of the recursion
   90:     double base = stoi(strLen);
   91:     int depth = stoi(strdepth);
   92:     int angle = 180;
   93:     //create window and display
   94:     sf::RenderWindow window(sf::VideoMode(500,500), "Window");
   95:
   96:     //create tree
   97:     FTree newFTree(base, depth, angle);
   98:
   99:     sf::Event event;
  100:     while (window.isOpen()){
  101:         sf::Event event;
  102:         while (window.pollEvent(event)){
  103:             if (event.type == sf::Event::Closed){
  104:                 window.close();
  105:             }
  106:         }
  107:         // window.clear();
  108:         window.draw(newFTree);
  109:         window.display();
  110:     }
  111:     return 0;
  112: }
```

```
    1: #ifndef FTree_HPP
    2: #define FTree_HPP
    3:
    4: //include libraries
    5: #include <iostream>
    6: #include <SFML/Graphics.hpp>
    7:
    8: using namespace std;
    9:
   10: //derived class from drawable
   11: class FTree : public sf::Drawable{
   12: public:
   13: //constructor
   14: FTree(int l, int s, int a);
   15: //destructor
   16: ~FTree(){}
   17: //recursive function
   18: void fTree(const sf::CircleShape triangle, int depth, int angle, sf::RenderT
arget& point) const;
   19:
   20: private:
   21: virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
   22:     int length;
   23:     int depth;
   24:     sf::CircleShape triangle;
   25: };
   26:
   27: #endif
```

```
 1: CC = g++
 2: CFLAGS = -g -c -Werror -Wall -pedantic -ansi -std=c++11
 3: LIBS = -lsfml-graphics -lsfml-window -lsfml-system
 4: SOURCES = CircularBuffer.cpp test.cpp
 5: OBJECTS =$(SOURCES:.cpp = .o)
 6: EXE = CircularBuffer
 7:
 8: all:    $(SOURCES) $(EXE)
 9:
10: $(EXE): $(OBJECTS)
11:         $(CC) $(OBJECTS) -o $@ $(LIBS)
12:
13: .cpp.o:
14:         $(CC) $(CFLAGS) $< -o $@
15:
16: clean:
17:         rm $(EXE) $(OBJECTS)
```

```
    1: #include "CircularBuffer.hpp"
    2: #include <stdint.h>
    3: #include <iostream>
    4: #include <exception>
    5: #include <stdexcept>
    6:
    7: CircularBuffer::CircularBuffer(int size) {
    8:     if (size < 1){
    9:         throw std::invalid_argument("CircularBuffer constructor: capacity mu
st be greater than zero. \n");
   10:     }
   11:     s = 0, first = 0, cap = size;
   12:     Buffer = new int16_t[cap];
   13: }
   14:
   15: int CircularBuffer::size() {
   16:     return s;
   17: }
   18:
   19: bool CircularBuffer::isEmpty() {
   20:     return s == 0;
   21: }
   22:
   23: bool CircularBuffer::isFull() {
   24:     return s == cap;
   25: }
   26: void CircularBuffer::enqueue(int16_t x) {
   27:     if (isFull()){
   28:     throw std::runtime_error("enqueue: can’t enqueue to a full ring. \n");
   29:     }
   30:
   31:     Buffer[(first + s) % cap] = x;
   32:     s++;
   33: }
   34:
   35: int16_t CircularBuffer::dequeue() {
   36:     if (isEmpty()){
   37:     throw std::runtime_error("CircularBuffer Empty\n");
   38:     }
   39:
   40:     s--;
   41:     int dequeued = first;
   42:     first = (first + 1) % cap;
   43:
   44:     return Buffer[dequeued];
   45: }
   46:
   47: int16_t CircularBuffer::peek() {
   48:     if (isEmpty()){
   49:         throw std::runtime_error("CircularBuffer Empty\n");
   50:     }
   51:
   52:     return Buffer[first];
   53: }
```

```
 1: #ifndef CircularBuffer_HPP
 2: #define CircularBuffer_HPP
 3: #include <stdint.h>
 4: #include <iostream>
 5:
 6: class CircularBuffer {
 7: public:
 8:     CircularBuffer(int size);
 9:     int size(); // return number of items currently in the buffer
10:     bool isEmpty(); // is the buffer empty (size equals zero)?
11:     bool isFull(); // is the buffer full (size equals size)?
12:     void enqueue(int16_t x); // add item x to the end
13:     int16_t dequeue(); // delete and return item from the front
14:     int16_t peek(); // return (but do not delete) item from the front
15:     private:
16:     int s;
17:     int cap;
18:     int16_t first;
19:     int16_t *Buffer;
20: };
21: #endif
```

```
 1: #include "CircularBuffer.hpp"
 2: #define BOOST_TEST_DYN_LINK
 3: #define BOOST_TEST_MODULE Main
 4: #include <boost/test/included/unit_test.hpp>
 5:
 6: BOOST_AUTO_TEST_CASE(CircularBufferConstructor) {
 7:     BOOST_REQUIRE_NO_THROW(CircularBuffer(100));
 8:
 9:     // requires a thrown exception
10:     BOOST_REQUIRE_THROW(CircularBuffer(0), std::exception);
11:     BOOST_REQUIRE_THROW(CircularBuffer(0), std::invalid_argument);
12:     BOOST_REQUIRE_THROW(CircularBuffer(-1), std::invalid_argument);
13: }
14:
15: //test size
16: BOOST_AUTO_TEST_CASE(size) {
17:     CircularBuffer test(20);
18:
19:     BOOST_REQUIRE(test.size() == 0);
20:     //add 1 to the buffer and check if buffer total equals 1
21:     test.enqueue(1);
22:     BOOST_REQUIRE(test.size() == 1);
23: }
24:
25: //test isEmpty
26: BOOST_AUTO_TEST_CASE(isEmpty) {
27:     CircularBuffer test(1);
28:     BOOST_REQUIRE(test.isEmpty()==1);
29:     test.enqueue(1);
30:     BOOST_REQUIRE(test.isFull()==1);
31: }
32:
33: //test isFull
34: BOOST_AUTO_TEST_CASE(isFull) {
35:     CircularBuffer test(1);
36:     test.enqueue(5);
37:     BOOST_REQUIRE(test.isFull()==1);
38: }
39:
40: //test enqueue
41: BOOST_AUTO_TEST_CASE(enqueue) {
42:     CircularBuffer test(3);
43:     BOOST_REQUIRE_NO_THROW(test.enqueue(1));
44:     BOOST_REQUIRE_NO_THROW(test.enqueue(5));
45:     BOOST_REQUIRE_NO_THROW(test.enqueue(2));
46: }
47:
48: //test dequeue
49: BOOST_AUTO_TEST_CASE(dequeue) {
50:     CircularBuffer test(5);
51:     test.enqueue(8);
52:     test.enqueue(6);
53:     test.enqueue(2);
54:     test.enqueue(9);
55:     test.enqueue(3);
56:     BOOST_REQUIRE_NO_THROW(test.dequeue());
57:     BOOST_REQUIRE_NO_THROW(test.dequeue());
58:     BOOST_REQUIRE_NO_THROW(test.dequeue());
59:     BOOST_REQUIRE_NO_THROW(test.dequeue());
60:     BOOST_REQUIRE_NO_THROW(test.dequeue());
61: }
```

```
62:
63: //test peek
64: BOOST_AUTO_TEST_CASE(peek) {
65:     CircularBuffer test(3);
66:     test.enqueue(8);
67:     test.enqueue(2);
68:     test.enqueue(3);
69:     BOOST_REQUIRE_NO_THROW(test.peek());
70:     BOOST_REQUIRE_NO_THROW(test.peek());
71:     BOOST_REQUIRE_NO_THROW(test.peek());
72: }
```

```
 1: CC=g++
 2: CFLAGS= -std=c++14 -Wall -Werror -pedantic
 3: OBJ=GuitarSim.o RingBuffer.o StringSound.o
 4: LIBS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
 5: DEPS=
 6: EXE=GuitarSim
 7:
 8: all: $(EXE)
 9:         @echo Make complete.
10:
11: $(EXE): $(OBJ)
12:         $(CC) $(CFLAGS) $(OBJ) -o $(EXE) $(LIBS)
13:
14: %.o: %.cpp $(DEPS)
15:         $(CC) $(CFLAGS) -c $<
16:
17: %.o: $.cpp $.hpp
18:         $(CC) $(CFLAGS) -c $^
19:
20: clean:
21:         rm $(EXE) $(OBJ)
```

```
 1: #include "RingBuffer.hpp"
 2: #include <stdint.h>
 3: #include <iostream>
 4: #include <exception>
 5: #include <stdexcept>
 6:
 7: using namespace std;
 8:
 9: RingBuffer::RingBuffer(int capacity) {
10: if (capacity < 1){
11: throw invalid_argument("Cannot Instantiate with capacity < 1)\n");
12: }
13: s = 0, top = 0, cap = capacity;
14: Buffer = new int16_t[cap];
15: }
16:
17: int RingBuffer::size() {
18: return s;
19: }
20:
21: bool RingBuffer::isEmpty() {
22: return s == 0;
23: }
24:
25: bool RingBuffer::isFull() {
26: return s == cap;
27: }
28: void RingBuffer::enqueue(int16_t x) {
29: if (isFull()){
30: throw runtime_error("RingBuffer Full\n");
31: }
32:
33: Buffer[(top + s) % cap] = x;
34: s++;
35: }
36:
37: int16_t RingBuffer::dequeue() {
38: if (isEmpty()){
39: throw runtime_error("RingBuffer Empty\n");
40: }
41:
42: s--;
43: int dequeued = top;
44: top = (top + 1) % cap;
45:
46: return Buffer[dequeued];
47: }
48:
49: int16_t RingBuffer::peek() {
50: if (isEmpty()){
51: throw runtime_error("RingBuffer Empty\n");
52: }
53:
54: return Buffer[top];
55: }
```

```
 1: #ifndef RINGBUFFER_HPP
 2: #define RINGBUFFER_HPP
 3: #include <stdint.h>
 4: #include <iostream>
 5:
 6: class RingBuffer {
 7: public:
 8: RingBuffer(int capacity);
 9:     int size(); // return number of items currently in the buffer
10:     bool isEmpty(); // is the buffer empty (size equals zero)?
11:     bool isFull(); // is the buffer full (size equals capacity)?
12:     void enqueue(int16_t x); // add item x to the end
13:     int16_t dequeue(); // delete and return item from the front
14:     int16_t peek(); // return (but do not delete) item from the front
15: private:
16:     int s;
17:     int cap;
18:     int16_t top;
19:     int16_t *Buffer;
20: };
21: #endif
```

```
 1: #ifndef StringSound_HPP
 2: #define StringSound_HPP
 3:
 4:
 5: #include <stdint.h>
 6: #include <vector>
 7: #include <exception>
 8: #include "RingBuffer.hpp"
 9:
10: using namespace std;
11:
12: class StringSound{
13: public:
14:     StringSound(double frequency);
15:     StringSound(vector<int16_t> init);
16:     StringSound (const StringSound &obj) {}; // no copy const
17:     ~ StringSound();
18:     void pluck();
19:     void tic();
20:     int16_t sample();
21:     int time();
22: private:
23:     RingBuffer* pBuffer;
24:     int step;
25: };
26:
27: #endif
```

```
 1: #include "StringSound.hpp"
 2: #include <vector>
 3: #include <cmath>
 4:
 5: StringSound::~StringSound(){}
 6:
 7: StringSound::StringSound(double frequency){
 8:     int x = ceil(44100 / frequency);
 9:     pBuffer = new RingBuffer(x);
10:     while(!pBuffer->isFull()){
11:         pBuffer->enqueue(0);
12:     }
13:     step = 0;
14: }
15:
16: StringSound::StringSound(vector<int16_t> init){
17:     pBuffer = new RingBuffer(init.size());
18:     int i = 0;
19:     while(!pBuffer->isFull()){
20:         pBuffer->enqueue(init[i]);
21:         i++;
22:     }
23: }
24:
25: void StringSound::pluck(){
26:     if(pBuffer->isFull()){
27:         for(int i = 0; i < pBuffer->size(); i++){
28:             pBuffer->dequeue();
29:         }
30:     }
31:     while(!pBuffer->isFull()){
32:         pBuffer->enqueue((int16_t)(rand()*0xffff));
33:     }
34: }
35:
36: void StringSound::tic(){
37:     int16_t x = .5 * .996 * (pBuffer->dequeue() + pBuffer->peek());
38:     pBuffer->enqueue(x);
39:
40:     ++step;
41: }
42:
43: int16_t StringSound::sample(){
44:     int16_t Sample = pBuffer->peek();
45:     return Sample;
46: }
47:
48: int StringSound::time(){
49:     return step;
50: }
```

```
    1: #include <SFML/Graphics.hpp>
    2: #include <SFML/System.hpp>
    3: #include <SFML/Audio.hpp>
    4: #include <SFML/Window.hpp>
    5:
    6: #include <math.h>
    7: #include <limits.h>
    8:
    9: #include <iostream>
   10: #include <string>
   11: #include <exception>
   12: #include <stdexcept>
   13: #include <vector>
   14:
   15: #include "RingBuffer.hpp"
   16: #include "StringSound.hpp"
   17:
   18: #define CONCERT_A 220.0
   19: #define SAMPLES_PER_SEC 44100
   20: const int keyInput = 37;
   21:
   22: using namespace std;
   23:
   24: vector<sf::Int16> makeSample(StringSound gs) {
   25:     vector<sf::Int16> samples;
   26:
   27:     gs.pluck();
   28:     int duration = 8;
   29:     int i;
   30:     for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
   31:         gs.tic();
   32:         samples.push_back(gs.sample());
   33:     }
   34:
   35:     return samples;
   36: }
   37:
   38: int main() {
   39:     sf::RenderWindow window(sf::VideoMode(300, 200), "SFML Plucked String So
und Lite");
   40:     sf::Event event;
   41:
   42:     // define freq and a vector for sound samples
   43:     double freq;
   44:     vector<sf::Int16> sample;
   45:     vector<vector<sf::Int16>> samples(keyInput);
   46:     vector<sf::SoundBuffer> buffers(keyInput);
   47:     vector<sf::Sound> sounds(keyInput);
   48:
   49:     // Keys used for sound
   50:     string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/â\200\231 ";
   51:
   52:     // Loop transitions through the keyboard for user input
   53:     for (int i = 0; i < 37; i++) {
   54:         freq = CONCERT_A * pow(2, ( (i - 24)/12.0));
   55:         StringSound tmp = StringSound(freq);
   56:         sample = makeSample(tmp);
   57:         samples[i] = sample;
   58:
   59:         if (!buffers[i].loadFromSamples(&samples[i][0], samples[i].size(), 2
, SAMPLES_PER_SEC)) {
```

```
   60:             throw runtime_error("Failed to load Samples");
   61:         }
   62:         sounds[i].setBuffer(buffers[i]);
   63:     }
   64:
   65:     while (window.isOpen()) {
   66:     while (window.pollEvent(event)) {
   67:       switch (event.type) {
   68:       case sf::Event::Closed:
   69:         window.close();
   70:         break;
   71:
   72:       case sf::Event::KeyPressed:
   73:         switch (event.key.code) {
   74:         case sf::Keyboard::Q:
   75:           sounds.at(0).play();
   76:           break;
   77:           case sf::Keyboard::Num2:
   78:           sounds.at(1).play();
   79:           break;
   80:           case sf::Keyboard::W:
   81:           sounds.at(2).play();
   82:           break;
   83:           case sf::Keyboard::E:
   84:           sounds.at(3).play();
   85:           break;
   86:           case sf::Keyboard::Num4:
   87:           sounds.at(4).play();
   88:           break;
   89:           case sf::Keyboard::R:
   90:           sounds.at(5).play();
   91:           break;
   92:           case sf::Keyboard::Num5:
   93:           sounds.at(6).play();
   94:           break;
   95:           case sf::Keyboard::T:
   96:           sounds.at(7).play();
   97:           break;
   98:           case sf::Keyboard::Y:
   99:           sounds.at(8).play();
  100:           break;
  101:           case sf::Keyboard::Num7:
  102:           sounds.at(9).play();
  103:           break;
  104:           case sf::Keyboard::U:
  105:           sounds.at(10).play();
  106:           break;
  107:           case sf::Keyboard::Num8:
  108:           sounds.at(11).play();
  109:           break;
  110:           case sf::Keyboard::I:
  111:           sounds.at(12).play();
  112:           break;
  113:           case sf::Keyboard::Num9:
  114:           sounds.at(13).play();
  115:           break;
  116:           case sf::Keyboard::O:
  117:           sounds.at(14).play();
  118:           break;
  119:           case sf::Keyboard::P:
  120:           sounds.at(15).play();
```

```
121:            break;
122:            case sf::Keyboard::Dash:
123:            sounds.at(16).play();
124:            break;
125:            case sf::Keyboard::LBracket:
126:            sounds.at(17).play();
127:            break;
128:            case sf::Keyboard::Equal:
129:            sounds.at(18).play();
130:            break;
131:            case sf::Keyboard::Z:
132:            sounds.at(19).play();
133:            break;
134:            case sf::Keyboard::X:
135:            sounds.at(20).play();
136:            break;
137:            case sf::Keyboard::D:
138:            sounds.at(21).play();
139:            break;
140:            case sf::Keyboard::C:
141:            sounds.at(22).play();
142:            break;
143:            case sf::Keyboard::F:
144:            sounds.at(23).play();
145:            break;
146:            case sf::Keyboard::V:
147:            sounds.at(24).play();
148:            break;
149:            case sf::Keyboard::G:
150:            sounds.at(25).play();
151:            break;
152:            case sf::Keyboard::B:
153:            sounds.at(26).play();
154:            break;
155:            case sf::Keyboard::N:
156:            sounds.at(27).play();
157:            break;
158:            case sf::Keyboard::J:
159:            sounds.at(28).play();
160:            break;
161:            case sf::Keyboard::M:
162:            sounds.at(29).play();
163:            break;
164:            case sf::Keyboard::K:
165:            sounds.at(30).play();
166:            break;
167:            case sf::Keyboard::Comma:
168:            sounds.at(31).play();
169:            break;
170:            case sf::Keyboard::Period:
171:            sounds.at(32).play();
172:            break;
173:            case sf::Keyboard::SemiColon:
174:            sounds.at(33).play();
175:            break;
176:            case sf::Keyboard::Slash:
177:            sounds.at(34).play();
178:            break;
179:            case sf::Keyboard::Quote:
180:            sounds.at(35).play();
181:            break;
```

```
182:           case sf::Keyboard::Space:
183:           sounds.at(36).play();
184:           break;
185:         default:
186:           break;
187:         }
188:
189:       default:
190:         break;
191:       }
192:         window.clear();
193:         window.display();
194:     }
195:     }
196:     return 0;
197: }
```

```
 1: CC= g++
 2: CFLAGS= -g -Wall -Werror -std=c++0x -pedantic
 3: SFLAGS= -lsfml-system
 4:
 5: all: ps5
 6:
 7: ps5: ps5.o main.o
 8:         $(CC) ps5.o main.o -o ps5 $(SFLAGS)
 9:
10: ps5.o: ps5.cpp ps5.hpp
11:         $(CC) -c ps5.cpp ps5.hpp $(CFLAGS)
12:
13: main.o: main.cpp ps5.hpp
14:         $(CC) -c main.cpp ps5.hpp $(CFLAGS)
15:
16: clean:
17:         rm *.o
18:         rm ps5
19:         rm .gch
```

```
 1: #include <iostream>
 2: #include <vector>
 3: #include <sstream>
 4:
 5: #include "ps5.hpp"
 6:
 7: using namespace std;
 8:
 9: //constructor
10: EDistance::EDistance(string n, string m):N(n),M(m){
11:     vector<int> newVect;
12:
13:     for(int i = 0; i < static_cast<int>(M.length()) + 1; i++){
14:         newVect.push_back(0);
15:     }
16:
17:     for(int i = 0; i < static_cast<int>(N.length()) + 1; i++){
18:         opt.push_back(newVect);
19:     }
20:
21:     editDistance = OptDistance();
22:     editStr = Alignment();
23: }
24:
25: //penalties for mismatching characters
26: int EDistance::penalty(char a, char b){
27:     if(a == b){
28:         return 0;
29:     }
30:     else{
31:         return 1;
32:     }
33: }
34:
35: int EDistance::min(int a, int b, int c){
36:     int min;
37:     if(a < b){
38:         min = a;
39:         if(min > c){
40:             min = c;
41:             return min;
42:         } else{
43:             return min;
44:         }
45:     }
46:     else{
47:         min = b;
48:         if(min > c){
49:             min = c;
50:             return min;
51:         } else{
52:             return min;
53:         }
54:     }
55: }
56:
57: //finds optdistance
58: int EDistance::OptDistance(){
59:     for(int i = opt.size() - 1; i >= 0; i--){
60:         for(int j = opt[i].size() - 1; j >= 0; j--){
61:             if((i == static_cast<int>(opt.size() - 1)) && (j == static_cast<
```

```
int>(opt[i].size() - 1))){
   62:                 opt[i][j] = 0;
   63:             }
   64:
   65:             else if(i == static_cast<int>(opt.size() - 1)){
   66:                 opt[i][j] = opt[i][j + 1] + 2;
   67:             }
   68:             else if (j == static_cast<int>(opt[i].size() - 1)){
   69:                 opt[i][j] = opt[i + 1][j] + 2;
   70:             }
   71:             else{
   72:                 opt[i][j] = min(static_cast<int>(opt[i+1][j+1] + penalty(N[i
], M[j])), static_cast<int>(opt[i + 1][j] + 2), static_cast<int>(opt[i][j + 1]+ 2))
;
   73:             }
   74:         }
   75:     }
   76:     return opt[0][0];
   77: }
   78:
   79: //Alignment function
   80: string EDistance::Alignment(){
   81:     int i = 0;
   82:     int j = 0;
   83:
   84:     stringstream sstream;
   85:
   86:     while(i < static_cast<int>(opt.size() - 1) || j < static_cast<int>(opt[0
].size() - 1))
   87:     {
   88:         if((i < static_cast<int>(opt.size() - 1)) && (j < static_cast<int> (
opt[0].size() - 1)) && (opt[i+1][j+1] <= opt[i+1][j] + 1) && (opt[i+1][j+1] <= opt[
i][j+1] + 1))
   89:         {
   90:             sstream << N[i] << " " << M[j] << " " << opt[i][j] - opt[i+1][j+
1] << endl;
   91:             i++;
   92:             j++;
   93:         }
   94:         else if(((i < static_cast<int>(opt.size() - 1)) && (opt[i+1][j] <= o
pt[i][j+1])) || (j == static_cast<int>(opt[0].size() - 1)))
   95:         {
   96:             sstream << N[i] << " " << "_" << " " << opt[i][j] - opt[i+1][j]
<< endl;
   97:             i++;
   98:         }
   99:         else
  100:         {
  101:             sstream << "_" << " " << M[j] << " " << opt[i][j] - opt[i][j+1]
<< endl;
  102:             j++;
  103:         }
  104:     }
  105:
  106:     return sstream.str();
  107: }
  108:
  109: //display matrix
  110: void EDistance::print()
  111: {
  112:     for(unsigned i = 0; i < opt.size(); i ++){
```

```
  113:          for(unsigned j = 0; j < opt[i].size(); j++){
  114:              cout << " " << opt[i][j];
  115:          }
  116:          cout << endl;
  117:      }
  118: }
```

```
 1: #ifndef PS5_HPP
 2: #define PS5_HPP
 3:
 4: #include <iostream>
 5: #include <vector>
 6:
 7: using namespace std;
 8:
 9: class EDistance {
10: public:
11:     EDistance(string n, string m);
12:     int penalty(char a, char b);
13:     int min(int a, int b, int c);
14:     int OptDistance();
15:     string Alignment();
16:     void print();
17:     int getDistance() const { return editDistance;}
18:     string getString() const{ return editStr;}
19:
20: private:
21:     vector< vector< int > > opt; //holds opt data
22:     string N; //X axis string
23:     string M; //Y axis string
24:     string editStr;
25:     int editDistance;
26: };
27:
28: #endif
```

```
 1: #include <iostream>
 2: #include <sstream>
 3: #include <SFML/System.hpp>
 4:
 5: #include "ps5.hpp"
 6:
 7: using namespace std;
 8:
 9: int main(){
10:     //SFML clock
11:     sf::Clock clock;
12:     sf::Time t;
13:
14:     string nStr, mStr;
15:     cin >> nStr;
16:     cin >> mStr;
17:
18:     //instantiate new ED
19:     EDistance newDistance(nStr, mStr);
20:
21:     //output in terminal
22:     cout << newDistance.getString() << endl;
23:     cout << "Edit distance: " << newDistance.getDistance() << endl;
24:     t = clock.getElapsedTime();
25:     cout << "Execution time is " << t.asSeconds() << " seconds \n";
26:
27:     return 0;
28: }
```

```
    1: Boost = -lboost_unit_test_framework
    2: all:
    3:         make TextGenerator
    4:         make btest
    5:
    6: TextGenerator: RandWriter.o TextWriter.o
    7:         g++ -o RandWriter TextWriter.o RandWriter.o -ansi -pedantic -Wall -W
error
    8:
    9: btest: test.o RandWriter.o
   10:         g++ test.o RandWriter.o -o btest $(Boost)
   11:
   12: TextWriter.o: RandWriter.h TextWriter.cpp
   13:         g++ -c TextWriter.cpp -ansi -pedantic -Wall -Werror
   14:
   15: RandWriter.o: RandWriter.h RandWriter.cpp
   16:         g++ -c RandWriter.cpp -ansi -pedantic -Wall -Werror
   17:
   18: test.o: test.cpp
   19:         g++ -c test.cpp $(Boost)
   20:
   21: clean:
   22:         rm -f *.o *Ë\234 TextWriter *Ë\234btest
```

```
 1: #include <iostream>
 2: #include <string>
 3: #include <vector>
 4: #include <map>
 5: #include <stdexcept>
 6: #include <cstdlib>
 7: #include <ctime>
 8: #include "RandWriter.h"
 9:
10: using namespace std;
11:
12: RandWriter::RandWriter(string text, int k){
13:     order = k;
14:     init = text;
15:     srand(time(NULL));
16:
17:     for (unsigned i = 0; i < text.size(); i++){
18:         if (string::npos == alphabet.find(text[i])){
19:             alphabet.push_back(text[i]);
20:         }
21:     }
22:     for (unsigned i = 0; i < text.size(); i++) {
23:         string newStr;
24:         string newStr2;
25:
26:         //create new string
27:         for (unsigned j = i; j < i + k; j++)
28:         if (j >= text.size())
29:             newStr.push_back(text[j - text.size()]);
30:         else
31:             newStr.push_back(text[j]);
32:
33:         // string repeat in new string
34:         if (k_grams.end() == k_grams.find(newStr))
35:             k_grams[newStr] = 1;
36:         else
37:             k_grams[newStr] += 1;
38:
39:         // all possible k+1 strings
40:         for (unsigned j = 0; j < alphabet.size(); j++)
41:         if (k_grams.end() == k_grams.find(newStr + alphabet[j]))
42:             k_grams[newStr + alphabet[j]] = 0;
43:
44:         for (unsigned j = i; j < i + k + 1; j++)
45:         if (j >= text.size())
46:             newStr2.push_back(text[j - text.size()]);
47:         else
48:             newStr2.push_back(text[j]);
49:
50:         k_grams[newStr2] += 1;
51:     }
52: }
53:
54: int RandWriter::order_k() const{
55:     return order;
56: }
57:
58: int RandWriter::freq(string k_gram){
59:     if (k_gram.size() != (unsigned)order)
60:         throw runtime_error("k_gram not equal to k");
61:
```

```
   62:     if (order == 0)
   63:         return init.size();
   64:     else
   65:         return k_grams[k_gram];
   66: }
   67: int RandWriter::freq(string k_gram, char c){
   68:     if (k_gram.size() != (unsigned)order)
   69:         throw runtime_error("k_gram not equal to k");
   70:     if (order == 0) {
   71:         int count = 0;
   72:         for (unsigned i = 0; i < init.size(); i++){
   73:             if (init[i] == c)
   74:             count++;
   75:             return count;
   76:         }
   77:     }
   78:     else {
   79:         return k_grams[k_gram + c];
   80:     }
   81:
   82:     return 0;
   83: }
   84:
   85: char RandWriter::k_Rand(string k_gram){
   86:     if (k_gram.size() != (unsigned)order || k_grams.end() == k_grams.find(k_
gram))
   87:         throw runtime_error("k_Rand: k_gram not valid");
   88:
   89:     string newStr;
   90:     for (unsigned i = 0; i < alphabet.size(); i++)
   91:     for (int j = 0; j < k_grams[k_gram + alphabet[i]]; j++)
   92:         newStr.push_back(alphabet[i]);
   93:
   94:     return newStr[rand() % newStr.size()];
   95: }
   96:
   97: string RandWriter::generate(string k_gram, int L){
   98:     string newStr = k_gram;
   99:     string ret = k_gram;
  100:     char rc;
  101:
  102:     for (int i = 0; i < L - order; i++){
  103:         rc = k_Rand(newStr);
  104:         ret.push_back(rc);
  105:         newStr.erase(newStr.begin());
  106:         newStr.push_back(rc);
  107:     }
  108:
  109:     return ret;
  110: }
  111:
  112: ostream& operator<<(ostream& in, RandWriter& m){
  113:     in << "\n" << "String: \"" << m.init << "\"" << endl;
  114:     in << "Order:" << m.order << endl;
  115:     in << "Alphabet: \"" << m.alphabet << "\"" << "\n" << endl;
  116:
  117:     in << "Markov Map" << endl;
  118:     map <string, int> newStr = m.k_grams;
  119:     for (map<string, int>::iterator it = newStr.begin();
  120:     it != newStr.end(); ++it) {
  121:         in << it->first << " " << it->second << " => ";
```

```
  122:          for (unsigned i = 0; i < m.alphabet.size(); i++) {
  123:              it++;
  124:              in << it->first << " " << it->second << " ";
  125:          }
  126:
  127:          in << endl;
  128:      }
  129:
  130:      return in;
  131: }
```

```
 1: #ifndef RANDWRITER_H
 2: #define RANDWRITER_H
 3:
 4: #include <iostream>
 5: #include <string>
 6: #include <map>
 7:
 8: using namespace std;
 9:
10: class RandWriter{
11: public:
12:     RandWriter(string text, int k);
13:
14:     int order_k() const;
15:     int freq(string k_gram);
16:     int freq(string k_gram, char c);
17:     char k_Rand(string k_gram);
18:     string generate(string k_gram, int L);
19:
20:     friend ostream& operator<<(ostream& in, RandWriter& m);
21:
22: private:
23:     string alphabet;
24:     string init;
25:     int order;
26:
27:     map <string,int> k_grams;
28: };
29: #endif
```

```
 1: #include <iostream>
 2: #include <string>
 3: #include <sstream>
 4: #include <cstdlib>
 5: #include "RandWriter.h"
 6:
 7: using namespace std;
 8:
 9: int main(int argc, char* argv[]) {
10:     if (argc < 3) {
11:         cout << "Not enough arguements" << endl;
12:         return 1;
13:     }
14:     else if (argc > 3) {
15:         cout << "Too many arguements" << endl;
16:         return 1;
17:     }
18:     int k = atoi(argv[1]);
19:     int L = atoi(argv[2]);
20:
21:     string input;
22:     string cursor;
23:
24:     while (cin >> cursor) {
25:         input += " " + cursor;
26:         cursor = "";
27:     }
28:     RandWriter model(input, k);
29:     string str;
30:
31:     for (int i = 0; i < k; i++){
32:         str.push_back(input[i]);
33:     }
34:     cout << "SUCCESS" << endl;
35:     cout << "Generated String: " << model.generate(str, L) << endl;
36:
37:     return 0;
38: }
```

```
 1: CC = g++
 2: CFLAGS =-g -c -Werror -Wall -pedantic -ansi -std=c++11
 3: BOOST = -lboost_regex -lboost_date_time
 4: SOURCES =ps7.cpp
 5: OBJECTS =$(SOURCES:.cpp = .o)
 6: EXE =ps7
 7:
 8: all:    $(SOURCES) $(EXE)
 9:
10: $(EXE): $(OBJECTS)
11:                 $(CC) $(OBJECTS) -o $@ $(BOOST)
12:
13: .cpp.o:
14:                 $(CC) $(CFLAGS) $< -o $@
15:
16: clean:
17:                 rm $(EXE) $(OBJECTS)
```

```
    1: #include <iostream>
    2: #include <fstream>
    3: #include <string>
    4: #include <iomanip>
    5: #include <boost/regex.hpp>
    6:
    7: #include "boost/date_time/gregorian/gregorian.hpp"
    8: #include "boost/date_time/posix_time/posix_time.hpp"
    9:
   10: using std::ifstream;
   11: using std::ofstream;
   12: using std::cout;
   13: using std::cin;
   14: using std::endl;
   15: using std::string;
   16: using std::left;
   17: using std::right;
   18: using std::getline;
   19: using std::setw;
   20:
   21: using boost::gregorian::date;
   22: using boost::gregorian::from_simple_string;
   23: using boost::gregorian::date_period;
   24: using boost::gregorian::date_duration;
   25: using boost::posix_time::duration_from_string;
   26: using boost::regex;
   27: using boost::regex_search;
   28: using boost::posix_time::ptime;
   29: using boost::posix_time::time_duration;
   30:
   31: int main(int argc, char* argv[]) {
   32:     if (argc < 2) {
   33:         cout << "Not enough arguments" << endl;
   34:         exit(1);
   35:     }
   36:     string inputFileName = argv[1];
   37:     string outputFileName = inputFileName + ".rpt";
   38:
   39:     ifstream inputFile(inputFileName);
   40:     ofstream outputFile(outputFileName);
   41:
   42:     string holder;
   43:
   44:     string bootStartDate, bootCompleteDate;
   45:     string bootStartTime, bootCompleteTime;
   46:
   47:     int initiated = 0;
   48:     int completed = 0;
   49:     int linesScanned =0;
   50:     int startLine = 0;
   51:
   52:     regex startRegex("(\\(log.c.166\\) server started)");
   53:     regex endRegex("(oejs.AbstractConnector:Started SelectChannelConnecctor)
");
   54:
   55:     outputFile << "---- DEVICE BOOT REPORT ----" << endl;
   56:
   57:     while (getline(inputFile, holder)) {
   58:         linesScanned++;
   59:
   60:         if (regex_search(holder, startRegex)) {
```

```
   61:             initiated++;
   62:
   63:             if(!bootStartDate.empty()) {
   64:                 outputFile << left << setw(6) << startLine
   65:                         << "(" << inputFileName << ")"
   66:                         << ": " << bootStartDate << " " << setw(12)
   67:                         << bootStartTime << " Boot Start" << endl;
   68:                 outputFile << "**** Incomplete boot ****" << endl << endl;
   69:
   70:                 bootStartDate.clear();
   71:                 bootStartTime.clear();
   72:             }
   73:
   74:             startLine = linesScanned;
   75:
   76:             for (int i = 0; i < 10; i++) {
   77:                 bootStartDate.push_back(holder.at(i));
   78:             }
   79:
   80:             for (int i = 11; i < 23; i++) {
   81:                 bootCompleteTime.push_back(holder.at(i));
   82:             }
   83:         }
   84:
   85:         if (regex_search(holder, endRegex)) {
   86:             completed++;
   87:
   88:             outputFile << left << setw(6) << startLine
   89:                 << "(" << inputFileName << ")"
   90:                 << ": " << bootStartDate << " " << setw(12)
   91:                 << bootStartTime << " Boot Start" << endl;
   92:
   93:             outputFile << left << setw(6) << linesScanned
   94:                 << "(" << inputFileName << ")"
   95:                 << ": " << bootCompleteDate << " " << setw(12)
   96:                 << bootCompleteTime << " Boot Start" << endl;
   97:
   98:             date d1(from_simple_string(bootStartDate));
   99:             date d2(from_simple_string(bootCompleteDate));
  100:
  101:             time_duration td1(duration_from_string(bootStartTime));
  102:             time_duration td2(duration_from_string(bootCompleteTime));
  103:
  104:             ptime t1(d1, td1);
  105:             ptime t2(d2, td2);
  106:             time_duration td = t2 - t1;
  107:
  108:             outputFile << "\tBoot Time: " << td.total_milliseconds() << "ms"
  109:                     << endl << endl;
  110:
  111:             bootStartDate.clear();
  112:             bootStartTime.clear();
  113:         }
  114:     }
  115:
  116:     outputFile << "---- TOTALS -----"                              << endl;
  117:     outputFile << "     Lines scanned: " << linesScanned          << endl;
  118:     outputFile << " Initiated boot-ups " << initiated             << endl;
  119:     outputFile << " Completed boot-ups " << completed             << endl;
  120:     outputFile << "Incomplete boot-ups " << initiated - completed << endl;
  121:
```

```
  122:     outputFile.close();
  123:     inputFile.close();
  124:
  125:     return 0;
  126: }
```