Your Name: Dangnhi Ngo
COMP IV: Project Portfolio
Spring 2018

Contents:

PS0: HELLO WORLD WITH SFML

Assignment Summary:

This assignment is to make us be familiar with SFML. We have built the SFML Hello Word by drawing the green circle, then drawn an image sprite and made something fun with it. I have created my sprite by the icon of a man and tried to make it move in 4 directions with 4 arrow keys on the keyboard (Left, Right, Up and Down).

In addition, to each different direction, the sprite would change its position which is appropriate to that position. Also, I have used the code "setRotation" to make the sprite rotate 45 degrees (Line 16, main.cpp).

Algorithms/Data Structures:

The assignment uses SFML window on Linux, sprites, textures and image sprite respond to keystrokes.

Some class members used: sf::RenderWindow, sf::CircleShape, sf::Texture, sf::Sprite.

Besides, sf::Keyboard::isKeyPressed is used to make the sprite move to the desired direction that user controls through the keyboard.

Accomplishment:

Through this assignment, I have learnt how to create the SFML window on Linux Operating System and, simultaneously, been familiar with class members and functions in SFML. Moreover, the assignment has taught me how to compile the source file and make it work by a bunch of command lines.

main.cpp

```
 1: #include <SFML/Graphics.hpp>
 2: int main() {
 3:     sf::RenderWindow window (sf::VideoMode(200, 200), "SFML works!");
 4:     sf::CircleShape shape(100.f);
 5:     shape.setFillColor(sf::Color::Green);
 6:     // Load a sprite to display
 7:     sf::Texture texture;
 8:     if (!texture.loadFromFile("sprite.png"))
 9:         return EXIT_FAILURE;
10:     sf::Sprite sprite(texture);
11:     sprite.setTexture(texture);
12:     sprite.setTextureRect(sf::IntRect(10, 130, 40, 68));
13:     //Make the sprite move
14:     sprite.move(sf::Vector2f(100, 80));
15:     //Make the sprite rotate 45 degrees
16:     sprite.setRotation(45);
17:     while (window.isOpen()) {
18:         sf::Event event;
19:         while (window.pollEvent(event)) {
20:             if (event.type == sf::Event::Closed)
21:             window.close();
22:         }
23:         // Right key is pressed: move the sprite
24:         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {
25:             sprite.setRotation(0);
26:             sprite.move(1, 0);
27:             sprite.setTextureRect(sf::IntRect(10, 190, 40, 70));
28:         }
29:         // Left key is pressed: move the sprite
30:         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
31:             sprite.setRotation(0);
32:             sprite.move(-1, 0);
33:             sprite.setTextureRect(sf::IntRect(10, 67, 40, 68));
34:         }
35:         // Up key is pressed: move the sprite
36:         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
37:             sprite.setRotation(0);
38:             sprite.move(0, -1);
39:             sprite.setTextureRect(sf::IntRect(10, 8, 40, 68));
40:         }
41:         // Down key is pressed: move the sprite
```

```
42:         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
43:             sprite.setRotation(0);
44:             sprite.move(0, 1);
45:             sprite.setTextureRect(sf::IntRect(10, 130, 40, 68));
46:         }
47:         //Control the frame rate
48:         window.setFramerateLimit(60);
49:         window.clear();
50:         window.draw(shape);
51:         window.draw(sprite);
52:         window.display();
53:     }
54:     return 0;
55: }
```

PS1: RECURSIVE GRAPHIC

Assignment Summary:

The purpose of this assignment is to develop a program that plots a Sierpinski triangle by using the recursion. The program takes one integer command-line argument N to control the depth of the recursion. The first equilateral triangle must be drawn in these constant endpoints (0,0), (1,0) and (1/2, √3/2), when N equals 0. Then, when N equals 1, we continue to draw an equilateral triangle inside with 3 points, respectively, in the mid points of the previous triangle's each side. After that, depends on the value of N(depth), the program provokes the function to create 3 other equilateral triangles (tri1, tri2, tri3) and uses recursion (Line 22, sierpinski.cpp).

Then, based on the Sierpinski assignment, we have created our original work called Original with any shape we want. I have used the recursion to generate the Hexagon.

Algorithms/Data Structures:

The Sierpinski class is derives from sf::Drawable, and uses sf::ConvexShape class to draw the filled triangles.

This assignment applies the definition of equilateral triangles to find the midpoint on each side of the triangle to draw triangles of the next level.

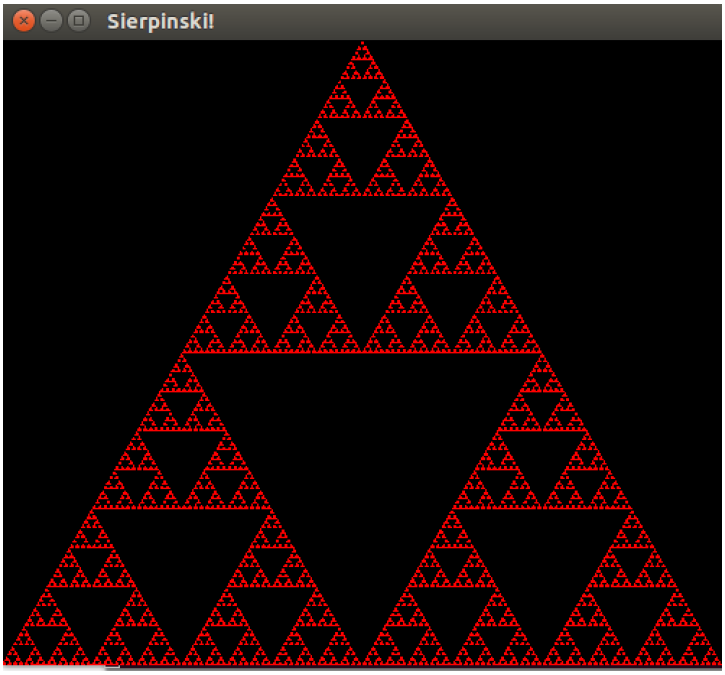
Accomplishment:

Through this assignment, I have learnt how to create the Makefile which contains two targets: all and clean to build and remove the executables.

Also, to make the program work, I have used the vector struct (push_back) to store the Sierpinski triangle as wel as my Hexagon for Original work (Line 7, 18 and Line 36-41, sierpinski.cpp).

In Original.cpp, I have applied the sf::Color(rand() % 255) to generate the random color for each of inside hexagon (Line 60)

Sierpinski Triangle:



Makefile

```
1:  CC = g++
2:  CFLAGS = -Wall -Werror -ansi -pedantic
3:  LFLAGS = -lsfml-window -lsfml-graphics -lsfml-audio -lsfml-system
4:
5:  all:  sierpinski original
6:
7:  sierpinski: main.o sierpinski.o
8:      $(CC) main.o sierpinski.o -o sierpinski $(LFLAGS)
9:  main.o: main.cpp sierpinski.hpp
10:     $(CC) -c main.cpp $(CFLAGS)
11: sierpinski.o: sierpinski.cpp sierpinski.hpp
12:     $(CC) -c sierpinski.cpp $(CFLAGS)
13: original: maino.o original.o
14:     $(CC) maino.o original.o -o original $(LFLAGS)
15: maino.o: maino.cpp original.hpp
16:     $(CC) -c maino.cpp $(CFLAGS)
17: original.o: original.cpp original.hpp
18:     $(CC) -c original.cpp $(CFLAGS)
19:
20: clean:
21:     rm *.o sierpinski original
```

main.cpp

```
 1: #include <SFML/Graphics.hpp>
 2: #include <iostream>
 3: #include <cmath>
 4: #include "sierpinski.hpp"
 5: using namespace std;
 6: int main(int argc, char* argv[]) {
 7:     if(argc < 3) {
 8:         cout<<"sierpinski [recursion-depth] [side-length]"<<endl;
 9:         return -1;
10:     }
11:     int depth = atoi(argv[1]);
12:     int side = atoi(argv[2]);
13:     Sierpinski s(side,depth);
14:     sf::RenderWindow
15:     window(sf::VideoMode(side,(int)(0.5*sqrt(3.)*(float)side)), "Sierpinski!");
16:     window.setVerticalSyncEnabled(true);
17:     window.setFramerateLimit(1);
18:     while(window.isOpen()) {
19:         sf::Event event;
20:         while(window.pollEvent(event)) {
21:             if(event.type == sf::Event::Closed)
22:                 window.close();
23:         }
24:         window.clear();
25:         window.draw(s);
26:         window.display();
27:     }
28:     return 0;
29: }
```

sierpinski.cpp

```
 1: #include <SFML/Graphics.hpp>
 2: #include <cmath>
 3: #include <iostream>
 4: #include <vector>
 5: #include "sierpinski.hpp"
 6: using namespace std;
 7: vector<struct triangle> vect;
 8: Sierpinski::Sierpinski(int side, int depth): _depth(depth) {
 9:     _top = sf::Vector2f(side / 2.0, 0);
10:     float height = 0.5 * sqrt(3.) * float(side);
```

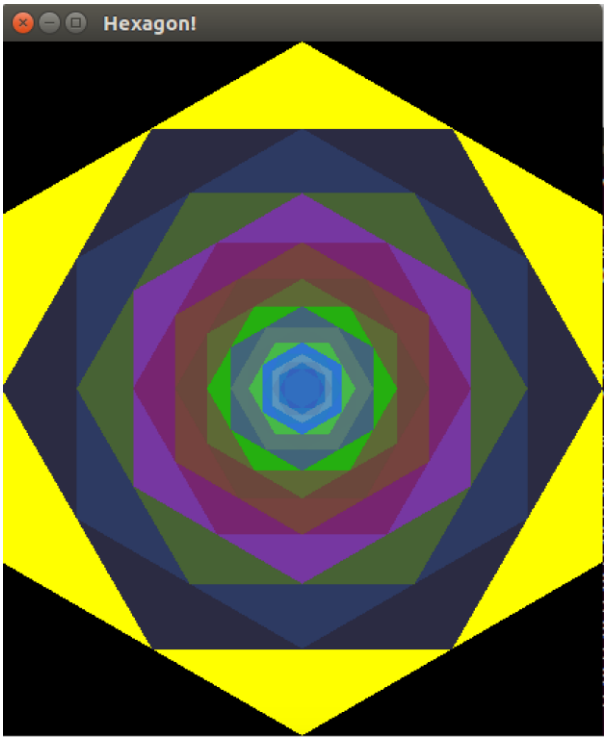
```
    _left = sf::Vector2f(0, height);
    _right = sf::Vector2f(side - 1, height);
    if (_depth >= 1) {
        struct triangle tri;
        tri.top   = sf::Vector2f((_left.x + _right.x)/2, _left.y);
        tri.right = sf::Vector2f((_top.x + _right.x)/2, _right.y/2);
        tri.left  = sf::Vector2f(_right.x/4, _right.y/2);
        vect.push_back(tri);
        sierpinski_tree(tri.top, tri.left, tri.right, _depth);
    }
}
void Sierpinski::sierpinski_tree(sf::Vector2f top, sf::Vector2f left, sf::Ve ctor2f right, int
depth) {
    struct triangle tri1;
    struct triangle tri2;
    struct triangle tri3;
    tri1.top  = sf::Vector2f((right.x-left.x)/2 + left.x, right.y);
    tri1.right = sf::Vector2f(right.x-(right.x-left.x)/4, left.y - (top.y-left.y)/2);
    tri1.left = sf::Vector2f( (right.x-left.x)/4 + left.x, left.y - (top.y-left.y)/2);
    tri2.top  = sf::Vector2f(top.x - (right.x-left.x)/2, top.y);
    tri2.right = sf::Vector2f(left.x + (right.x-left.x)/4, (top.y-left.y)/2 +left.y);
    tri2.left = sf::Vector2f(left.x - (right.x-left.x)/4, (top.y-left.y)/2 + left.y);
    tri3.top  = sf::Vector2f(top.x + (right.x-left.x)/2, top.y);
    tri3.right = sf::Vector2f(right.x + (right.x-left.x)/4, (top.y-left.y)/2 +left.y);
    tri3.left = sf::Vector2f(right.x - (right.x-left.x)/4, (top.y-left.y)/2 +left.y);
    if (depth > 1) {
        vect.push_back(tri1);
        vect.push_back(tri2);
        vect.push_back(tri3);
        sierpinski_tree(tri1.top, tri1.left, tri1.right, depth - 1);
        sierpinski_tree(tri2.top, tri2.left, tri2.right, depth - 1);
        sierpinski_tree(tri3.top, tri3.left, tri3.right, depth - 1);
    }
}
void Sierpinski::draw(sf::RenderTarget& target, sf::RenderStates states) const {
    sf::ConvexShape filledTriangle;
    filledTriangle.setPointCount(3);
    filledTriangle.setPoint(0, _left);
    filledTriangle.setPoint(1, _right);
    filledTriangle.setPoint(2, _top);
    filledTriangle.setFillColor(sf::Color::Red);
    target.draw(filledTriangle, states);
```

```
52:     for (unsigned int i = 0; i < vect.size(); i++) {
53:         filledTriangle.setPointCount(3);
54:         filledTriangle.setPoint(0, vect[i].top);
55:         filledTriangle.setPoint(1, vect[i].left);
56:         filledTriangle.setPoint(2, vect[i].right);
57:         filledTriangle.setFillColor(sf::Color::Black);
58:         target.draw(filledTriangle, states);
59:     }
60: }
```

sierpinski.hpp

```
 1: #ifndef SIERPINSKI_H
 2: #define SIERPINSKI_H
 3: #include <SFML/Graphics.hpp>
 4: using namespace std;
 5: struct triangle {
 6:     sf::Vector2f top, left, right;
 7: };
 8: class Sierpinski: public sf::Drawable {
 9: public:
10:     //Constructor that takes size and depth
11:     Sierpinski(int side, int depth);
12:     //Constructor that takes three points and depth
13:     void sierpinski_tree(sf::Vector2f top, sf::Vector2f left, sf::Vector2f rig ht, int depth);
14:     void virtual draw(sf::RenderTarget &target, sf::RenderStates states) const;
15: private:
16:     sf::Vector2f _top, _left, _right;
17:     int _depth;
18: };
19:
20: #endif
```

Original:



maino.cpp

```
 1:  #include <SFML/Graphics.hpp>
 2:  #include <iostream>
 3:  #include <cmath>
 4:  #include "original.hpp"
 5:  using namespace std;
 6:  int main(int argc, char* argv[]) {
 7:      if(argc < 3) {
 8:          cout<<"Hexagon [recursion-depth] [side-length]"<<endl;
 9:          return -1;
10:      }
11:      int depth = atoi(argv[1]);
12:      int side = atoi(argv[2]);
13:      Hexagon s(side, depth);
14:      sf::RenderWindow
15:      window(sf::VideoMode((int)(0.5*sqrt(3.)*(float)side),side), "Hexagon!");
16:      window.setVerticalSyncEnabled(true);
17:      window.setFramerateLimit(1);
18:      while(window.isOpen()) {
19:          sf::Event event;
20:          while(window.pollEvent(event)) {
21:              if(event.type == sf::Event::Closed)
```

```
22:                 window.close();
23:             }
24:             window.clear();
25:             window.draw(s);
26:             window.display();
27:         }
28:         return 0;
29:     }
```

original.cpp

```
 1:  #include <SFML/Graphics.hpp>
 2:  #include <cmath>
 3:  #include <iostream>
 4:  #include <vector>
 5:  #include "original.hpp"
 6:  using namespace std;
 7:  vector<struct hexagon> vect;
 8:  Hexagon::Hexagon(int side,int depth): _depth(depth) {
 9:      float width = (int)(0.5*sqrt(3.)*(float)side);
10:      _top = sf::Vector2f(width/2, 0);
11:      _left1 = sf::Vector2f(0, side/4);
12:      _left2 = sf::Vector2f(0, 3.0*side/4);
13:      _bottom = sf::Vector2f(width/2, side);
14:      _right1 = sf::Vector2f(width, 3.0*side/4);
15:      _right2 = sf::Vector2f(width, side/4);
16:      if (_depth >= 1) {
17:          struct hexagon hex;
18:          hex.top = sf::Vector2f((_top.x+_left1.x)/2, (_top.y+_left1.y)/2);
19:          hex.left1 = sf::Vector2f((_left1.x+_left2.x)/2,(_left1.y+_left2.y)/2);
20:          hex.left2 = sf::Vector2f((_left2.x+_bottom.x)/2,(_left2.y+_bottom.y)/2);
21:          hex.bottom = sf::Vector2f((_bottom.x+_right1.x)/2,(_bottom.y+_right1.y)/2);
22:          hex.right1 = sf::Vector2f((_right1.x+_right2.x)/2,(_right1.y+_right2.y)/2);
23:          hex.right2 = sf::Vector2f((_right2.x+_top.x)/2,(_right2.y+_top.y)/2);
24:          vect.push_back(hex);
25:          hex_tree(hex.top, hex.left1, hex.left2, hex.bottom, hex.right1, hex.right2, depth);
26:      }
27:  }
28:  void Hexagon::hex_tree(sf::Vector2f top, sf::Vector2f left1, sf::Vector2f le ft2, sf::Vector2f
bottom,sf::Vector2f right1,sf::Vector2f right2, int depth) {
29:      struct hexagon hex;
30:      hex.top = sf::Vector2f((top.x+left1.x)/2,(top.y+left1.y)/2);
31:      hex.left1 = sf::Vector2f((left1.x+left2.x)/2,(left1.y+left2.y)/2);
```

```
32:     hex.left2 = sf::Vector2f((left2.x+bottom.x)/2,(left2.y+bottom.y)/2);
33:     hex.bottom = sf::Vector2f((bottom.x+right1.x)/2,(bottom.y+right1.y)/2);
34:     hex.right1 = sf::Vector2f((right1.x+right2.x)/2,(right1.y+right2.y)/2);
35:     hex.right2 = sf::Vector2f((right2.x+top.x)/2,(right2.y+top.y)/2);
36:     if (depth > 1) {
37:         vect.push_back(hex);
38:         hex_tree(hex.top, hex.left1, hex.left2, hex.bottom, hex.right1, hex.righ t2, depth - 1);
39:     }
40: }
41: void Hexagon::draw(sf::RenderTarget& target, sf::RenderStates states) const {
42:     sf::ConvexShape filledHexagon;
43:     filledHexagon.setPointCount(6);
44:     filledHexagon.setPoint(0, _top);
45:     filledHexagon.setPoint(1, _left1);
46:     filledHexagon.setPoint(2, _left2);
47:     filledHexagon.setPoint(3, _bottom);
48:     filledHexagon.setPoint(4, _right1);
49:     filledHexagon.setPoint(5, _right2);
50:     filledHexagon.setFillColor(sf::Color::Yellow);
51:     target.draw(filledHexagon, states);
52:     for (unsigned int i = 0; i < vect.size(); i++) {
53:         filledHexagon.setPointCount(6);
54:         filledHexagon.setPoint(0, vect[i].top);
55:         filledHexagon.setPoint(1, vect[i].left1);
56:         filledHexagon.setPoint(2, vect[i].left2);
57:         filledHexagon.setPoint(3, vect[i].bottom);
58:         filledHexagon.setPoint(4, vect[i].right1);
59:         filledHexagon.setPoint(5, vect[i].right2);
60:         filledHexagon.setFillColor(sf::Color(rand() % 255, rand() % 255, rand()%255,
            rand() % 255));
61:         target.draw(filledHexagon, states);
62:     }
63: }
```

original.hpp

```
1: #ifndef HEXAGON_H
2: #define HEXAGON_H
3: #include <SFML/Graphics.hpp>
4: using namespace std;
5: struct hexagon {
6:     sf::Vector2f top, left1, left2, bottom, right1, right2;
7: };
```

```
 8:  class Hexagon: public sf::Drawable {
 9:  public:
10:      //Constructor that takes size and depth
11:      Hexagon(int side, int depth);
12:      //Constructor that takes six points and depth
13:      void hex_tree(sf::Vector2f top, sf::Vector2f left1, sf::Vector2f left2, sf ::Vector2f
bottom, sf::Vector2f right1, sf::Vector2f right2, int depth);
14:      void virtual draw(sf::RenderTarget &target, sf::RenderStates states) const;
15:  private:
16:      sf::Vector2f _top, _left1, _left2, _bottom, _right1, _right2;
17:      int _depth;
18:  };
19:
20: #endif
```

PS2: LINEAR FEEDBACK SHIFT REGISTER AND IMAGE ENCODING

PS2a: LINEAR FEEDBACK SHIFT REGISTER AND UNIT TESTING

Assignment Summary:

The assignment is to write a program that produces pseudo-random bits by simulating a linear feedback shift register, and then use it to implement a simple form of encryption for digital pictures.

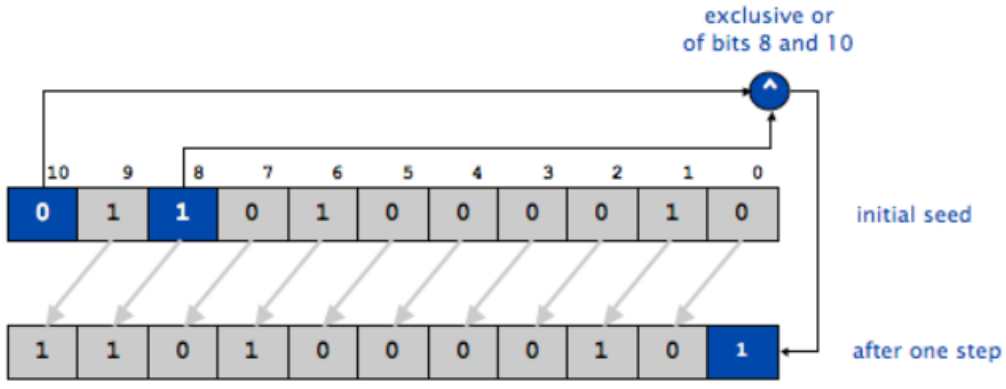
The class LFSR (Linear Feedback Shift Register) has the constructor that accepts the initial seed (the sequence of bits 0 or 1) and tap position (Line 6-8, LSFR.cpp).

The bits will be shifted to the left 1 position. The step() function is to simulate one step and return the new bit as 0 or 1 by exclusive or of the tap bit and the first bit of the sequence (Line 11-25, LSFR.cpp). The generate (int k) function simulates k steps and return k-bit integer by calling the step() function k time (Line 27-35, LSFR.cpp).

Algorithms/Data Structures:

The algorithm of the program is to take the first bit exclusive or with the tap position and store the result to the end of the sequence.

This example shows how to take one step of a sequence of 11-bit LFSR and the tap position 8.



One step of an 11-bit LFSR with initial seed 01101000010 and tap at position 8

Accomplishment:

Through this assignment, I have learnt how to create the BOOST_AUTO_TEST_CASE with the initial register bits and position, we can test whether the function step() prints out the vacated bit is equal to our computation. Simultaneously, it tests if the function generate() can compute exactly the integer when we simulate k steps.

```
daphne@daphne-VirtualBox:~/ps2a$ ./ps2a
Running 4 test cases...

*** No errors detected
daphne@daphne-VirtualBox:~/ps2a$
```

Makefile

```
1:  CC = g++
2:  CFLAGS = -Wall -Werror -ansi -pedantic -lboost_unit_test_framework
3:  LFLAGS = -lsfml-window -lsfml-graphics -lsfml-system -lboost_unit_test_framework
4:
5:  .PHONY: all clean
6:
7:  all: ps2a
8:
9:  ps2a: test.o LFSR.o
10:     $(CC) test.o LFSR.o -o ps2a $(LFLAGS)
11: test.o: test.cpp LFSR.hpp
12:     $(CC) -c $< $(CFLAGS)
13: LFSR.o: LFSR.cpp LFSR.hpp
14:     $(CC) -c $< $(CFLAGS)
15:
16: clean:
17:     rm LFSR.o test.o ps2a
```

test.cpp

```
1:  #include <iostream>
2:  #include <string>
3:  #include "LFSR.hpp"
4:  #include <boost/test/unit_test.hpp>
5:  #define BOOST_TEST_DYN_LINK
6:  #define BOOST_TEST_MODULE Main
7:  BOOST_AUTO_TEST_CASE(fiveBitsTapAtTwo) {
8:      LFSR l("00111", 2);
9:      BOOST_REQUIRE(l.step() == 1);
10:     BOOST_REQUIRE(l.step() == 1);
11:     BOOST_REQUIRE(l.step() == 0);
12:     BOOST_REQUIRE(l.step() == 0);
13:     BOOST_REQUIRE(l.step() == 0);
14:     BOOST_REQUIRE(l.step() == 1);
15:     BOOST_REQUIRE(l.step() == 1);
16:     BOOST_REQUIRE(l.step() == 0);
17:     LFSR l2("00111", 2);
```

```
18:     BOOST_REQUIRE(l2.generate(8) == 198);
19: }
20: BOOST_AUTO_TEST_CASE(sevenBitsTapAtFour) {
21:     LFSR lfsr("0110011",4);
22:     BOOST_REQUIRE(lfsr.step()==1);
23:     BOOST_REQUIRE(lfsr.step()==1);
24:     BOOST_REQUIRE(lfsr.step()==1);
25:     BOOST_REQUIRE(lfsr.step()==1);
26:     BOOST_REQUIRE(lfsr.step()==1);
27:     BOOST_REQUIRE(lfsr.step()==0);
28:     BOOST_REQUIRE(lfsr.step()==0);
29:     BOOST_REQUIRE(lfsr.step()==0);
30:     BOOST_REQUIRE(lfsr.step()==0);
31:     BOOST_REQUIRE(lfsr.step()==0);
32:     BOOST_REQUIRE(lfsr.step()==1);
33:     LFSR lfsr2("0110011",4);
34:     BOOST_REQUIRE(lfsr2.generate(5)==31);
35: }
36: BOOST_AUTO_TEST_CASE(thirteenBitsTapAtEleven) {
37:     LFSR lfsr("00101000110010",11);
38:     BOOST_REQUIRE(lfsr.step()==1);
39:     BOOST_REQUIRE(lfsr.step()==0);
40:     BOOST_REQUIRE(lfsr.step()==0);
41:     BOOST_REQUIRE(lfsr.step()==0);
42:     BOOST_REQUIRE(lfsr.step()==1);
43:     BOOST_REQUIRE(lfsr.step()==0);
44:     BOOST_REQUIRE(lfsr.step()==1);
45:     BOOST_REQUIRE(lfsr.step()==1);
46:     BOOST_REQUIRE(lfsr.step()==1);
47:     LFSR lfsr2("00101000110010",11);
48:     BOOST_REQUIRE(lfsr2.generate(7)==69);
49: }
50: BOOST_AUTO_TEST_CASE(fourBitsTapAtTwo) {
51:     LFSR lfsr("0111",2);
52:     BOOST_REQUIRE(lfsr.step()==1);
53:     BOOST_REQUIRE(lfsr.step()==0);
54:     BOOST_REQUIRE(lfsr.step()==0);
55:     BOOST_REQUIRE(lfsr.step()==0);
56:     BOOST_REQUIRE(lfsr.step()==1);
57:     LFSR lfsr2("0111",2);
58:     BOOST_REQUIRE(lfsr2.generate(3)==4);
59:     }
```

LFSR.cpp

```
#include <iostream>
#include <cmath>
#include "LFSR.hpp"
//constructor to create LFSR with the given initial seed and tap
//it takes a String argument whose characters are a sequence of 0s and 1s
LFSR::LFSR(std::string seed, int t) {
    Seed = seed;
    tap = t;
}
//simulate one step and return the new bit as 0 or 1
int LFSR::step() {
    int length = Seed.length();
    int bit = Seed[0]-'0';
    int T = Seed[length-tap-1]-'0';
    if (bit == T) {
        bit = 0;
    } else {
        bit=1;
    }
    for (int i=0; i<length-1; i++) {
        Seed[i] = Seed[i+1];
    }
    Seed[length-1] = bit + 48;
    return bit;
}
//simulate k steps and return k-bit integer
int LFSR::generate(int k) {
    int number = 0;
    for (int i=k; i>0; i--) {
        if (step()==1) {
            number += pow(2, i-1);
        }
    }
    return number;
}
std::ostream& operator<<(std::ostream& out, LFSR& lsfr) {
    out << lsfr.Seed;
    return out;
    }
```

LFSR.hpp

```
 1: #include <iostream>
 2: class LFSR {
 3: public:
 4:     LFSR(std::string seed, int t);
 5:     int step();
 6:     int generate(int k);
 7:     friend std::ostream& operator<<(std::ostream& out, LFSR& lsfr);
 8: private:
 9:     std::string Seed;
10:     int tap;
11: };
```

PS2: LINEAR FEEDBACK SHIFT REGISTER AND IMAGE ENCODING

PS2b: ENCODING IMAGES WITH LFSR

Assignment Summary:

This assignment is the part 2 of the LSFR (Linear Feedback Shift Register) with creating the PhotoMagic.cpp which takes the image input file and encrypt it. The method of this transforming is based on extracting the red, green and blue components of the color (each component is an integer between 0 and 255). Then, xor the red/blue/green component with a newly-generated 8-bit integer.

Algorithms/Data Structures:

The assignment changes color values of the image input. We can both encrypt and decrypt the image by using the LSFR – exclusive or (Line 6-19, PhotoMagic.cpp).

Accomplishment:

Through this assignment, I have learnt how to encrypt or decrypt the input image by using XOR bitwise operator. The program takes 4 arguments from command line as below:

./PhotoMagic decode.png encode.png 01101000010100010000 16

where first and second argument is input and output image, respectively

third argument is a string of seed

and the last argument is tap position

Makefile

```
1:  CC = g++
2:  CFLAGS = -Wall -ansi -pedantic -Werror -std=c++0x -lboost_unit_test_framework
3:  LFLAGS = -lsfml-window -lsfml-graphics -lsfml-system -lboost_unit_test_framework
4:
5:  .PHONY: all clean
6:
7:  all: PhotoMagic
8:
9:  PhotoMagic: PhotoMagic.o LFSR.o
10:     $(CC) PhotoMagic.o LFSR.o -o PhotoMagic $(LFLAGS)
11: PhotoMagic.o: PhotoMagic.cpp LFSR.hpp
12:     $(CC)  -c $< $(CFLAGS)
13: LFSR.o: LFSR.cpp LFSR.hpp
14:     $(CC)  -c $< $(CFLAGS)
15:
16: clean:
17:     rm *.o PhotoMagic
```

PhotoMagic.cpp

```
1:  #include <SFML/System.hpp>
2:  #include <SFML/Window.hpp>
3:  #include <SFML/Graphics.hpp>
4:  #include <iostream>
5:  #include "LFSR.hpp"
6:  sf::Image createOutput(sf::Image image, LFSR lfsr) {
7:      sf::Vector2u size = image.getSize();
8:      sf::Color p;
9:      for (unsigned x = 0; x < size.x; x++) {
10:         for (unsigned y = 0; y < size.y; y++) {
11:             p = image.getPixel(x, y);
12:             p.r ^= lfsr.generate(8);
13:             p.g ^= lfsr.generate(8);
14:             p.b ^= lfsr.generate(8);
15:             image.setPixel(x, y, p);
16:         }
17:     }
18:     return image;
19: }
20: int main(int argc, char *argv[]) {
21:     sf::Image input;
22:     if (!input.loadFromFile(argv[1])) {
```

```
23:         return -1;
24:     }
25:     sf::Image output;
26:     if (!output.loadFromFile(argv[1])) {
27:         return -1;
28:     }
29:     LFSR lfsr(argv[3],atoi(argv[4]));
30:     output = createOutput(output, lfsr);
31:     sf::Vector2u size = input.getSize();
32:     //create window1 for Input
33:     sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "Input");
34:     //create window2 for Output
35:     sf::RenderWindow window2(sf::VideoMode(size.x, size.y), "Output");
36:     //create texture and sprite for input
37:     sf::Texture texture;
38:     texture.loadFromImage(input);
39:     sf::Sprite sprite;
40:     sprite.setTexture(texture);
41:     //create texture and sprite for output
42:     sf::Texture output_texture;
43:     output_texture.loadFromImage(output);
44:     sf::Sprite output_sprite;
45:     output_sprite.setTexture(output_texture);
46:     while (window1.isOpen() && window2.isOpen()) {
47:         sf::Event event;
48:         while (window1.pollEvent(event)) {
49:             if (event.type == sf::Event::Closed)
50:                 window1.close();
51:         }
52:         while (window2.pollEvent(event)) {
53:             if (event.type == sf::Event::Closed)
54:                 window2.close();
55:         }
56:         window1.clear();
57:         window1.draw(sprite);
58:         window1.display();
59:         window2.clear();
60:         window2.draw(output_sprite);
61:         window2.display();
62:     }
63:     //create output file
64:     if (!output.saveToFile(argv[2]))
```

```
65:         return -1;
66:     return 0;
67: }
```

LFSR.cpp

```
 1: #include <iostream>
 2: #include <cmath>
 3: #include "LFSR.hpp"
 4: //constructor to create LFSR with the given initial seed and tap
 5: //it takes a String argument whose characters are a sequence of 0s and 1s
 6: LFSR::LFSR(std::string seed, int t) {
 7:     Seed = seed;
 8:     tap = t;
 9: }
10: //simulate one step and return the new bit as 0 or 1
11: int LFSR::step() {
12:     int length = Seed.length();
13:     int bit = Seed[0]-'0';
14:     int T = Seed[length-tap-1]-'0';
15:     if (bit == T) {
16:         bit = 0;
17:     } else {
18:         bit=1;
19:     }
20:     for (int i=0; i < length-1; i++) {
21:         Seed[i] = Seed[i+1];
22:     }
23:     Seed[length-1] = bit + 48;
24:     return bit;
25: }
26: //simulate k steps and return k-bit integer
27: int LFSR::generate(int k) {
28:     int number = 0;
29:     for (int i=k; i>0; i--) {
30:         if (step()==1) {
31:             number += pow(2, i-1);
32:         }
33:     }
34:     return number;
35: }
36: std::ostream& operator<<(std::ostream& out, LFSR& lsfr) {
37:     out << lsfr.Seed;
```

```
38:     return out;
39: }
```

LFSR.hpp

```
 1: #include <iostream>
 2: class LFSR {
 3: public:
 4:     LFSR(std::string seed, int t);
 5:     int step();
 6:     int generate(int k);
 7:     friend std::ostream& operator<<(std::ostream& out, LFSR& lsfr);
 8: private:
 9:     std::string Seed;
10:     int tap;
11: };
```

PS3: N-BODY SIMULATION

PS3a: DESIGN A PROGRAM THAT LOADS AND DISPLAYS A STATIC UNIVERSE

Assignment Summary:

This assignment is to simulate the universe by reading the input text file which contains the information of a universe and calculating their positions. The first value is an integer N which represents the radius of the universe; therefore, 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 name of an image file used to display the particle.

The example of the input text file as below:

```
% more planets.txt
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
```

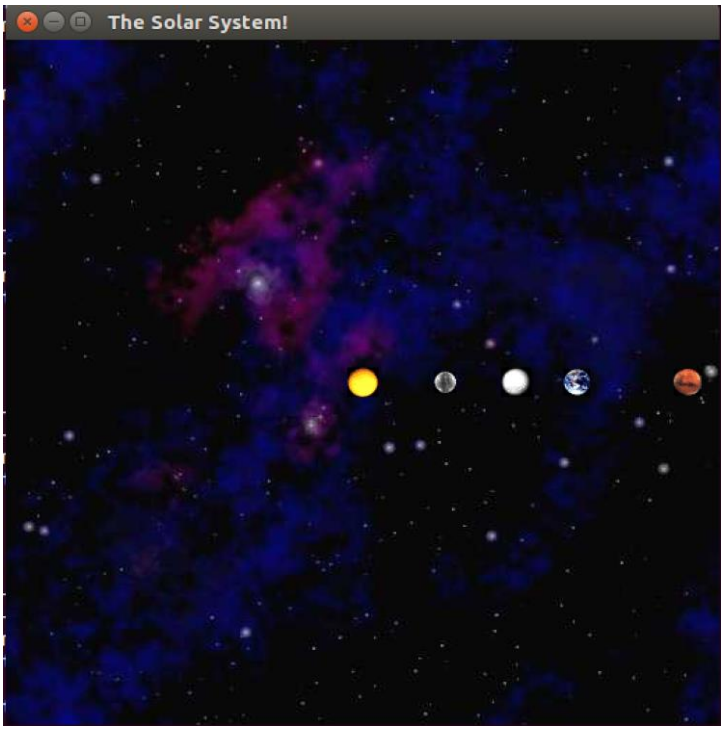
Algorithms/Data Structures:

I have created the array called Planets[] to store 6 initial values as the information about the universe (Line 20, main.cpp). Based on these values, the program draws the planets in the universe (Line 32-35, main.cpp) by taking the window size divided by the scale multiplied by the position (Line 35-37, nbody.cpp).

Besides, the program uses the virtual void method draw() and the implementation of operator >> for inputs ((Line 22-33, nbody.cpp).

Accomplishment:

Through this assignment, I have learnt how to read the input file and implement the input stream operator >> as well as use it to load parameter data into an object.



Makefile

```
1:  CC = g++
2:  CFLAGS = -Wall -ansi -pedantic -Werror -std=c++0x -lboost_unit_test_framework
3:  LFLAGS = -lsfml-window -lsfml-graphics -lsfml-system
4:
5:  all: NBody
6:
7:  NBody: nbody.o main.o
8:      $(CC) nbody.o main.o $(LFLAGS) -o NBody
9:  main.o: main.cpp nbody.cpp
10:     $(CC) -c main.cpp $(CFLAGS)
11: nbody.o: nbody.cpp nbody.hpp
12:     $(CC) -c nbody.cpp $(CFLAGS)
13:
14: clean:
15:     rm *.o NBody
```

main.cpp

```
1:  #include "nbody.hpp"
2:  int main(int argc, char *argv[]) {
3:      Body **Planets = new Body*[4];
4:      int numOfPlanets= 0;
```

```
    double scale = 0;
    //create the background image
    sf::Image background;
    if (!background.loadFromFile("starfield.jpg")) {
        return -1;
    }
    sf::Vector2u size = background.getSize();
    sf::RenderWindow window(sf::VideoMode(size.x, size.y), "The Solar System!");
    sf::Texture texture;
    texture.loadFromImage(background);
    sf::Sprite sprite;
    sprite.setTexture(texture);
    //planets.txt is the input file
    std::cin >> numOfPlanets >> scale;
    for (int i = 0; i < numOfPlanets; i++) {
        Planets[i]= new Body(0.0,0.0,0.0,0.0,0.0,"o");
        std::cin >> *Planets[i];
    }
    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window.close();
        }
        window.clear();
        window.draw(sprite);
        //drawing the planets
        for (int i=0; i < numOfPlanets; i++) {
            (*Planets[i]).updatePosition(scale, size.x*1/2, size.y*1/2);
            window.draw(*Planets[i]);
        }
        window.display();
    }
    return 0;
}
```

nbody.cpp

```
#include "nbody.hpp"
Body::Body (double x_pos, double y_pos, double xSpeed, double ySpeed, double mass,
std::string image) {}
void Body:: draw (sf::RenderTarget& target, sf::RenderStates states) const {
    target.draw(sprite,states);
```

```
 5:  }
 6:  void Body:: setPosition (double x, double y) {
 7:      X = x;
 8:      Y = y;
 9:  }
10:  void Body:: setMass (double mass) {
11:      Mass = mass;
12:  }
13:  double Body::getMass() {
14:      return Mass;
15:  }
16:  double Body:: get_x() const {
17:      return X;
18:  }
19:  double Body:: get_y() const {
20:      return Y;
21:  }
22:  std::istream& operator>> (std::istream& in, Body& planet) {
23:        in >> planet.X
24:           >> planet.Y
25:           >> planet.xSpeed
26:           >> planet.ySpeed
27:           >> planet.Mass
28:           >> planet.Image;
29:      planet.ImageOfPlanet.loadFromFile(planet.Image);
30:      planet.texture.loadFromImage(planet.ImageOfPlanet);
31:      planet.sprite.setTexture(planet.texture);
32:      return in;
33:  }
34:  void Body:: updatePosition(double scale, int window_x, int window_y) {
35:      sf::Vector2u size = ImageOfPlanet.getSize();
36:      int x = window_x - size.x*1/2 + X/scale*window_x;
37:      int y = window_y - size.y*1/2 + Y/scale*window_y;
38:      sprite.setPosition(x,y);
39:      }
```

nbody.hpp

```
 1:  #include <SFML/System.hpp>
 2:  #include <SFML/Window.hpp>
 3:  #include <SFML/Graphics.hpp>
 4:  #include <iostream>
 5:  class Body : public sf::Drawable {
```

```
 6:  private:
 7:      double X;
 8:      double Y;
 9:      double xSpeed;
10:      double ySpeed;
11:      double Mass;
12:      std::string Image;
13:      sf::Image ImageOfPlanet;
14:      sf::Texture texture;
15:      sf::Sprite sprite;
16:  public:
17:      Body(double x_pos, double y_pos, double xSpeed, double ySpeed, double mass,
         std::string image);
18:      void setPosition(double x_pos, double y_pos);
19:      void setMass(double mass);
20:      double getMass();
21:      double get_x() const;
22:      double get_y() const;
23:      virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
24:      void updatePosition(double scale,int window_x, int window_y);
25:      friend std::istream& operator>> (std::istream& in, Body& planet);
26:  };
```

PS3: N-BODY SIMULATION

PS3b: USING NEWTON'S LAWS OF PHYSICS, ANIMATE THE UNIVERSE

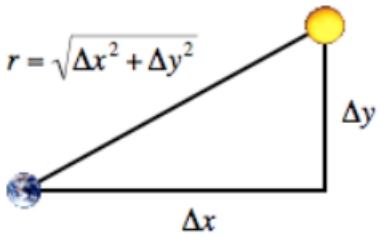
Assignment Summary:

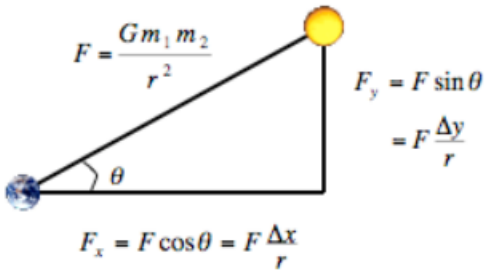
This assignment is the part 2 of the N-Body Simulation – loading Universe files and displaying the static universe. In this assignment, we keep simulating the universe by reading the input text file which contains the information for a particular universe, but simultaneously, we update their positions and calculate velocities to make them move around the sun.

Algorithms/Data Structures:

I have created the array called Planets[] to store 6 initial values as the information about the universe (Line 14, main.cpp). Based on these values, the program draws the planets in the universe (Line 43-45, main.cpp) by taking the window size divided by the scale multiplied by the position (Line 24-29, nbody.cpp). Besides, the program uses the virtual void method draw() and the implementation of operator >> for inputs.

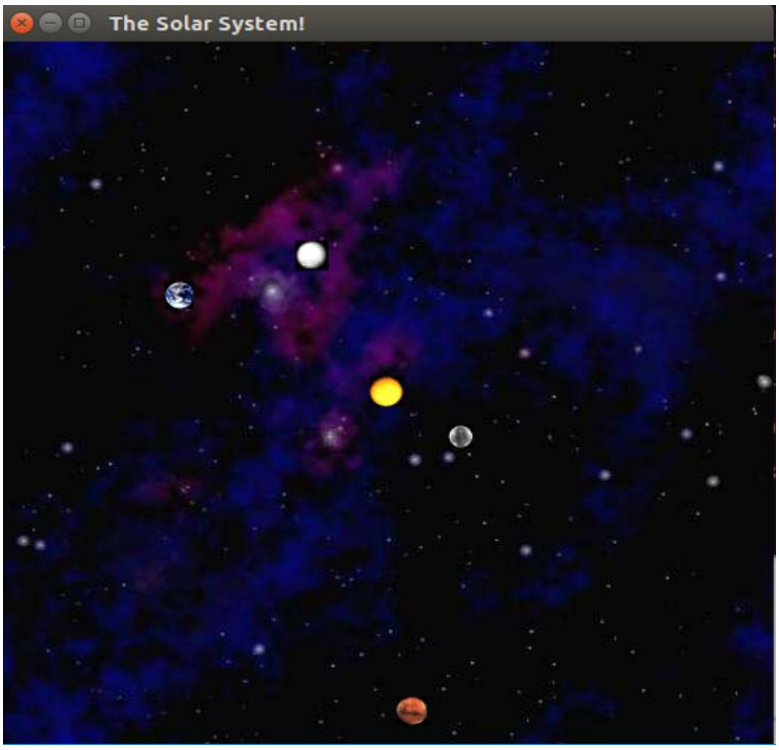
In part 3b, we have applied the Physics theory – Newton's law of universal gravitation. We compute the gravitational force between two particles, then break up the force into its x- and y-component (F_x, F_y) (as the image below) (Line 30-41, nbody.cpp). Next, we calculate the planet's new velocity and position to make it move in the universe.





Accomplishment:

Through this assignment, I have learnt how to read the input file and implement the input stream operator >> as well as use it to load parameter data into an object. Besides, I can apply the Physics theory into calculating the force and velocity of planets around the sun. I have also inserted the music while the planets are moving in the universe (Line 21-25, main.cpp).



Makefile

```
1:  CC = g++
2:  CFLAGS = -Wall -ansi -pedantic -Werror -std=c++0x -lboost_unit_test_framework
3:  LFLAGS = -lsfml-window -lsfml-graphics -lsfml-audio -lsfml-system
4:
5:  all: NBody
6:
7:  NBody: nbody.o main.o
8:      $(CC) nbody.o main.o $(LFLAGS) -o NBody
9:  main.o: main.cpp nbody.cpp
10:     $(CC) -c main.cpp $(CFLAGS)
11: nbody.o: nbody.cpp nbody.hpp
12:     $(CC) -c nbody.cpp $(CFLAGS)
13:
14: clean:
15:     rm *.o NBody
```

main.cpp

```
1:  #include "nbody.hpp"
2:  int main(int argc, char *argv[]) {
3:      Body **Planets = new Body*[4];
4:      int numOfPlanets= 0;
```

```
    double scale = 0;
    double t = 0;
    double time = (double)atoi(argv[1]);
    double DTime=(double)atoi(argv[2]);
    sf::Music music;
    sf::Image background;
    //planets.txt is the input file
    std::cin >> numOfPlanets >> scale;
    for (int i = 0; i<numOfPlanets; i++) {
        Planets[i]= new Body(0.0,0.0,0.0,0.0,0.0,"o");
        std::cin >> *Planets[i];
        (*Planets[i]).setnum_planets(numOfPlanets);
    }
    if (!background.loadFromFile("starfield.jpg")) {
        return -1;
    }
    if (!music.openFromFile("music.ogg")) {
        return -1;
    }
    music.setLoop(true);
    music.play();
    sf::Vector2u size = background.getSize();
    sf::RenderWindow window(sf::VideoMode(size.x, size.y), "The Solar System!");
    sf::Texture texture;
    sf::Sprite sprite;
    texture.loadFromImage(background);
    sprite.setTexture(texture);
    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window.close();
        }
        if(t<time) {
            window.clear();
            window.draw(sprite);
            (*Planets[0]).step(DTime,Planets,0);
            //drawing the planets
            for (int i=0; i<numOfPlanets; i++) {
                (*Planets[i]).updatePosition(scale, size.x/2, size.y/2);
                window.draw(*Planets[i]);
            }
```

```
47:             t=t+DTime;
48:         }
49:         window.display();
50:     }
51:     return 0;
52: }
```

nbody.cpp

```
 1: #include "nbody.hpp"
 2: Body::Body(double x,double y,double xSpeed, double ySpeed,double mass, std:: string
image) {}
 3: void Body:: draw(sf::RenderTarget& target, sf::RenderStates states) const {
 4:     target.draw(sprite,states);
 5: }
 6: void Body:: setPosition(double x, double y) {
 7:     X = x;
 8:     Y = y;
 9: }
10: void Body::setnum_planets(int num){num_planets = num;}
11: int Body::getnum_planets(){return num_planets;}
12: std::istream& operator>> (std::istream& in, Body& planet) {
13:     in >> planet.X
14:          >> planet.Y
15:          >> planet.X_Speed
16:          >> planet.Y_Speed
17:          >> planet.Mass
18:          >> planet.Image;
19:     planet.ImageOfPlanet.loadFromFile(planet.Image);
20:     planet.texture.loadFromImage(planet.ImageOfPlanet);
21:     planet.sprite.setTexture(planet.texture);
22:     return in;
23: }
24: void Body:: updatePosition(double scale,int window_x, int window_y) {
25:     sf::Vector2u size = ImageOfPlanet.getSize();
26:     int x = window_x-size.x/2+X/scale*window_x;
27:     int y = window_y-size.y/2+Y/scale*window_y;
28:     sprite.setPosition(x,y);
29: }
30: void Body:: step(double time,Body **Planets,int planet){
31:     double r, F, FX, FY;
32:     for (int i=0;i<getnum_planets();i++) {
33:         if (planet!=i) {
```

```
34:             r = sqrt(pow(X-(*Planets[i]).X,2)+pow(Y-(*Planets[i]).Y,2));
35:             F = (6.67e-11)*(*Planets[i]).Mass*Mass/pow(r,2);
36:             FX = F*(X-(*Planets[i]).X)/r;
37:             FY = F*(Y-(*Planets[i]).Y)/r;
38:             X_Speed = X_Speed+time*FX/Mass;
39:             Y_Speed = Y_Speed+time*FY/Mass;
40:         }
41:     }
42:     if (planet<getnum_planets()-1) {
43:         (*Planets[planet+1]).step(time, Planets, planet+1);
44:     }
45:     X -=(time*X_Speed);
46:     Y -=(time*Y_Speed);
47: }
```

nbody.hpp

```
 1: #include <SFML/System.hpp>
 2: #include <SFML/Window.hpp>
 3: #include <SFML/Graphics.hpp>
 4: #include <SFML/Audio.hpp>
 5: #include <cmath>
 6: #include <ctime>
 7: #include <iostream>
 8: class Body : public sf::Drawable {
 9: private:
10:     double X;
11:     double Y;
12:     double X_Speed;
13:     double Y_Speed;
14:     double Mass;
15:     double num_planets;
16:     std::string Image;
17:     sf::Texture texture;
18:     sf::Image ImageOfPlanet;
19:     sf::Sprite sprite;
20: public:
21:     Body(double x,double y,double xSpeed, double ySpeed,double mass, std::stri ng image);
22:     void setPosition(double x_pos, double y_pos);
23:     void setnum_planets(int num);
24:     int getnum_planets();
25:     virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
26:     void updatePosition(double scale,int window_x, int window_y);
```

```
27:     friend std::istream& operator>> (std::istream& in, Body& planet);
28:     void step(double time,Body **Planets,int planet);
29: };
```

PS4: EDIT DISTANCE

Assignment Summary:

The goal of this assignment is to measure the similarity of two genetic sequences by their edit distance. We align the two sequences, and we are permitted to insert gaps in either sequence; simultaneously, we pay a penalty for each gap that we insert and for each part of characters that mismatch in the final alignment. We produce a numerical score according to the following table:

operation	*cost*
insert a gap	2
align two characters that mismatch	1
align two characters that match	0

Algorithms/Data Structures:

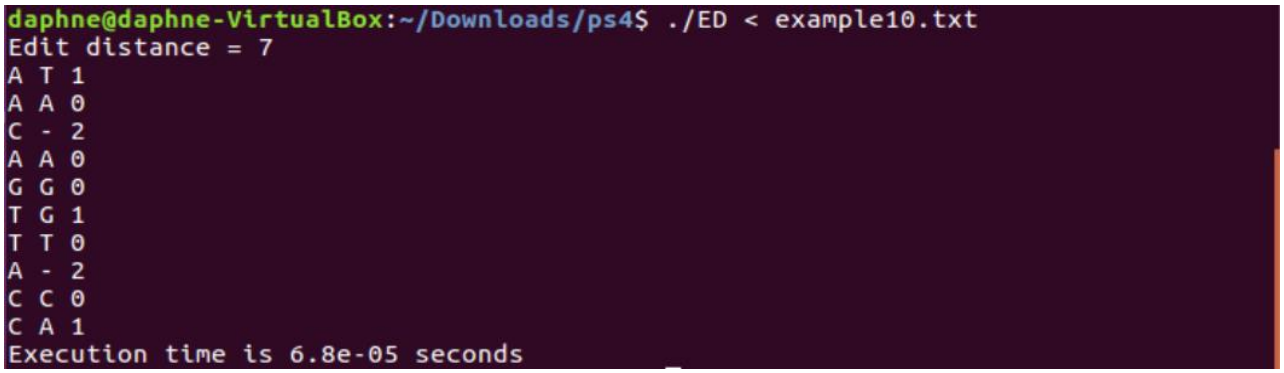
The edit-distance is the score of the best possible alignment between the two genetic sequences over all possible alignments. We have used the dynamic programming approach for this assignment (or Needleman and Wunsch). We break up a large computational problem into smaller subproblems, store the answers to those smaller subproblems, and eventually, use the stored answers to solve the original problem.

We have used a nested loop that calculates opt[i][j] in the "right" order so that opt[i+1][j+1], opt[i+1][j], and opt[i][j+1] are all computed before we try to compute opt[i][j]. Then, we have tried to recover the optimal alignment itself.

Accomplishment:

In this assignment, I have used the function int ED::OptDistance() (Line18, EditDistance.cpp) to create the table of two strings and fill in all the cells in that table with suitable number. Also, I have used 2 more functions ED::penalty(char a, char b) (used to find the number that position opt[i+1][j+1] should add is 0 or 1) (Line 12-14, EditDistance.cpp) and ED::min(int a, int b, int c) (Line 15-17, EditDistance.cpp) (used to find the minimum values in 3 numbers). Finally, this function returns the optimal edit distance at the position opt[0][0].

Next, I have used the function std::string ED::Alignment() () (Line35, EditDistance.cpp) to print out the alignment for 2 input strings. I followed exactly the rules described in Princeton, part "Recovering the alignment itself" to find the appropriate letter, gap and cost.



Makefile

```
1:  CC = g++
2:  CFLAGS = -g -Wall -ansi -pedantic -Werror -std=c++0x -lboost_unit_test_framework
3:  LFLAGS = -g -lsfml-window -lsfml-graphics -lsfml-system -std=c++0x -
lboost_unit_test_framework
4:
5:  .PHONY: all clean
6:
7:  all: ED
8:
9:  ED: main.o EditDistance.o
10:     $(CC) main.o EditDistance.o -o ED $(LFLAGS)
11: main.o: main.cpp EditDistance.hpp
12:     $(CC) -c $< $(CFLAGS)
13: EditDistance.o: EditDistance.cpp EditDistance.hpp
14:     $(CC) -c $< $(CFLAGS)
15:
16: clean:
17:     rm EditDistance.o main.o ED
```

main.cpp

```
1:  #include <SFML/System.hpp>
2:  #include "EditDistance.hpp"
3:  int main(){
4:      sf::Clock clock;
5:      sf::Time time;
6:      std::string string1, string2;
7:      std::cin >> string1 >> string2;
8:      ED edit_distance(string1,string2);
9:      int distance = edit_distance.OptDistance();
10:     std::cout << "Edit distance = " << distance << "\n";
11:     std::string result = edit_distance.Alignment();
```

```
12:     std::cout << result;
13:     time = clock.getElapsedTime();
14:     std::cout << "Execution time is " << time.asSeconds() << " seconds \n";
15: }
```

EditDistance.cpp

```
 1: #include "EditDistance.hpp"
 2: ED::ED(std::string s1, std::string s2) {
 3:     string1 = s1;
 4:     string2 = s2;
 5:     size_string1 = string1.length();
 6:     size_string2 = string2.length();
 7:     opt = new int*[size_string1 + 1];
 8:     for (int i=0; i<(size_string1 + 1); i++) {
 9:         opt[i]= new int[size_string2 + 1];
10:     }
11: }
12: int ED::penalty(char a, char b) {
13:     return (a == b ? 0 : 1);
14: }
15: int ED::min(int a, int b, int c) {
16:     return std::min(std::min(a, b),c);
17: }
18: int ED::OptDistance() {
19:     int score = 0;
20:     for (int i=size_string1; i>=0; i--) { //fill in the column base
21:         opt[i][size_string2] = score;
22:         score += 2;
23:         for (int j = size_string2; j>=0; j--) {//fill in the row base
24:             opt[size_string1][j-1] = opt[size_string1][j]+2;
25:         }
26:     }
27:     //fill in all the table
28:     for (int i = size_string1-1; i>=0; i--) {
29:         for (int j = size_string2-1; j>=0; j--) {
30:             opt[i][j] = min(opt[i+1][j+1] + penalty(string1[i],string2[j]), opt[i] [j+1]+2,
opt[i+1][j]+2);
31:         }
32:     }
33:     return opt[0][0];
34: }
35: std::string ED::Alignment() {
```

```
36:     int i = 0, j = 0, k= 0;
37:     std::string align;
38:     std::string str="s s 0\n";
39:     k = (size_string1 > size_string2) ? size_string1 : size_string2;
40:     for (int n=0; n<k; n++) {
41:        if (string1[i]!=string2[j] && opt[i][j] == opt[i+1][j+1]+1) {
42:           str[0]=string1[i];
43:           str[2]=string2[j];
44:           str[4]='1';
45:           align += str;
46:           i++;
47:           j++;
48:        } else if (string1[i]==string2[j] && opt[i][j] == opt[i+1][j+1]) {
49:           str[0]=string1[i];
50:           str[2]=string2[j];
51:           str[4]='0';
52:           align += str;
53:           i++;
54:           j++;
55:        } else if (opt[i][j] == opt[i][j+1] + 2) {
56:           str[0]='-';
57:           str[2]=string2[j];
58:           str[4]='2';
59:           align += str;
60:           j++;
61:        } else if (opt[i][j] == opt[i+1][j] + 2){
62:           str[0]=string1[i];
63:           str[2]='-';
64:           str[4]='2';
65:           align += str;
66:           i++;
67:        }
68:     }
69:     while ((i < size_string1) && (j < size_string2)) {
70:        str[0] = string1[i];
71:        str[2] = string2[j];
72:        str[4] = '0';
73:        align += str;
74:        i++;
75:        j++;
76:     }
77:     while (i < size_string1) {
```

```
78:         str[0]=string1[i];
79:         str[2]='-';
80:         str[4]='2';
81:         align += str;
82:         i++;
83:     }
84:     while (j < size_string2) {
85:         str[0] = '-';
86:         str[2] = string2[j];
87:         str[4] = '2';
88:         align += str;
89:         j++;
90:     }
91:     return align;
92: }
```

EditDistance.hpp

```
 1: #include <iostream>
 2: #include <string>
 3: class ED {
 4: private:
 5:     std::string string1;
 6:     std::string string2;
 7:     int size_string1;
 8:     int size_string2;
 9:     int **opt;
10: public:
11:     ED(std::string s1, std::string s2);
12:     int penalty(char a, char b);
13:     int min(int a, int b, int c);
14:     int OptDistance();
15:     std::string Alignment();
16: };
```

PS5: RING BUFFER AND GUITAR HERO

PS5a: RING BUFFER WITH CPPLINT, TESTING, AND EXEPTIONS

Assignment Summary:

This assignment is to write a program that implements the ring buffer that will hold the guitar string position data. Also, we write the test functions to run some tests with the code and throw the exception handling.

The program also uses the Boost functions BOOST_REQUIRE_THROW and BOOST_REQUIRE_NO_THROW to verify that the code properly throws the specified exceptions when appropriate.

Using the cpplint – Google's style to style-check the file by running cpplint.py.

Algorithms/Data Structures:

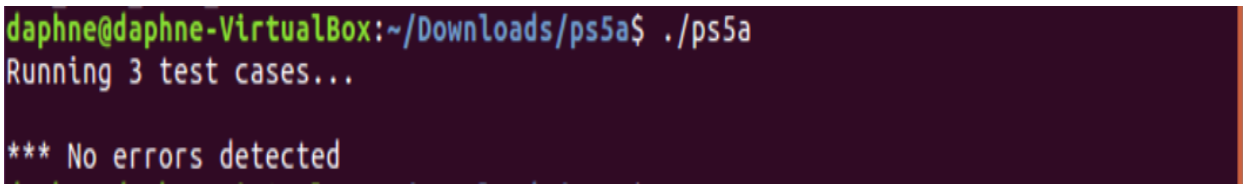
I have used the array ringBuffer – type 16-bit integer to represent the guitar string position data. It includes the functions enqueue (Line 39, RingBuffer.cpp), dequeue (Line 58, RingBuffer.cpp), to add item to the array or delete and return it from the front.

To enqueue to the ring, if the ring is not full and the size = 0, we add the item x to the ring, and it becomes the head. Otherwise, if the size is not equal to 0, depending on the tail is 0 or not, we add the item x to the end of the ring. The size of the ring increments.

To dequeue from the ring, if the ring is not empty, we delete the item from the front and return its value. Simultaneously, the size decrements.

Accomplishment:

Through this assignment, I have learnt how to create the buffer, enqueue and dequeue the ring. Besides, I have to throw exception when it catches the error, such as: enqueue to a full ring (Line 41, RingBuffer.cpp), or dequeue (Line 61, RingBuffer.cpp)/peek (Line 78, RingBuffer.cpp) from an empty ring.



Makefile

```
1:  CC = g++
2:  CFLAGS = -Wall -Werror -pedantic -ansi -std=c++0x -lboost_unit_test_framework
3:  LFLAGS = -lsfml-window -lsfml-graphics -lsfml-system -lboost_unit_test_framework
4:
5:  .PHONY: all clean
6:  all: ps5a
7:
8:  ps5a: RingBuffer.o test.o
9: $(CC) RingBuffer.o test.o -o ps5a $(LFLAGS)
10:  RingBuffer.o: RingBuffer.cpp RingBuffer.hpp
11:      $(CC) -c $< $(CFLAGS)
12:  test.o: test.cpp RingBuffer.hpp
13:      $(CC) -c $< $(CFLAGS)
14:
15:  clean:
16:      rm RingBuffer.o test.o ps5a
```

test.cpp

```
1:  // Copyright 2018 Dangnhi Ngo
2:  #define BOOST_TEST_DYN_LINK
3:  #define BOOST_TEST_MODULE Main
4:  #include <boost/test/unit_test.hpp>
5:  #include <stdint.h>
6:  #include <iostream>
7:  #include <string>
8:  #include <exception>
9:  #include <stdexcept>
10:  #include "RingBuffer.hpp"
11:  BOOST_AUTO_TEST_CASE(RBcontructor) {
12:      // normal constructor
13:      BOOST_REQUIRE_NO_THROW(RingBuffer(100));
14:      // this should fail
15:      BOOST_REQUIRE_THROW(RingBuffer(0), std::exception);
16:      BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument);
17:  }
18:  BOOST_AUTO_TEST_CASE(RBenque_dequeue) {
19:      RingBuffer rb(100);
20:      rb.enqueue(2);
21:      rb.enqueue(1);
22:      rb.enqueue(0);
23:      BOOST_REQUIRE(rb.dequeue() == 2);
```

```
24:     BOOST_REQUIRE(rb.dequeue() == 1);
25:     BOOST_REQUIRE(rb.dequeue() == 0);
26:     BOOST_REQUIRE_THROW(rb.dequeue(), std::runtime_error);
27: }
28: BOOST_AUTO_TEST_CASE(RB_test_sample) {
29:     RingBuffer rb(2);
30:     BOOST_REQUIRE(rb.isEmpty() == 1);
31:     rb.enqueue(2);
32:     rb.enqueue(1);
33:     BOOST_REQUIRE_THROW(rb.enqueue(3), std::runtime_error);
34:     BOOST_REQUIRE(rb.isEmpty() == 0);
35:     BOOST_REQUIRE(rb.peek() == 2);
36:     BOOST_REQUIRE(rb.size() == 2);
37:     BOOST_REQUIRE(rb.isFull() == 1);
38:     BOOST_REQUIRE(rb.dequeue() == 2);
39:     BOOST_REQUIRE(rb.dequeue() == 1);
40:     BOOST_REQUIRE_THROW(rb.peek(), std::runtime_error);
41:     BOOST_REQUIRE_THROW(rb.dequeue(), std::runtime_error);
42:     }
```

RingBuffer.cpp

```
 1: // Copyright 2018 Dangnhi Ngo
 2: #include <stdint.h>
 3: #include <iostream>
 4: #include <exception>
 5: #include <stdexcept>
 6: #include "RingBuffer.hpp"
 7: // create an empty ring buffer, with given max capacity
 8: RingBuffer::RingBuffer(int capacity) : Capacity(capacity) {
 9:     if (Capacity <= 0) {
10:         throw std::invalid_argument("capacity must be greater than zero.");
11:     } else {
12:         ringBuffer = new int16_t[Capacity];
13:         head = Capacity-1;
14:         tail = Capacity-1;
15:         _size = 0;
16:     }
17: }
18: // return number of items currently in the buffer
19: int RingBuffer::size() {
20:     return _size;
21: }
```

```
22:  // is the buffer empty (size equals zero)?
23:  bool RingBuffer::isEmpty() {
24:      if (_size == 0) {
25:          return true;
26:      } else {
27:          return false;
28:      }
29:  }
30:  // is the buffer full (size equals capacity)?
31:  bool RingBuffer::isFull() {
32:      if (_size == Capacity) {
33:          return true;
34:      } else {
35:          return false;
36:      }
37:  }
38:  // add item x to the end
39:  void RingBuffer::enqueue(int16_t x) {
40:      if (isFull()) {
41:          throw std::runtime_error("can't enqueue to a full ring");
42:      } else if (_size == 0) {
43:          ringBuffer[head] = x;
44:          _size++;
45:      } else {
46:          if (tail == 0) {
47:              tail = Capacity-1;
48:              ringBuffer[tail] = x;
49:              _size++;
50:          } else {
51:              ringBuffer[tail-1] = x;
52:              tail--;
53:              _size++;
54:          }
55:      }
56:  }
57:  // delete and return item from the front
58:  int16_t RingBuffer::dequeue() {
59:      int16_t temp = ringBuffer[head];
60:      if (isEmpty()) {
61:          throw std::runtime_error("can't dequeue from an empty ring");
62:      } else {
63:          if (head-1 < 0 && _size > 1) {
```

```
64:             ringBuffer[head] = 0;
65:             head = Capacity-1;
66:         } else {
67:             ringBuffer[head] = 0;
68:             if (_size > 1)
69:                 head--;
70:         }
71:     }
72:     _size--;
73:     return temp;
74: }
75: // return (but do not delete) item from the front
76: int16_t RingBuffer::peek() {
77:     if (isEmpty()) {
78:         throw std::runtime_error("The ring is empty");
79:     }
80:     return ringBuffer[head];
81: }
```

RingBuffer.hpp

```
 1: #ifndef _HOME_DAPHNE_PS5A_RINGBUFFER_H_
 2: #define _HOME_DAPHNE_PS5A_RINGBUFFER_H_
 3: // Copyright 2018 Dangnhi Ngo
 4: #include <stdint.h>
 5: class RingBuffer {
 6: public:
 7:   explicit RingBuffer(int capacity);
 8:   int size();
 9:   bool isEmpty();
10:   bool isFull();
11:   void enqueue(int16_t x);
12:   int16_t dequeue();
13:   int16_t peek();
14:   int16_t *ringBuffer;
15: private:
16:   int Capacity;
17:   int _size;
18:   int head;
19:   int tail;
20: };
21:
22: #endif  // _HOME_DAPHNE_PS5A_RINGBUFFER_H_
```

PS5: RING BUFFER AND GUITAR HERO

PS5b: GUITAR HERO

Assignment Summary:

This assignment is the part 2 of the Assignment – Ring buffer with cpplint, testing and exceptions. We write a program that simulates plucking a guitar string using the Karplus-Strong algorithm. When a guitar string is plucked, the string vibrates and creates sound.

The program supports a total of 37 notes on the chromatic scale from 110Hz to 880Hz. Use the following 37 keys to represent the keyboard, from lowest note to highest note:

char keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' "; (Line 47, GuitarHero.cpp)

This keyboard arrangement imitates a piano keyboard: The "white keys" are on the “qwerty” and “zxcv” rows and the "black keys" on the “12345” and “asdf” rows of the keyboard. We generate a stream of string samples for audio playback under keyboard control.

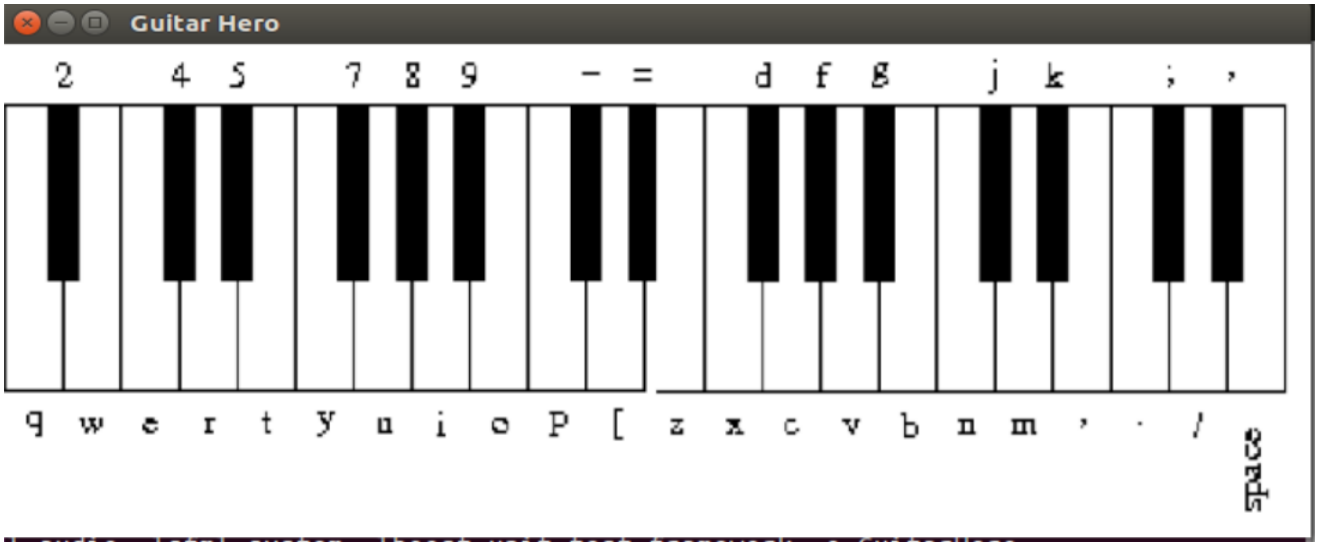
Algorithms/Data Structures:

Using the Karplus-Strong algorithm to stimulate the plucking of the guitar string. It simulates the vibration by maintaining a ring buffer of the N samples: the algorithm repeatedly deletes the first sample from the buffer and adds to the end of the buffer the average of the deleted sample and the first sample, scaled by an energy decay factor of 0.996.

Accomplishment:

Through this assignment, I have learnt how to create the buffer, enqueue and dequeue the ring. Besides, we have had to throw exception when it catches the error, such as: enqueue to a full ring or dequeue/peek from an empty ring.

Moreover, I have created the sound for the guitar string by filling the guitar string’s ring buffer with random numbers over the int16_t range in the pluck() method (Line 38-45, GuitarString.cpp)



Makefile

```
1:  CC = g++
2:  CFLAGS = -Wall -Werror -ansi -pedantic -std=c++0x -lboost_unit_test_framework
3:  LFLAGS = -lsfml-window -lsfml-graphics -lsfml-audio -lsfml-system -
lboost_unit_test_framework
4:
5:  .PHONY: all clean
6:
7:  all: GuitarHero gstest
8:
9:  GuitarHero: GuitarString.o GuitarHero.o RingBuffer.o
10:     $(CC) GuitarString.o GuitarHero.o RingBuffer.o $(LFLAGS) -o GuitarHero
11: gstest: GuitarString.o GStest.o RingBuffer.o
12:     $(CC) GuitarString.o GStest.o RingBuffer.o $(LFLAGS) -o gstest
13: GuitarHero.o: GuitarHero.cpp GuitarString.hpp RingBuffer.hpp
14:     $(CC) -c $< $(CFLAGS)
15: GuitarString.o: GuitarString.cpp GuitarString.hpp RingBuffer.hpp
16:     $(CC) -c $< $(CFLAGS)
17: RingBuffer.o: RingBuffer.cpp RingBuffer.hpp
18:     $(CC) -c $< $(CFLAGS)
19:
20: clean:
21:     rm *.o  GuitarHero gstest
```

GuitarHero.cpp

```
1:  // Copyright 2018 Dangnhi Ngo
2:  #include <SFML/Graphics.hpp>
3:  #include <SFML/System.hpp>
4:  #include <SFML/Audio.hpp>
5:  #include <SFML/Window.hpp>
6:  #include <vector>
7:  #include "GuitarString.hpp"
8:  #define CONCERT_A 440.0
9:  #define SAMPLES_PER_SEC 44100
10: std::vector<sf::Int16> makeSamplesFromString(GuitarString* gs) {
11:     std::vector<sf::Int16> samples;
12:     gs->pluck();
13:     int duration = 8;  // seconds
14:     int i;
15:     for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
16:         gs->tic();
17:         samples.push_back(gs->sample());
18:     }
19:     return samples;
20: }
21: int main() {
22:     sf::Image image;
23:     std::vector<std::vector<sf::Int16> > samples;
24:     std::vector<sf::SoundBuffer> buf1;
25:     std::vector<sf::Sound> sound1;
26:     double freq;
27:     if (!image.loadFromFile("keyboard.png")) {
28:         return -1;
29:     }
30:     sf::Vector2u size = image.getSize();
31:     sf::RenderWindow window(sf::VideoMode(size.x, size.y), "Guitar Hero");
32:     sf::Texture texture;
33:     texture.loadFromImage(image);
34:     sf::Sprite sprite;
35:     sprite.setTexture(texture);
36:     samples.reserve(37);
37:     buf1.resize(37);
38:     sound1.resize(37);
39:     for (double i = 0.0; i < 37; i++) {
40:         freq = CONCERT_A * pow(2, (i-24)/12.0);
41:         GuitarString gs(freq);
```

```
42:         samples.push_back(makeSamplesFromString(&gs));
43:         if (!(buf1[i]).loadFromSamples(&samples[i][0], samples[i].size(), 2,
SAMPLES_PER_SEC))
44:             throw std::runtime_error("sf::SoundBuffer: failed to load from samples.");
45:         sound1[i].setBuffer(buf1[i]);
46:   }
47:     char keyboard[] = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
48:     while (window.isOpen()) {
49:         sf::Event event;
50:         while (window.pollEvent(event)) {
51:             if (event.type == sf::Event::Closed)
52:                 window.close();
53:             for (int i = 0; i < 37; i++) {
54:                 if (event.type == sf::Event::TextEntered) {
55:                     if (event.text.unicode  == (unsigned)keyboard[i]) {
56:                         sound1[i].play();
57:                     }
58:                 }
59:             }  // end for loop
60:             window.clear();
61:             window.draw(sprite);
62:             window.display();
63:         }  // end while
64:     }
65:     return 0;
66:   }  // end main
```

GuitarString.cpp

```
 1:  // Copyright 2018 Dangnhi Ngo
 2:  #include <stdint.h>
 3:  #include <math.h>
 4:  #include <vector>
 5:  #include <exception>
 6:  #include <stdexcept>
 7:  #include "GuitarString.hpp"
 8:  #include "RingBuffer.hpp"
 9:  // create a guitar string of the given frequency
10:  // using a sampling rate of 44,100
11:  GuitarString::GuitarString(double frequency) {
12:     if (frequency > 0) {
13:         int capacity = 44100.0/frequency + .05;
14:         RB = new RingBuffer(capacity);
```

```
15:         for (int i = 0; i < capacity; i++) {
16:             (*RB).enqueue(0);
17:         }
18:         _time = 0;
19:     } else {
20:         throw std::invalid_argument("capacity must be greater than zero");
21:     }
22: }
23: // create a guitar string with
24: // size and initial values are given by the vector
25: GuitarString::GuitarString(std::vector<sf::Int16> init) {
26:     if (init.size() > 0) {
27:         RB = new RingBuffer(init.size());
28:         for (unsigned i = 0; i < init.size() ; i++) {
29:             (*RB).enqueue(init.at(i));
30:         }
31:         _time = 0;
32:     } else {
33:         throw std::invalid_argument("capacity must be greater than zero");
34:     }
35: }
36: // pluck the guitar string by replacing the buffer
37: // with random values, representing white noise
38: void GuitarString::pluck() {
39:     while (!(*RB).isEmpty()) {
40:         (*RB).dequeue();
41:     }
42:     while (!(*RB).isFull()) {
43:         (*RB).enqueue((sf::Int16)(rand() & 0xffff));
44:     }
45: }
46: // advance the simulation one time step
47: void GuitarString::tic() {
48:     int16_t temp = (((*RB).dequeue() + (*RB).peek())/2) * .996;
49:     (*RB).enqueue(temp);
50:     _time++;
51: }
52: // return the current sample
53: sf::Int16 GuitarString::sample() {
54:     return (*RB).peek();
55: }
56: // return number of times tic was called so far
```

```
57:  int GuitarString::time() {
58:      return _time;
59:      }
```

GuitarString.hpp

```
 1:  // Copyright 2018 Dangnhi Ngo
 2:  #ifndef _HOME_DAPHNE_PS5B_GUITARSTRING_H_
 3:  #define _HOME_DAPHNE_PS5B_GUITARSTRING_H_
 4:  #include <stdint.h>
 5:  #include <SFML/Audio.hpp>
 6:  #include <vector>
 7:  #include "RingBuffer.hpp"
 8:  class GuitarString {
 9:  public:
10:      explicit GuitarString(double frequency);
11:      explicit GuitarString(std::vector<sf::Int16> init);
12:      void pluck();
13:      void tic();
14:      sf::Int16 sample();
15:      int time();
16:  private:
17:      int time;
18:      RingBuffer* RB;
19:  };
20:
21: #endif  // _HOME_DAPHNE_PS5B_GUITARSTRING_H_
```

RingBuffer.cpp

```
 1:  // Copyright 2018 Dangnhi Ngo
 2:  #include "RingBuffer.hpp"
 3:  RingBuffer::RingBuffer(int Capacity) : capacity(Capacity) {
 4:      if (capacity <= 0) {
 5:          throw std::invalid_argument("capacity must be greater than zero.");
 6:      } else {
 7:          ringBuffer.reserve(capacity);
 8:          head = capacity-1;
 9:          tail = capacity-1;
10:          size = 0;
11:      }
12:  }
13:  int RingBuffer::Size() {
14:      return size;
```

```
15:  }
16:  bool RingBuffer::isEmpty() {
17:      if (Size() == 0) {
18:          return true;
19:      } else {
20:          return false;
21:      }
22:  }
23:  bool RingBuffer::isFull() {
24:      if (Size() == capacity) {
25:          return true;
26:      } else {
27:          return false;
28:      }
29:  }
30:  void RingBuffer::enqueue(int16_t x) {
31:      if (!isFull()) {
32:          if (Size() == 0) {
33:              ringBuffer[head] = x;
34:              size++;
35:          } else {
36:              if (tail != 0) {
37:                  ringBuffer[tail-1] = x;
38:                  tail--;
39:                  size++;
40:              } else {
41:                  tail = capacity-1;
42:                  ringBuffer[tail] = x;
43:                  size++;
44:              }
45:          }
46:      } else {
47:          throw std::runtime_error("can't enqueue to a full ring.");
48:      }
49:  }
50:  int16_t RingBuffer::dequeue() {
51:      int16_t temp = ringBuffer[head];
52:      if (!isEmpty()) {
53:          if (head-1 < 0 && Size() > 1) {
54:              ringBuffer[head] = 0;
55:              head = capacity-1;
56:          } else {
```

```
57:             ringBuffer[head] = 0;
58:             if (Size() > 1) {
59:                 head--;
60:             }
61:             }
62:         size--;
63:         return temp;
64:     } else {
65:         throw std::runtime_error("can’t dequeue from an empty ring.");
66:     }
67: }
68: int16_t RingBuffer::peek() {
69:     if (!isEmpty()) {
70:         return ringBuffer[head];
71:     } else {
72:         throw std::runtime_error("the ring is empty.");
73:     }
74: }
```

RingBuffer.hpp

```
 1: //  Copyright 2018 Dangnhi Ngo
 2: #ifndef _HOME_DAPHNE_PS5B_RINGBUFFER_H_
 3: #define _HOME_DAPHNE_PS5B_RINGBUFFER_H_
 4: #include <stdint.h>
 5: #include <iostream>
 6: #include <stdexcept>
 7: #include <vector>
 8: class RingBuffer {
 9: public:
10:     explicit RingBuffer(int Capacity);
11:     int Size();
12:     bool isEmpty();
13:     bool isFull();
14:     void enqueue(int16_t x);
15:     int16_t dequeue();
16:     int16_t peek();
17: private:
18:     std::vector<int16_t> ringBuffer;
19:     int capacity;
20:     int size;
21:     int head;
22:     int tail;
```

```
23:   };
24:
25: #endif  // _HOME_DAPHNE_PS5B_RINGBUFFER_H_
```

PS6: MARKOV MODEL OF NATURAL LANGUAGE

Assignment Summary:

The goal of this assignment is to analyze an input text file for transitions between k-grams (a fixed number of characters) and the following letter. Then, we export the k-gram, there frequencies, frequency of the next characters, and calculate the probability of the next characters that are correlational of each k-gram.

Algorithms/Data Structures:

The algorithm of this assignment is to apply the Markov model of natural languages. It creates an output table comprised of the k-grams, the number of times it appears and the frequency of the following character (Line 54-64, MarkovModel.cpp). It creates the trajectory by appending the random character selected at each step (Line 112-128, MarkovModel.cpp).

For example, the image below shows the text is the 17-character string "gagggagaggcgagaaa" and k = 2

		frequency of next char			probability that next char is		
kgram	freq	a	c	g	a	c	g
aa	2	1	0	1	1/2	0	1/2
ag	5	3	0	2	3/5	0	2/5
cg	1	1	0	0	1	0	0
ga	5	1	0	4	1/5	0	4/5
gc	1	0	0	1	0	0	1
gg	3	1	1	1	1/3	1/3	1/3
	17	7	1	9			

Accomplishment:

Through this assignment, I have known how to create a Markov model of order k from the given text as well as generate a string of length T characters by simulating a trajectory through the corresponding Markov chain.

```
daphne@daphne-VirtualBox:~/Downloads/ps6$ ./mmtest
Running 3 test cases...

*** No errors detected
```

```
daphne@daphne-VirtualBox:~/Downloads/ps6$ ./TextGenerator 2 5 < input17.txt
gaggg
kgram   freq     prob
----------------------

aa       2       2/17
aaa      1
aac      0
aag      1

ag       5       5/17
aga      3
agc      0
agg      2

cg       1       1/17
cga      1
cgc      0
cgg      0

ga       5       5/17
gaa      1
gac      0
gag      4

gc       1       1/17
gca      0
gcc      0
gcg      1

gg       3       3/17
gga      1
ggc      1
ggg      1
```

Makefile

```
1:  CC = g++
2:  CFLAGS = -Wall -ansi -pedantic -Werror -std=c++0x
3:  LFLAGS = -lsfml-window -lsfml-graphics -lsfml-audio -lsfml-system -
    lboost_unit_test_framework
4:
5:  .PHONY: all clean
6:
7:  all: TextGenerator mmtest
8:
9:  TextGenerator: MarkovModel.o TextGenerator.o
10:     $(CC) MarkovModel.o TextGenerator.o $(LFLAGS) -o TextGenerator
11: mmtest: MarkovModel.o mmtest.o
12:     $(CC) MarkovModel.o mmtest.o $(LFLAGS) -o mmtest
13: MarkovModel.o: MarkovModel.cpp MarkovModel.hpp
14:     $(CC) -c $< $(CFLAGS)
15: TextGenerator.o: TextGenerator.cpp
16:     $(CC) -c $< $(CFLAGS)
17: mmtest.o: mmtest.cpp
18:     $(CC) -c $< $(CFLAGS)
19:
```

```
20: clean:
21:     rm *.o TextGenerator  mmtest
```

TextGenerator.cpp

```
 1: // Copyright 2018, Dangnhi Ngo
 2: #include <string>
 3: #include <iostream>
 4: #include <exception>
 5: #include <stdexcept>
 6: #include <cstdlib>
 7: #include "MarkovModel.hpp"
 8: int main(int argc, char* argv[]) {
 9:     int k = atoi(argv[1]);
10:     int T = atoi(argv[2]);
11:     std::string kgram;
12:     std::cin >> kgram;
13:     MarkovModel mm(kgram, k);
14:     kgram = kgram.substr(0, k);
15:     std::cout << mm.gen(kgram, T) << std::endl;
16:     std::cout << mm << std::endl;
17: return 0;
18: }
```

MarkovModel.cpp

```
 1: // Copyright 2018, Dangnhi Ngo
 2: #include <vector>
 3: #include <exception>
 4: #include <stdexcept>
 5: #include <iostream>
 6: #include <map>
 7: #include <string>
 8: #include "MarkovModel.hpp"
 9: using std::string;
10: // create a Markov model of order k from given text
11: // Assume that text has length at least k.
12: MarkovModel::MarkovModel(std::string text, int k)  {
13:     strOfChar = text;
14:     _order = k;
15:     if ((unsigned) _order >= text.length()) {
16:         throw std::runtime_error("k is larger than the length of text");
17:     }
```

```
18:     for (unsigned i = 0; i < text.size(); i++) {
19:         if (ch.find(text.at(i)) == std::string::npos) {
20:             ch.push_back(text.at(i));
21:         }
22:     }
23:     for (unsigned i = 0; i < text.size(); i++) {
24:         std::string cha1, cha2;
25:         for (unsigned j = i; j < i + k; j++)
26:             if (j < text.size()) {
27:                 cha1.push_back(text[j]);
28:             } else {
29:                 cha1.push_back(text[j - text.size()]);
30:             }
31:         if (kk.end() != kk.find(cha1)) {
32:             kk[cha1] += 1;
33:         } else {
34:             kk[cha1] = 1;
35:         }
36:         for (unsigned j = 0; j < strOfChar.size(); j++)
37:             if (kk.end() == kk.find(cha1 + strOfChar[j])) {
38:                 kk[cha1 + strOfChar[j]] = 0;
39:             }
40:         for (unsigned j = i; j < i + k + 1; j++)
41:             if (j >= text.size())
42:                 cha2.push_back(text[j - text.size()]);
43:             else
44:                 cha2.push_back(text[j]);
45:         kk[cha2] += 1;
46:     }
47: }
48: // returns order k of Markov model
49: int MarkovModel::order() {
50:     return _order;
51: }
52: // returns number of occurrences of kgram in text
53: // (throw an exception if kgram is not of length k)
54: int MarkovModel::freq(std::string kgram) {
55:     if (kgram.size() == (unsigned)_order) {
56:         if (_order != 0) {
57:             return kk[kgram];
58:         } else {
59:             return strOfChar.size();
```

```
60:         }
61:     } else {
62:         throw std::runtime_error("kgram is not of length k");
63:     }
64: }
65: // returns number of times that character c follows kgram
66: // if order=0, return num of times char c appears
67: // (throw an exception if kgram is not of length k)
68: int MarkovModel::freq(std::string kgram, char c) {
69:     if (kgram.size() == (unsigned)_order) {
70:         if (_order == 0) {
71:             int num = 0;
72:             for (unsigned i = 0; i < strOfChar.size(); i++) {
73:                 if (strOfChar[i] == c) {
74:                     num++;
75:                 }
76:             }
77:             return num;
78:         } else {
79:             return kk[kgram + c];
80:         }
81:         return 0;
82:     } else {
83:         throw std::runtime_error("kgram is not of length k");
84:     }
85: }
86: // random character following given kgram
87: // (Throw an exception if kgram is not of length k.
88: //  Throw an exception if no such kgram.)
89: char MarkovModel::randk(std::string kgram) {
90:     if (kgram.length() == (unsigned)_order) {
91:         if (freq(kgram) != 0) {
92:             std::string temp;
93:             for (unsigned i = 0 ; i <strOfChar.size() ; i++) {
94:                 for (int j = 0; j < kk[kgram + strOfChar[i]]; j++) {
95:                     temp.push_back(strOfChar[i]);
96:                 }
97:             }
98:             return temp[rand() % temp.size()];
99:         } else {
100:             throw std::runtime_error("can not find kgram");
101:         }
```

```
102:     } else {
103:         throw std::runtime_error("kgram is not of length k");
104:     }
105: }
106: // generate a string of length T characters
107: // by simulating a trajectory through the corresponding
108: // Markov chain.  The first k characters of the newly
109: // generated string should be the argument kgram.
110: // Throw an exception if kgram is not of length k.
111: // Assume that T is at least k.
112: string MarkovModel::gen(std::string kgram, int T) {
113:     std::string output = kgram, str;
114:     if (kgram.length() == (unsigned)order()) {
115:         while ((unsigned) T > output.length()) {
116:             string temp(1, randk(kgram));
117:             output.append(temp);
118:             if (_order > 0) {
119:                 kgram = kgram.substr(1);
120:             }
121:             str = kgram + temp;
122:             kgram = str;
123:         }
124:         return output;
125:     } else {
126:         throw std::runtime_error("kgram does not match order size");
127:     }
128: }
129: // overload the stream insertion operator and display
130: // the internal state of the Markov Model. Print out
131: // the order, the alphabet, and the frequencies of
132: // the k-grams and k+1-grams.
133: std::ostream& operator << (std::ostream& out, MarkovModel& mm) {
134:     std::map<std::string, int>::iterator kk;
135:     out << "kgram\tfreq\t prob" << std::endl;
136:     out << "----------------------" << std::endl;
137:     for (kk = mm.kk.begin(); kk != mm.kk.end(); ++kk) {
138:         if ((*kk).first.length() != (unsigned) mm._order) {
139:             out <<(*kk).first << "\t " <<(*kk).second;
140:             out << std::endl;
141:         } else {
142:             out << std::endl;
143:             out << (*kk).first <<"\t "<< (*kk).second;
```

```
144:             out << "\t ";
145:             out << (*kk).second << "/" << mm.strOfChar.size();
146:             out << std::endl;
147:         }
148:     }
149:     return out;
150: }
```

MarkovModel.hpp

```
 1: // Copyright 2018, Dangnhi Ngo
 2: #ifndef _HOME_DAPHNE_PS6_MARKOVMODEL_H_
 3: #define _HOME_DAPHNE_PS6_MARKOVMODEL_H_
 4: #include <map>
 5: #include <string>
 6: #include <cstdlib>
 7: class MarkovModel {
 8: public:
 9:     MarkovModel(std::string text, int k);
10:     int order();
11:     int freq(std::string kgram);
12:     int freq(std::string kgram, char c);
13:     char randk(std::string kgram);
14:     std::string gen(std::string kgram, int T);
15:     friend std::ostream& operator <<(std::ostream& out, MarkovModel& mm);
16: private:
17:     int _order;
18:     std::string strOfChar;
19:     std::string ch;
20:     std::map <std::string, int> kk;
21: };
22:
23: #endif  // _HOME_DAPHNE_PS6_MARKOVMODE
```

PS7: KRONOS INTOUCHING PARSING

PS7a: KRONOS TIME CLOCK: INTRODUCTION TO REGULAR EXPRESSION PARSING

Assignment Summary:

The goal of this assignment is to analyze the Kronos InTouch time clock log by using regular expressions to parse the file. In this assignment, we need to verify the device boot up timing by reading in an entire InTouch log and reporting each startup, whether it completed or failed, and the time required to complete. If it finds the matching completion message, it will print out the successful boot up; otherwise, the result will be incomplete boot up if no matching.

Algorithms/Data Structures:

The program reads the input text file and exports the output file with the same name but with a suffix ".rpt"

Accomplishment:

Throught this assignment, I have learnt how to scan the input text file, find the matching messages to print out the appropriate results.

I have created two regrexes regex_st and regex_ed to check the begin and finish string (Line 10-16, main.cpp). Then, I have used the regrex_match to check if the new line matched with the regex begin and end. Also, I have calculated the time starts and ends to compute the time process. If the boot is not completed, it will print out the result **** Incomplete boot **** (Line 43, main.cpp)

```
=== Device boot ===
435369(device1_intouch.log): 2014-03-25 19:11:59 Boot Start
435759(device1_intouch.log): 2014-03-25 19:15:02 Boot Completed
     Boot Time: 183000ms

=== Device boot ===
436500(device1_intouch.log): 2014-03-25 19:29:59 Boot Start
436859(device1_intouch.log): 2014-03-25 19:32:44 Boot Completed
     Boot Time: 165000ms

=== Device boot ===
440719(device1_intouch.log): 2014-03-25 22:01:46 Boot Start
440791(device1_intouch.log): 2014-03-25 22:04:27 Boot Completed
     Boot Time: 161000ms

=== Device boot ===
440866(device1_intouch.log): 2014-03-26 12:47:42 Boot Start
441216(device1_intouch.log): 2014-03-26 12:50:29 Boot Completed
     Boot Time: 167000ms

=== Device boot ===
442094(device1_intouch.log): 2014-03-26 20:41:34 Boot Start
442432(device1_intouch.log): 2014-03-26 20:44:13 Boot Completed
     Boot Time: 159000ms

=== Device boot ===
443073(device1_intouch.log): 2014-03-27 14:09:01 Boot Start
443411(device1_intouch.log): 2014-03-27 14:11:42 Boot Completed
     Boot Time: 161000ms
```

Makefile

```
1:  CC = g++
2:  CFLAGS = -Wall -ansi -pedantic -Werror -ansi -std=c++0x -lboost_unit_test_framework
3:  LFLAGS = -lsfml-system -lsfml-audio -lsfml-graphics -lsfml-window -lboost_regex -
    lboost_date_time -lboost_unit_test_framework
4:
5:  .PHONY: all clean
6:
7:  all: ps7a
8:
9:  ps7a: main.o
10:     $(CC) main.o $(LFLAGS) -o ps7a
11: main.o: main.cpp
12:     $(CC) -c $< $(CFLAGS)
13:
14: clean:
15:     rm ps7a main.o
```

main.cpp

```
1:  // Copyright 2018 Dangnhi Ngo
2:  #include <boost/regex.hpp>
3:  #include <boost/date_time/gregorian/gregorian.hpp>
4:  #include <boost/date_time/posix_time/posix_time.hpp>
5:  #include <exception>
6:  #include <iostream>
7:  #include <fstream>
8:  #include <string>
9:  int main(int argc, char* argv[]) {
10:     std::string regex_st =
11:         "([0-9]+)-([0-9]+)-([0-9]+) "
12:         "([0-9]+):([0-9]+):([0-9]+): "
13:         ".*\\(log.c.166\\) server started.*";
14:     std::string regex_ed =
15:         "([0-9]+)-([0-9]+)-([0-9]+) "
16:         "([0-9]+):([0-9]+):([0-9]+).([0-9]+):INFO:"
17:         "oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080.*";
18:     boost::regex begin_boot(regex_st);
19:     boost::regex end_boot(regex_ed);
20:     std::ifstream logfile(argv[1]);
21:     std::string newfile = argv[1];
22:     std::ofstream outputfile(newfile + ".rpt", std::ofstream::out);
23:     std::string pos;
```

```
24:     boost::posix_time::ptime beginTime;
25:     using boost::posix_time::time_duration;
26:     bool isCompleted = true;
27:     int _code = 1;
28:     if (argc != 2) {
29:        throw std::runtime_error("Please input the log file");
30:     } else {
31:        if (logfile.is_open()) {
32:        } else {
33:              throw std::runtime_error("Cannot open file");
34:        }
35:        while (logfile.eof() == false) {
36:           std::getline(logfile, pos);
37:           boost::smatch startT, endT;
38:           if (regex_match(pos, startT, begin_boot)) {
39:              boost::gregorian::date dateT(stoi(startT[1]), stoi(startT[2]), stoi(startT[3]));
40:              boost::posix_time::ptime _startT(dateT, time_duration(stoi(startT[4]),
stoi(startT[5]), stoi(startT[6]))); // NOLINT
41:              beginTime = _startT;
42:              if (!isCompleted) {
43:                 outputfile << "**** Incomplete boot ****\n" << std::endl;
44:              }
45:              outputfile << "=== Device boot ===\n"
46:                         << _code << "(" << argv[1] << "): "
47:                         << startT[1] << "-" << startT[2] << "-"
48:                         << startT[3] << " " << startT[4] << ":"
49:                         << startT[5] << ":" << startT[6]
50:                         << " Boot Start" << std::endl;
51:              isCompleted = false;
52:           } else if (regex_match(pos, endT, end_boot)) {
53:              boost::gregorian::date dateT(stoi(endT[1]), stoi(endT[2]), stoi(endT[3]));
54:              boost::posix_time::ptime finishTime(dateT, time_duration
(stoi(endT[4 ]),stoi(endT[5]), stoi(endT[6]))); // NOLINT
55:              outputfile << _code << "(" << argv[1] << "): "
56:                         << endT[1] << "-" << endT[2] << "-"
57:                         << endT[3] << " " << endT[4] << ":"
58:                         << endT[5] << ":" << endT[6]
59:                         << " Boot Completed" << std::endl;
60:              time_duration time_d = finishTime - beginTime;
61:              outputfile << "       "
62:                         << " Boot Time: " << time_d.total_milliseconds() << "ms\n
63:              isCompleted = true;
```

```
64:             }
65:             _code++;
66:         }
67:         outputfile.close();
68:         logfile.close();
69:     }
70:     return 0;
71: }
```

PS7: KRONOS INTOUCHING PARSING

PS7b: KRONOS TIME CLOCK: FULL CHALLENGE

I have not completed successfully this assignment for extra credits.