SHREYA PATIL

COMP IV: PROJECT-PORTFOLIO
SPRING 2020
05/01/2020

CONTENTS:

PS0 Hello World with SFML

The main task in this assignment is to build a sprite image by getting familiar with SFML and its different libraries.

In this assignment I have used an image of squirrel as sprite image. When the arrow keystrokes are used the image responds by moving in that direction.

The program created, uses SFML's sprite class for displaying a moving image on SFML window. Also, the sprite image rotates in left direction for L keyword and in right direction for R keyword.

In this assignment I've learned to play around with SFML and its libraries such as, Images, Sprites, Window, Text, Texture, keyboard events and other events. I was already familiar with setting up and programming in Linux environment.

Output:

Main.cpp:

```
1.  #include <SFML/Graphics.hpp>
2.  #include <SFML/Audio.hpp>
3.  #include <SFML/Window.hpp>
    int main()
4.
5.  {
6.  //creating the main window
7.  sf::RenderWindow window(sf::VideoMode(800,600),"SFML WINDOW!");
    //Loading a sprite
8.
9.  sf::Texture texture;
10. if(!texture.loadFromFile("sprite.jpeg"))
11. return EXIT_FAILURE;
    sf::Sprite sprite(texture);
12.
13. //setting the positions for the sprite
14. sprite.setPosition(200,100);


15. sf::Font font;
16. sf::Text text("Hello SFML",font,30);
17. //initiating the loop
18.  while(window.isOpen())
19.  {//initializing a variable for events occuring in the SFML window

20.  sf::Event event;
21.  while(window.pollEvent(event))
22.  {//if pressed left arrow key
23.  if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
24.  sprite.move(-10,0);
25.  //if pressed right arrow key
26.  if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
27.  sprite.move(10,0);
28.  //if pressed up arrow key
29.  if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
30.  sprite.move(0,-10);
31.  //if pressed down arrow key
```

```
32.         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
33.         sprite.move(0,10);
34.         //rotates in Right direction when pressed R keyword
35.         if(sf::Keyboard::isKeyPressed(sf::Keyboard::R))
36.         sprite.rotate(1);
37.         //rotates in Left direction when pressed L keyword
38.         if(sf::Keyboard::isKeyPressed(sf::Keyboard::L))
39.         sprite.rotate(-1);
40.         //Exiting the window
41.         if(event.type == sf::Event::Closed)
42.         window.close();
43.         }
44.         window.clear();
45.         window.draw(sprite);
46.         window.display();
47.         }
48.         return EXIT_SUCCESS;
49.         }
```

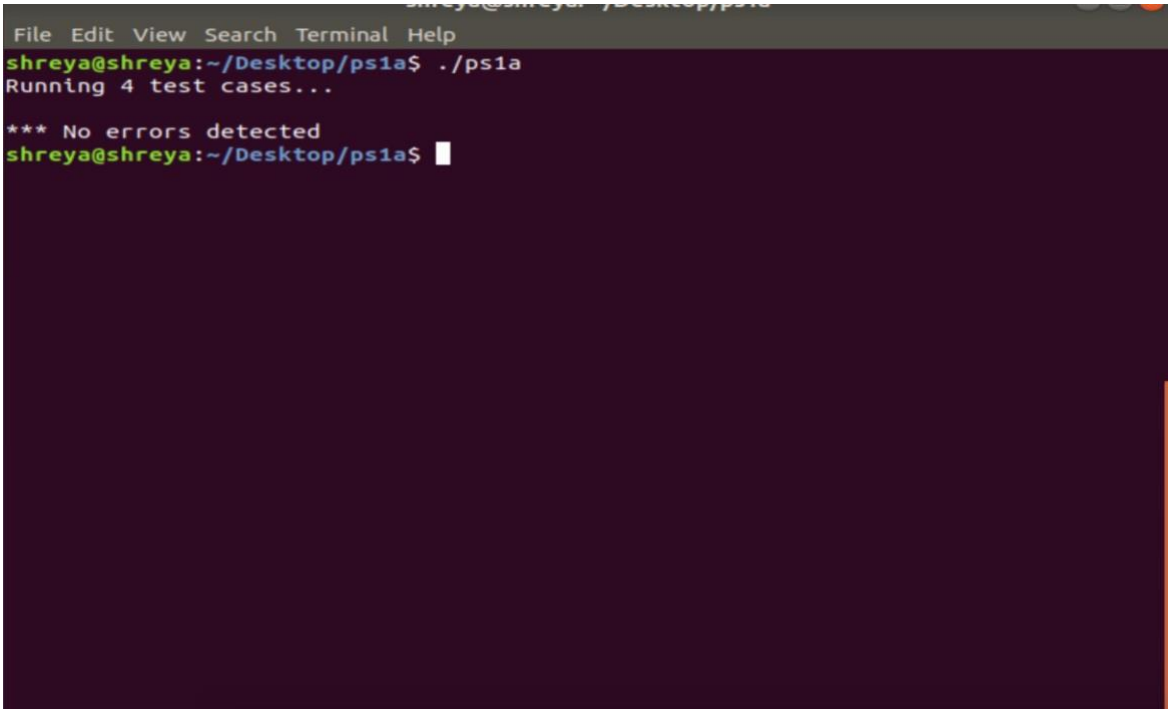
PS1 (part A) LINEAR FEEDBACK SHIFT REGISTER

The main purpose of this assignment was to build a Linear Feedback Shift
Register which produces pseudo-random bits. We also had to implement unit tests using Boost test framework.

The Fibonacci LFSR created depends on the input of the previous output. The output for the given LFSR was obtained by tapping the fixed positions. 13,12,10 positions were XORed with left most bit. The generate function produced the decimal value for the sequence of given number of output bits of step functions.

The basic idea for the algorithm used in this assignment is to create a register to store input seed value and to left shift all the bits by using step and generate functions. Thus a constructor is created to hold the initial seeds and the tap values. The step function generates result after simulating a single step. And the generate function returns a k-bit integer after simulating k-steps. The input operator is overloaded to generate current display value in printable form.
The test framework was used to test FibLFSR class, step and generate function including other typical cases.

The project helped me about the unit testing framework. It helped me understand whether my code is working or broken. Also, overloading the input character for displaying was quite interesting.



PS1 (part B) LINEAR FEEDBACK SHIFT REGISTER

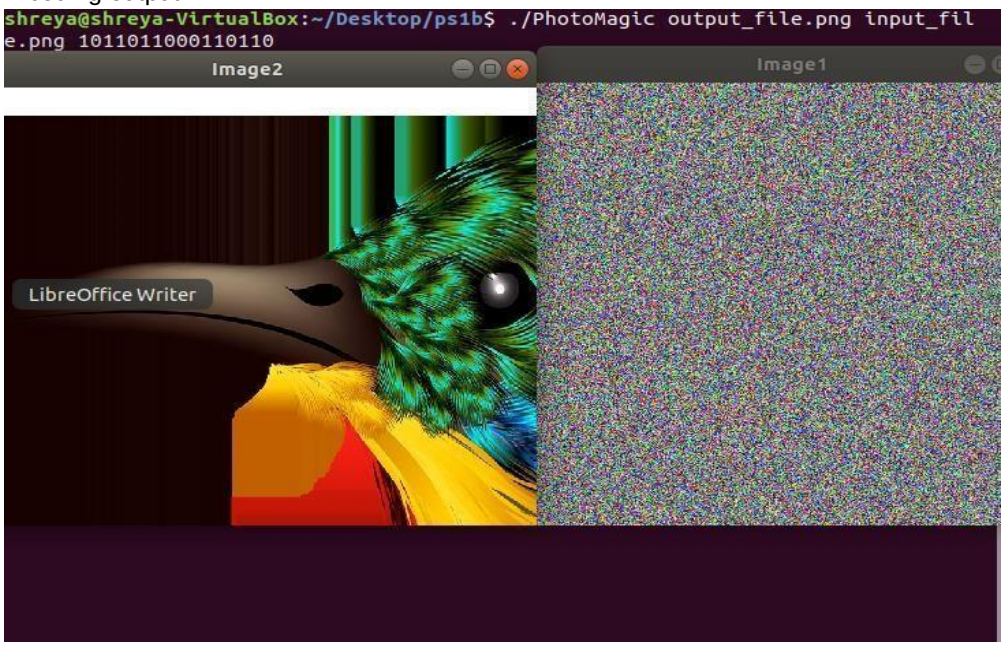
The main purpose of this assignment was to build a program that encodes/decodes a given image and saves it to the disk.

The created program takes the source image filename, output image filename and seed from the command line, transforms the image into pixels and then each color of the pixel to a new one. Thus giving a whole new encrypted version of the image and if the encrypted image is given as input it gives back the original image. I have successfully completed 100% of the assignment.

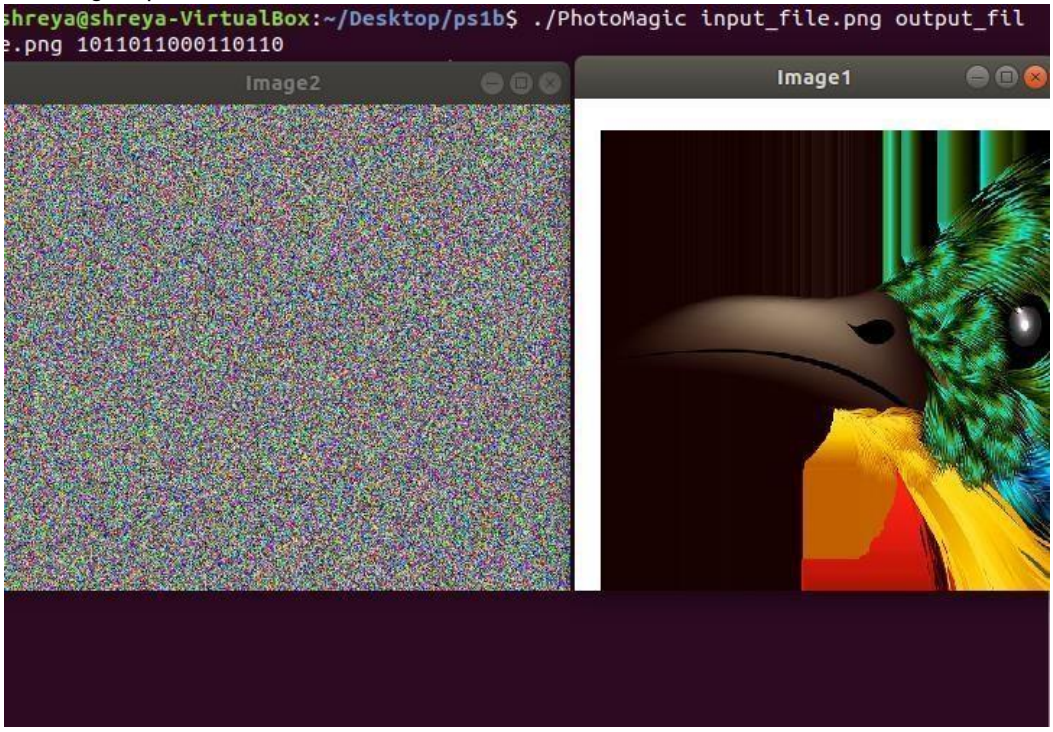
The previous assignment is used for creating a Fibonacci LFSR and left shifting all the bits. Then the transform function takes LFSR and an image as arguments and transforms the image using LFSR. To encode pixels in the image, getPixel() method is used. Various SFML objects were also used such as window to display the output, textures, images and srpites.

In this assignment I've learned methods of encoding, extracting pixels from an image and playing around with them. I've also learned how FibLFSR assignment can be reused for building an encrypted image.

Encoding output:



Decoding output:



Makefile:

```
1.CC=g++
2.CFLAGS= - Wall - Werror  -ansi - pedantic
   3.SFML=  -lsfml    - graphics - lsfml     -  window -  lsfml-  system
4.

5.  all:     PhotoMagic
6.
7.     PhotoMagic:  PhotoMagic.o  FibLFSR.o
8.     $(CC)  PhotoMagic.o  FibLFSR.o  - o  PhotoMagic  $(SFML)
9.
10.          PhotoMagic.o:    PhotoMagic.cpp  FibLFSR.hpp
11.          $(CC)  - c  PhotoMagic.cpp  FibLFSR.hpp  $(CFLAGS)
12.
13.          FibLFSR.o:       FibLFSR.cpp  FibLFSR.hpp
14.          $(CC)  - c  FibLFSR.cpp  $(CFLAGS)
15.
16.          clean:
17.          rm  *.o
18.          rm  PhotoMagic
```

PhotoMagic.cpp

```
#include "FibLFSR.hpp"
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

using namespace sf;

sf::Image transform( sf::Image input_file, FibLFSR fiblfsr ){
        Color p1;
        Color p2;
        sf::Vector2u size = input_file.getSize();    //returns the size( width and height) in pixels
        int s_x = size.x;  //x=width
        int s_y = size.y; //y=height

                Image output_file;
                output_file.create(s_x, s_y);   //create an output image from a n array of pixels
                for (int i = 0; i<s_x; i++)
                {
                for (int j = 0; j< s_y; j++) {
                //extracting the color of the pixel in row major order
                p1 = input_file.getPixel(i, j);  //get the color of pix el
                p2 = output_file.getPixel(i, j);
                p2.r = p1.r ^ fiblfsr.generate(8);
                p2.g = p1.g ^ fiblfsr.generate(8);
```

```
                    p2.b = p1.b ^ fiblfsr.generate(8);
                    output_file.setPixel(i, j, p2); //change the color of pixel based on output generated
                }
            }
        return output_file;


      }

        int main(int argc, char** argv)
        {
            if(argc < 2)
            {
                cout << "Not enough arguments" << endl;
                exit(1);
            }
            int  len;

            string img_src = argv[1];
            string enc_img = argv[2];
            string seed = argv[3];


            len = seed.size();
            for(int i = 0 ; i < len; i++)
            {
                seed[i] = (seed[i] % 2) + '0';
            }
        Image input_file;
        if(!input_file.loadFromFile(img_src))
            return -1;

        FibLFSR fiblfsr = FibLFSR(seed);
        Image result;
        result=transform( input_file, seed);
         sf::Vector2u size = result.getSize();
        RenderWindow window1(VideoMode(size.x, size.y), "Image1");
            RenderWindow window2(VideoMode(size.x, size.y), "Image2");

            sf::Texture texture;
            texture.loadFromImage(input_file);

            sf::Sprite sprite;
            sprite.setTexture(texture);

            sf::Texture texture1;
            texture1.loadFromImage(result);

            sf::Sprite sprite1;
            sprite1.setTexture(texture1);

            while (window1.isOpen() && window2.isOpen()) {

            sf::Event event;
            while (window1.pollEvent(event)) {
              if (event.type == sf::Event::Closed)
```

```
81.                 window1.close();
82.             }
83.             while (window2.pollEvent(event)) {
84.               if (event.type == sf::Event::Closed)
85.                 window2.close();
86.             }
87.             window1.clear();
88.             window1.draw(sprite);
89.             window1.display();
90.             window2.clear();
91.             window2.draw(sprite1);
92.             window2.display();
93.           }
94.         if (!result.saveToFile(enc_img))
95.             return -1;
96.
97.
98.             return 0;
99.         }
```

FibLFSR.hpp

```
#include <iostream>
#include <string>

using namespace std;

class FibLFSR {
public:
    FibLFSR(string seed);
    int step();
          int generate(int k);
          friend ostream &operator<<(ostream &out, FibLFSR lfsr);

      private:
      string seed1;

      };
```

FibLFSR.cpp

```
#ifndef SFML_HPP
#define SFML_HPP
#include "FibLFSR.hpp"     4.
                    //defining the constructor
                    FibLFSR::FibLFSR(string seed)
                    {
                    seed1 = seed;
                    }
                    //defining step  member function
                    int FibLFSR::step()
                    {/*to tap the seed at 13,12,10 positions, it can also be consider ed as 2,3,5
                    positions in a string*/
                    int tap = (seed1[0]) ^ (seed1[2]);
                    tap = tap ^ seed1[3];
                    tap = tap ^ seed1[5];
                    //shifting the  string to left by 1 bit
                    int len = seed1.size();
                    for(int i = 0; i < len  - 1; i++)

                    seed1[i] = seed1[i+1];
                     {
                }
                seed1[seed1.size() - 1] = tap + '0';
                return tap;
            }
            int FibLFSR::generate(int k)
            {//obtaining a decimal value for output of k-step functions
                int k_bit = 0;
                for(int i = 0; i < k; i++)
                {
                    int val = step();
                    k_bit = 2*k_bit + val;
                }
                return k_bit;
            }
            //overloading the stream insertion operator
            ostream &operator<<(ostream &out, FibLFSR lfsr)
            {
                out << lfsr.seed1;
                return out;
```

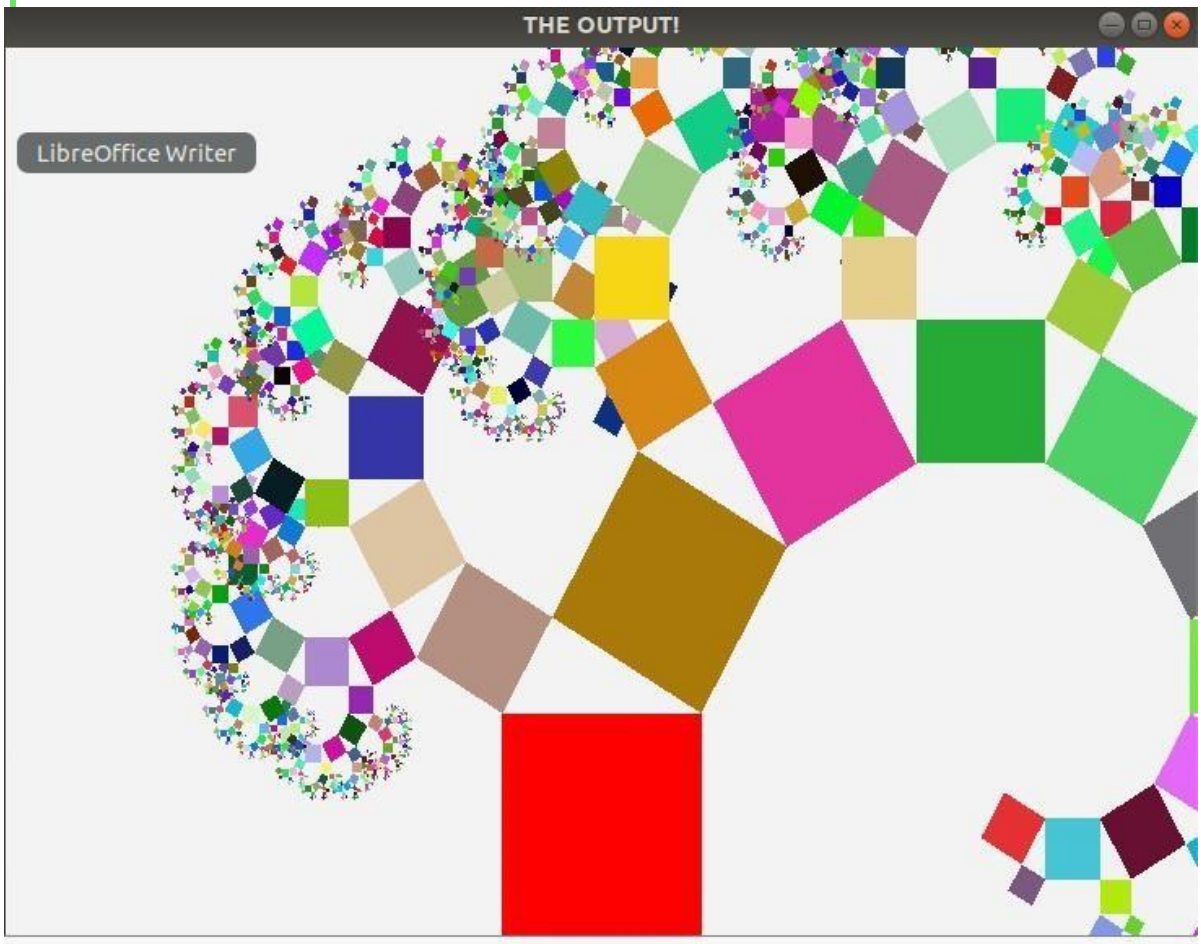
```
40.         }
41.         #endif
```

PS2 RECURSIVE GRAPHICS (PYTHAGORAS TREE)

The core task of this assignment was to build a Pythagoras tree using SFML libraries. The main technique was to create a square and recursively build multiple squares with certain given angles and position at every recursive point. It takes size of the base square and depth of recursion as arguments and an angle to build the tree. The depth is used to determine the number of times the pattern is to be repeated for forming the tree. I was able to complete 100% of the assignment along with extra credit to build a beautiful tree with custom angles and different colors.

The main OO design I used to implement this assignment is, I used the main_sq as the base square and then added left and right objects by using left and right pointers to keep everything connected. After setting the base square the angle and position is set for every recursive step.
The functionality and working of the tree was implemented with the help of
SFML libraries to transform, and recursively set the size, origin, rotation and position of each square. In SFML library, transform class provides all functionalities to calculate positioning and angle of the next square.
The main program calls the pTree function of the pTree class to pass the values for building the tree. pyth_tree() function then recursively calls pTree function for its left and right squares and again on these squares pyth_tree() function is called. This level of recursion is done until a certain number of times i.e., the depth of recursion provided by the user.
To color the tree with different colors, I could do this with the help of setFillColor() function.
To build the tree with customized angle provided by the user., the angle given is assigned as the angle of positioning to the left square and the right square is placed at an angle of 90-given angle. I was able to do this with the help of Transform function to calculate size, position and rotation of the co-ordinates.

This assignment helped me learn different possibilities to implement SFML libraries. It also helped get used to the idea of recursion in building fractal patterns to create a Pythagoras tree.


THE OUTPUT!
LibreOffice Writer

Makefile:

```
CC= g++
CFLAGS= -pedantic -std=c++11
LFLAGS= -lsfml-graphics -lsfml-window -lsfml-system

all: pTree

pTree: main.o pTree.o
    $(CC) main.o pTree.o -o pTree $(LFLAGS) $(CFLAGS)

main.o: main.cpp pTree.hpp
    g++ -c main.cpp -pedantic -std=c++11


pTree.o: pTree.cpp pTree.hpp
    $(CC) -c pTree.cpp $(CFLAGS)

clean:
    rm *.o
    rm pTree
```

main.cpp:

```cpp
#include "pTree.hpp"
#include <iostream>
#include <cmath>
#include <SFML/Graphics.hpp>
using namespace std;
using namespace sf;
int main(int argc, char** argv)
{
  if(argv[1] == NULL || argv[2] == NULL)
        {
          cout << "Sorry canot build the tree." << endl;
          cout << "BYE" << endl;
          exit(1);
        }
        int x = atoi(argv[2]);
        int y = atoi(argv[1]);
        int default_angle= 45;
        if(argv[3] != NULL) default_angle = atoi(argv[3]);
        pTree obj(y, x, default_angle);

        RenderWindow window(sf::VideoMode(6.0 * y, 4.0 * y), "THE OUTPUT!");
        window.setVerticalSyncEnabled(true);

        while (window.isOpen())
          {
            Event event;
            while (window.pollEvent(event))
              {
                  if (event.type == sf::Event::Closed)
                      window.close();
              }
            window.clear(Color(243,243,243));
            window.draw(obj);
            window.display();
          }

        return 0;
      }
```

PTree.hpp:

```
#ifndef PTREE_HPP

#define PTREE_HPP

#include <iostream>

#include <cmath>

#include <SFML/Graphics.hpp>

      using namespace sf;

      class pTree : public Drawable

      {

      public:

        pTree(double Len, int Num, double given_angle);

        pTree(double Len, int Num, double deg_X, Vector2f pos, double g
iven_angle);

        void pyth_tree();


      private:

        int cnt;

        double n;

        double deg;

        double angle;

        Vector2f position;

        Vector2f nextL;

        Vector2f nextR;

          RectangleShape main_sq;

          pTree* lside;

          pTree* rside;

          virtual void draw(RenderTarget& t, RenderStates states) const
```

```
50.
51.         {
52.
53.           if(cnt < 1)
54.
55.           {
56.
57.             t.draw(main_sq, states);
58.
59.             return;
60.
61.           }
62.
63.           t.draw(main_sq, states);
64.
65.           this->lside->draw(t,states);
66.
67.           this->rside->draw(t,states);
68.
69.         }
70.
71.       };
72.
73.       #endif
```

PTree.cpp:

```cpp
#include "pTree.hpp"
#include <iostream>
#include <cmath>
#include <SFML/Graphics.hpp>
using namespace std;
using namespace sf;


pTree::pTree(double Len, int Num, double given_angle)
      {
        cnt = Num ;
        n = Len;
        deg = 45;
        angle = given_angle;
        main_sq.setSize(Vector2f(n, n));
        main_sq.setOrigin(main_sq.getPoint(3));
        main_sq.setPosition(n * 2.5,n * 4.0);

        main_sq.setOutlineColor(Color::Black);
        main_sq.setFillColor(Color::Red);

        nextL = main_sq.getTransform().transformPoint(main_sq.getPoint(
0));
        nextR= main_sq.getTransform().transformPoint(main_sq.getPoint(1
));
        pyth_tree();
      }

      pTree::pTree(double Len, int Num, double deg_X, Vector2f pos, dou
ble given_angle)
      {
        cnt = Num;
        n = Len;
        deg = deg_X;
        position = pos;
        angle = given_angle;
        main_sq.setSize(Vector2f(n, n));
        main_sq.setOrigin(main_sq.getPoint(3));
        main_sq.setPosition(position);
      main_sq.rotate( deg);
        main_sq.setOutlineColor(Color::Black);

        nextL = main_sq.getTransform().transformPoint(main_sq.getPoint(
0));
        nextR = main_sq.getTransform().transformPoint(main_sq.getPoint( 1));
        }


          void pTree::pyth_tree()
```

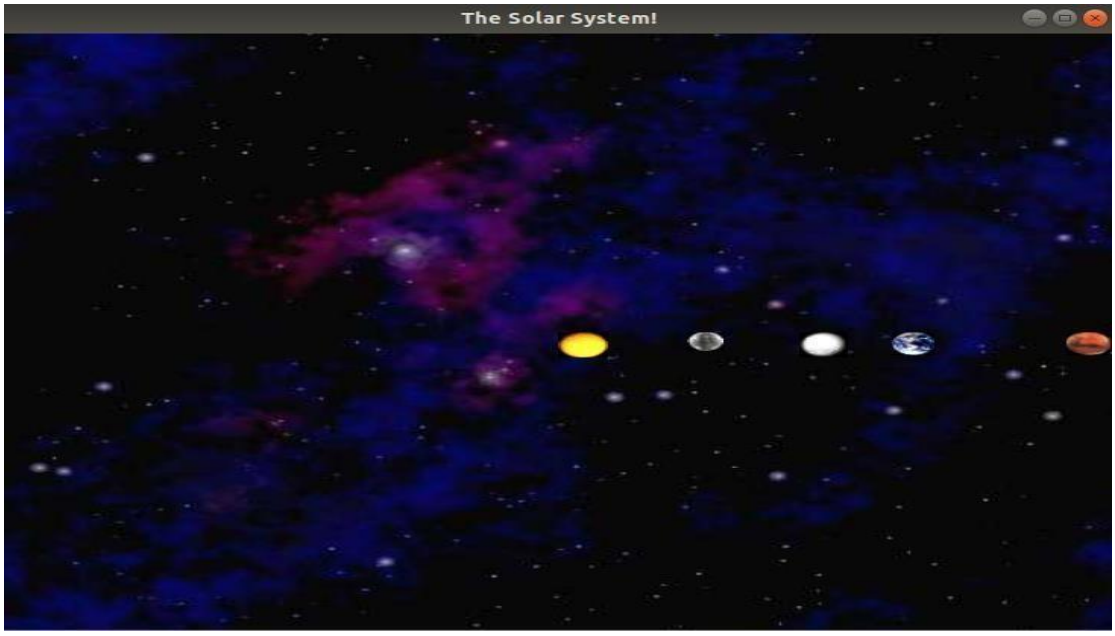
```
45.             {
46.             if(cnt < 0) return;
47.             double lrotation = -(angle) + main_sq.getRotation();
48.             double lsize = cos((angle)*M_PI/180.0) * n;
49.
50.             double rrotation = ( 90 - angle) + main_sq.getRotation();
51.             double rsize = sin(( angle)*M_PI/180.0) * n;
52.
53.             this-
    > lside = new pTree(lsize, cnt - 1, lrotation , nextL, angle);
54.             this-> lside -> deg = this-> lside -> main_sq.getRotation();
55.             this-> lside -
    > main_sq.setFillColor(Color(rand()%254, rand()%254, rand()%254));
56.
57.             this-
    > rside = new pTree(rsize, cnt - 1, rrotation, nextR, angle);
58.             this-> rside -> main_sq.setOrigin(rsize,rsize);
59.             this-> rside ->nextL = this-> rside -
    > main_sq.getTransform().transformPoint(this-> rside -
    > main_sq.getPoint(0));
60.             this-> rside -> nextR = this-> rside -
    > main_sq.getTransform().transformPoint(this-> rside -
    > main_sq.getPoint(1));
61.             this-> rside -> deg =  this-> rside -> main_sq.getRotation();
62.             this-> rside -
    > main_sq.setFillColor(Color(rand()%254, rand()%254, rand()%254));
63.             this-> lside ->pyth_tree();
64.             this-> rside ->pyth_tree();
65.             }
```

PS 3 (A) N-BODY SIMULATION

The task given in this assignment was to implement and display a static universe by reading a text file, which consisted of number of components or planets, radius, and each components position, its velocity, mass and image file name.
I was able to complete 100% of the assignment and also implemented a background image.

I've used vector of smart pointers, to store the planets objects from the text file.
Initially, the vector creates objects for every planet in the file and stdin is overloaded to read every variable for the planet. Thus, for every planet object the constructor is initiated and to initialize every variable of the planet, >> operator is overloaded. Now, the stdin initializes: x,y position, velocity, mass and image name.
Then for calculating the position of the planet, the radius is divided by the x-position of the planet and then multiplied with half of window size. This gives SFML's (0,0) point, so add window size/2 for the universe center point. Same calculations can be done for y-position.
The sf::Drawable is used to implement virtual void draw method in CelestialBody class for drawing sprites in the window. For creating these sprites, images are loaded using Texture object. Also used other SFML library features to set the window and scale to set the size of universe within the window.

To mainly implement this assignment, familiar concepts were used such as overloading stdin operator and using draw method. Other than that, I did learn to calculate and convert the coordinates of planets to SFML units.

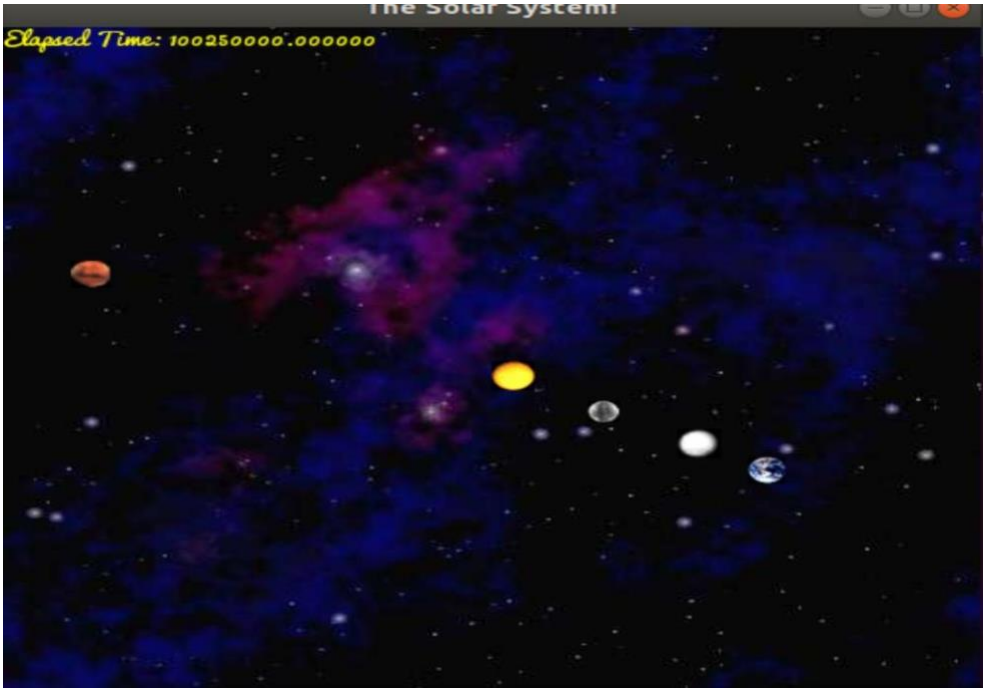


PS 3 (B) N-BODY SIMULATION

The main task in this assignment was to used physics simulation and add animation to the previous assignment. After building the static universe, make the planets move around the sun. This calculation of new positions can be done with the help of data in planets.txt file and implementing the data in the equation.
I was able to complete 100% of the assignment and also implemented a background image.

The core algorithm used in this assignment was to use implement the physics equations to calculate their positions, velocity and acceleration. I've also used celestialbody class, smart pointers and overloaded the operators as part of the assignment. The key concepts of physics that were used in this assignment were, Newtons Universal law of gravitation, Newtons law of second motion and the principle of superposition. These concepts were used to govern the forces acting on these bodies and acceleration of these bodies in space. In order for the simulation to look continuous, everything of CelestialBody class was recalculated for every change in delta time.
Other features of SFML were used for drawing sprites, displaying timee and playing sound.

I learned about physics simulation for implementing equations in the program. These equations were important as a little difference would mess with positioning of the planets. Other than that, the implementation of previous assignment was used for animation.



Makefile:

```
all:
	g++ -Wall -Werror -pedantic -c NBody.cpp
	g++ NBody.o -o NBody -lsfml-graphics -lsfml-window -lsfml-system lsfml-audio


clean:
	rm NBody NBody.o
```

NBody.cpp

```
1.  #include <iostream>
2.  #include <string>
3.  #include <vector>
4.  #include <memory>
5.  #include <cmath>
6.  #include <iomanip>
7.  #include <SFML/Audio.hpp>
8.  #include <SFML/Graphics.hpp>
9.  #include <SFML/System.hpp>
10. #include <SFML/Window.hpp>
11.
12. using namespace std;
13. using namespace sf;
14.
15. const int WSIZE = 500;
16. const double GRAVITY = 6.67 * pow(10, -11);
17.
18. class CelestialBody : public Drawable
19. {
20. public:
21.     CelestialBody(double R) : radius(R) { net_force = Vector2f(0,0); };
22.     void step(double seconds);
23.     friend istream &operator>>( istream  &input, CelestialBody &Body);
24.     friend ostream &operator<<( ostream  &out, CelestialBody &Body);
25.     void find_force(CelestialBody& Body);
26.     Vector2f getPosition() { return pos; }
27.     double getMass() { return mass; }
28. private:
29.     virtual void draw(RenderTarget& target, RenderStates states) const
30.     {
31.         target.draw(sprite, states);
32.     }
33.     Texture texture;
34.     Sprite sprite;
35.     Vector2f pos;
36.     Vector2f velocity;
37.     Vector2f acceleration;
38.     Vector2f net_force;
39.     double mass;
40.     double radius;
41.     string img;
42.
43. };
44.
45. void CelestialBody::step(double seconds)
46. {
47.     double x_acc = net_force.x / mass;
48.     double y_acc = net_force.y / mass;
49.     double x_vel = velocity.x + (seconds * x_acc);
50.     double y_vel = velocity.y + (seconds * y_acc);
51.     double x_pos = pos.x + (seconds * x_vel);
52.     double y_pos = pos.y + (seconds * y_vel);
53.     pos.x = x_pos ;
54.     pos.y = y_pos ;
55.     velocity.x = x_vel;
56.     velocity.y = y_vel;
57.     acceleration.x = x_acc;
58.     acceleration.y = y_acc;
59.     double x = ((pos.x/radius) * (WSIZE/2.0)) + (WSIZE/2.0);
```

```
60.     double y = ((pos.y/radius) * (WSIZE/2.0))  + (WSIZE/2.0);
61.     sprite.setPosition(y, x);
62.     net_force = Vector2f(0,0);
63.     //cout << sprite.getPosition().x << " " << sprite.getPosition().y << endl;
64. }
65.
66. istream& operator>>(istream  &input, CelestialBody &Body)
67. {
68.     input >> Body.pos.x >> Body.pos.y >> Body.velocity.x >> Body.velocity.y >> Body.mass >> Body.img;

69.     if (!Body.texture.loadFromFile(Body.img))
70.     {
71.         cout << "Failed to open the image. EXITING!!" << endl;
72.         exit(1);
73.     }
74.     Body.texture.setSmooth(true);
75.     Body.sprite.setTexture(Body.texture);
76.     double x_pos = ((Body.pos.x/Body.radius) * (WSIZE/2.0)) + (WSIZE/2.0);
77.     double y_pos = ((Body.pos.y/Body.radius) * (WSIZE/2.0)) + (WSIZE/2.0);
78.     Body.sprite.setPosition(y_pos, x_pos);
79.     return input;
80. }
81.
82. ostream& operator<<( ostream  &out, CelestialBody &Body)
83. {
84.     out << setw(15) << left << Body.pos.x << " " << setw(15) << left << Body.pos.y << " " << scientif
    ic << setw(15) << left << Body.velocity.x;
85.     out << " " << scientific << setw(15) << left << Body.velocity.y << " " << setw(15) << left << Bod
    y.mass << " " << setw(15) << left << Body.img;
86.     return out;
87. }
88.
89. void CelestialBody::find_force(CelestialBody& Body)
90. {
91.     double r = sqrt((pow(Body.pos.x - this->pos.x ,2) + pow(Body.pos.y - this->pos.y,2)));
92.     double pairWise_force = (GRAVITY * this->mass * Body.mass) / (r * r);
93.     double x_net = pairWise_force * ((Body.pos.x - this->pos.x) / r);
94.     double y_net = pairWise_force * ((Body.pos.y - this->pos.y) / r);
95.     this->net_force.x += x_net;
96.     this->net_force.y += y_net;
97. }
98.
99. int main(int argc, char** argv)
100.        {
101.            if(argc != 3)
102.            {
103.                cout << "Error" << endl;
104.
105.                exit(1);
106.            }
107.            int N;
108.            double R, time, delta_time, current_time = 0.0;
109.            time = atof(argv[1]);
110.            delta_time = atof(argv[2]);
111.
112.            cin >> N >> R;
113.
114.            vector<shared_ptr<CelestialBody>> Body;
115.            for(int i = 0 ; i < N; i++)
116.            {
117.                shared_ptr<CelestialBody> temp = make_shared<CelestialBody>(R);
118.                cin >> (*temp);
119.                Body.push_back(temp);
```

```
120.          }
121.
122.          Texture texture;
123.          if (!texture.loadFromFile("starfield.jpg"))
124.          {
125.              cout << "Error opening image" << endl;
126.              exit(1);
127.          }
128.          texture.setSmooth(true);
129.          Sprite sprite(texture);
130.          sprite.setScale(WSIZE/500.0, WSIZE/500.0);
131.          RenderWindow window(sf::VideoMode(WSIZE, WSIZE), "The Solar System!");
132.          window.setVerticalSyncEnabled(true);
133.          window.setFramerateLimit(40);
134.
135.         Font font;
136.         font.loadFromFile("Pacifico.ttf");
137.          Text text;
138.          string temp = "Time Elapsed: " + to_string(current_time);
139.          text.setString(temp);
140.          text.setFont(font);
141.          text.setCharacterSize(14);
142.          text.setFillColor(Color::Yellow);
143.
144.          Music music;
145.          if(!music.openFromFile("2001.wav"))
146.          {
147.              return -1;
148.          }
149.
150.          music.play();
151.
152.          while (window.isOpen())
153.              {
154.              Event event;
155.              while (window.pollEvent(event))
156.                  {
157.                      if (event.type == sf::Event::Closed)
158.                          window.close();
159.                  }
160.              window.clear(Color(243,243,243));
161.              window.draw(sprite);
162.              window.draw(text);
163.              for(int i = 0 ; i < N; i++)
164.              {
165.                  for(int j = 0 ; j < N; j++)
166.                  {
167.                      if(i != j)
168.                      {
169.                          Body[i]->find_force(*Body[j]);
170.                      }
171.                  }
172.              }
173.
174.              for(auto i : Body)
175.              {
176.                  window.draw(*i);
177.                  if(current_time < time)
178.                  {
179.                      i->step(delta_time);
180.                  }
181.              }
182.
```

```
183.                 window.display();
184.                 current_time += delta_time;
185.                 text.setString("Elapsed Time: " + to_string(current_time));
186.                 if(current_time > time)
187.                 {
188.                     break;
189.                 }
190.
191.                 }
192.
193.             cout << N << endl;
194.             cout << R << endl;
195.             for(int j = 0 ; j < N; j++)
196.             {
197.                 cout << *Body[j] << endl;
198.             }
199.
200.             return 0;
201.         }
```

PS4 (A) SYNTHESIZING A PLUCKED STRING SOUND:

The main task of this assignment is to build an algorithm to simulate plucking a guitar string. To test the program, I implemented exception handling mechanism in StringSound implementation. I did complete the whole assignment and tested each method using test cases to check their functionality.

To build an algorithm, I created a double array of buffer, for storing N samples which will be used in next assignment for Karplus-Strong algorithm.

The algorithm in this assignment deletes the initial value from RingBuffer array and adds it to the end using enqueue and dequeue methods.

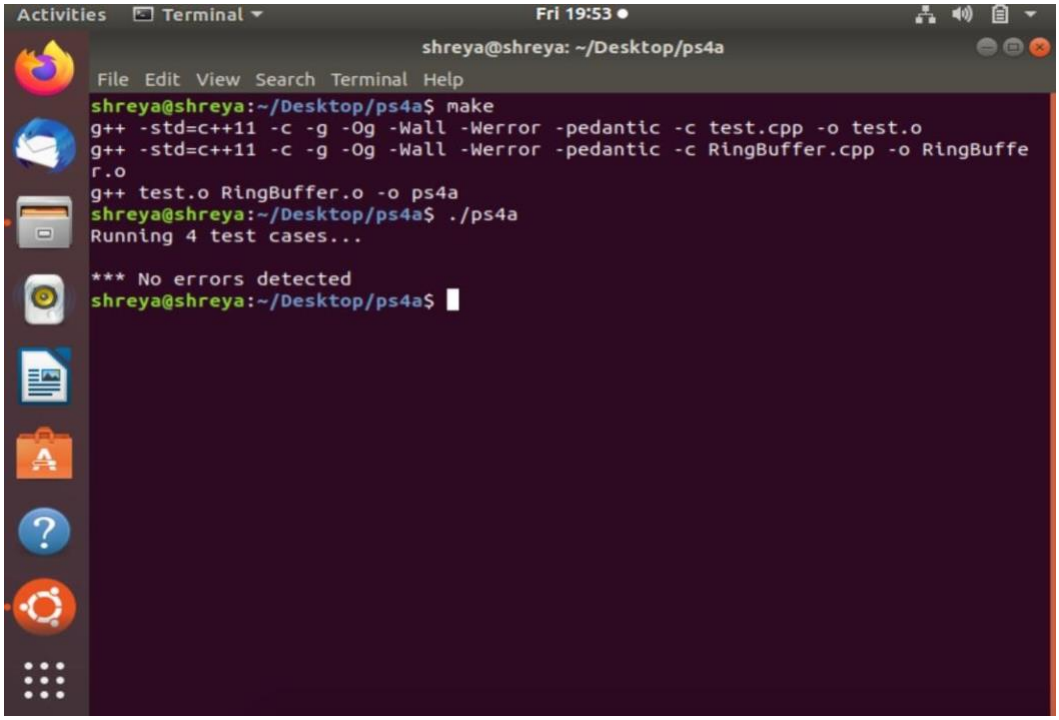
It also checks whether the buffer array is full or empty using isFull and isEmpty methods.

The peek method returns the initial element from the buffer.

The program also raises standard exceptions when the capacity created for buffer is less than 1 in size, or when you try to remove an element from an empty buffer or when to enqueue a full array.

To verify every method in my assignment is working I used Boost test library functions for every method. I also used boost test to verify whether the program throws an exception when the buffer array or method is used inappropriately. Such as when you try to add an element to full array or remove an element from empty array.

I learned to write a program which produced no errors when run by cpplint. Also, learned a lot about implementing exceptions using exception and boost unit testing.



PS4 (B) SYNTHESIZING A PLUCKED STRING SOUND:

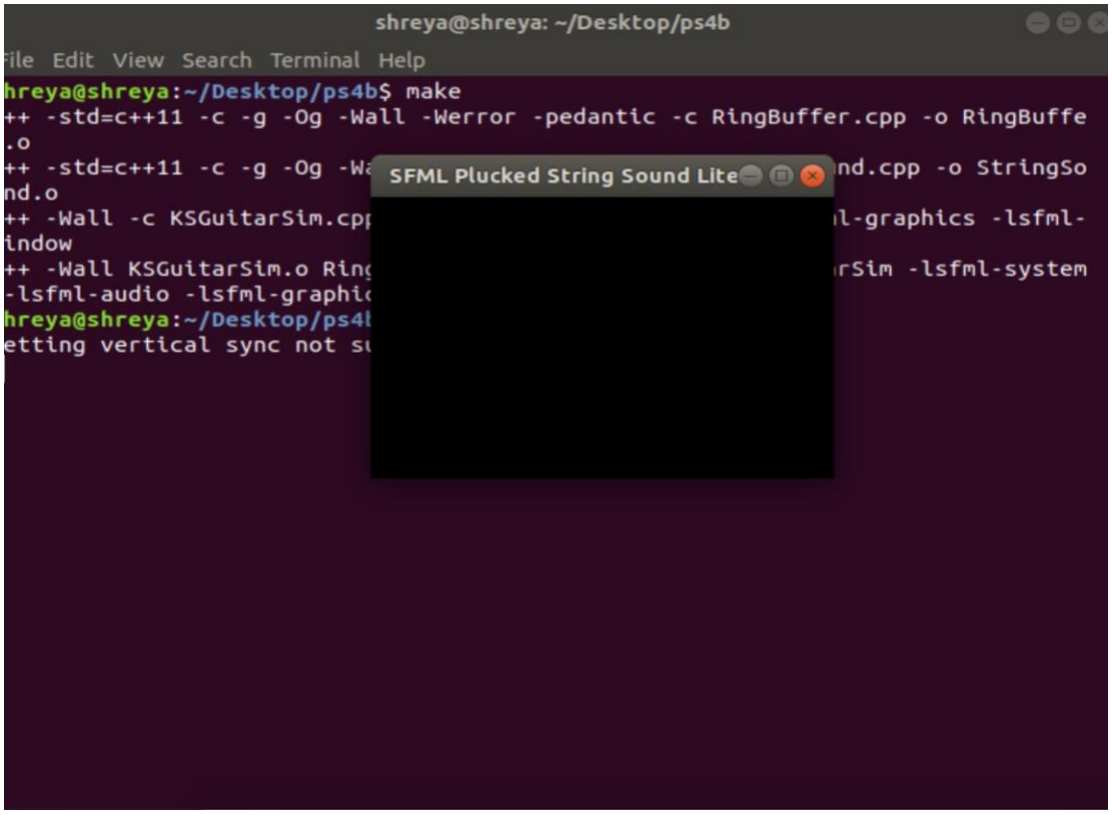
The main idea of this assignment is to generate a stream of string samples for generating audio under keyboard control, by using Karplus-Strong guitar simulation. That is, too calculate the frequency of the note provided as keyboard input and produce the sound using SFML libraries. Also, to test the program, I implemented exception handling mechanism in StringSound and KSGuitarSim implementation.

The idea of Karplus-Strong algorithm is used to simulate plucking a guitar string. It takes average of first two values and multiplies it to energy decay factor that is .996. The RingBuffer is created by deleting the initial value from RingBuffer array and adds it to the end using enqueue and dequeue methods. The algorithm along with the RingBuffer created in previous assignment, model sound in a style like a guitar string is being plucked.

Also, exceptions were implemented to check whether the program produced any errors. The constructor StringSound (double frequency) throws an invalid_argument exception if the frequency is less than or equal to zero or the buffer is less than 2.

The other constructor StringSound (vector<sf::Int16> init) also throws an invalid_argument exception if the size of array is less than 2. The main KSGuitarSim implementation also throws a runtime exception, if there's an error to load from the samples.

Also, to implement this guitar string assignment, I learned to play around with SFML's keyboard library to control a guitar. Other than that, Karplus-Strong Algorithm helped me understand about generating audio in a program.



Makefile:

```
all:
g++ -std=c++11 -c -g -Og -Wall -Werror -pedantic c RingBuffer.cpp -o RingBuffer.o
g++ -std=c++11 -c -g -Og -Wall -Werror -pedantic c StringSound.cpp -o StringSound.o
g++ -Wall -c KSGuitarSim.cpp -lsfml-system -lsfml-audio -lsfmlgraphics -lsfml-window
g++ -Wall KSGuitarSim.o RingBuffer.o StringSound.o -o KSGuitarSim lsfml-system -lsfml-audio -lsfml-graphics -lsfml-window

	clean:
	rm *.o
	rm KSGuitarSim
```

KSGuitarSim.cpp:

```cpp
#include "StringSound.h"

using namespace sf;

#define CONCERT_A 440.0
#define SAMPLES_PER_SEC 44100

vector<sf::Int16> makeSamplesFromString(StringSound& ss) {
    std::vector<sf::Int16> sample;

    ss.pluck();
    int duration = 8;  // seconds
    int i;
    for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
            ss.tic();
            sample.push_back(ss.sample());
    }

    return sample;
}

int main() {
    sf::RenderWindow window(sf::VideoMode(300, 200), "SFML Plucked String Sound Lite");
    sf::Event event;
    string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";

    vector<std::vector<sf::Int16> > samples1;
    vector<sf::SoundBuffer> SB_1;
    vector<sf::Sound> sound_1;
    samples1.reserve(37);
    SB_1.resize(37);
    sound_1.resize(37);
    for (double i = 0; i < 37; i++)
    {
        StringSound gs1(CONCERT_A * pow(2, (i-24)/12.0));
        samples1.push_back(makeSamplesFromString(gs1));
        if (!(SB_1[i]).loadFromSamples(&samples1[i][0],samples1[i].size(), 2, SAMPLES_PER_SEC))
        throw std::runtime_error("Error loading from samples.");
        sound_1[i].setBuffer(SB_1[i]);
    }

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

            if(event.type == Event::TextEntered)
            {
                for (int i = 0; i < 37; i++)
                {
                    if (event.text.unicode == (unsigned)keyboard[i])
                    {
                        sound_1[i].play();
                    }
                }
            }
```

```
61.
62.         window.clear();
63.         window.display();
64.         }
65.     }
66.     return 0;
67. }
```

StringSound.hpp

```
#include <math.h>
#include <limits.h>
#include <iostream>
#include <string>
#include <exception>
#include <stdexcept>
#include <vector>
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
      #include <SFML/Audio.hpp>
      #include <SFML/Window.hpp>
      #include "RingBuffer.h"

      #define SAMPLING RATE 44100;
      using namespace std;

            class StringSound

            {
            public:
            StringSound(double frequency); // create a guitar string of t he given frequency using a sampling rate of 44,100
              StringSound(vector<sf::Int16> init); // create a guitar strin g with size and initial values are given by the vector
            ~StringSound() { delete ringbuffer_pointr; } // destructor to  explicitly deleting ring buffer object;
            void pluck(); // pluck the guitar string by replacing the buf fer with random values, representing white noise
            void tic(); // advance the simulation one time step

            sf::Int16 sample(); // return the current sample
            int time(); // return number of times tic was called so far
             int get_capacity() { return cap; }          void reset_time();

                private:
            RingBuffer* ringbuffer_pointr;

            int num;
      int cap;
      };
```

StringSound.cpp

```
1. #include  "StringSound.h"    2.
3.
4.                          StringSound::StringSound(  double  frequency)
5.                          {
6.                          int size_n = ceil(44100 / frequency);    // calculating size of buffer
7.                          if (frequency<=0 || size_n<2){
8.                          throw  std::invalid_argument(  "Error: insufficient frequency or buffer si ze."    );}
9.                          ringbuffer_pointr =   new  RingBuffer(size_n);
10.                         for(int i = 0; i < size_n; i++)
11.                         {
12.                         ringbuffer_pointr ->enqueue(0);
13.                         }
14.                         num = 0;
15.                         cap = size_n;
16.             }
17.
18.                         StringSound::StringSound(vector<sf::Int16> init)
19.             {
20.                                 ringbuffer_pointr = new RingBuffer(init.size());
21.                         cap = init.size();
22.                     if( cap< 2) {
23. throw std::invalid_argument("Error: insufficient size of array.") ;}
24.
25.                             for(int i = 0; i < cap; i++)
26.                     {
27.                                     ringbuffer_pointr->enqueue(init.at(i));
28.                     }
29.                      num = 0;
30.             }
31.
32.                      void StringSound::pluck()
33.             {
34.                             ringbuffer_pointr->mt_buffer();
35.                             for(int i = 0; i < cap; i++)
36.                     {
37.                                 ringbuffer_pointr-
    >enqueue((int16_t)(rand() & 0xffff)); // fill with random numbers over the int16 t range
```

```
38.                 }
39.         }
40.
41.                 void StringSound::tic()
42.         {
43.                         int temp = ringbuffer_pointr->dequeue();
44.   int new_value = 0.996 * ((temp + ringbuffer_pointr>peek()) / 2.0);
45.                         ringbuffer_pointr->enqueue(new_value);
46.                 num++;
47.         }
48.
49.                  sf::Int16 StringSound::sample()
50.         {
51.                         return ringbuffer_pointr->peek();
52.         }
53.
54.                 int StringSound::time()
55.         {
56.                  return num;
57.         }
58.
59.                  void StringSound::reset_time()
60.         {
61.                 num = 0;
62.         }
```

RingBuffer.hpp

```
1. /*Copyright 2020 Shreya Patil*/
2. #ifndef _HOME_SHREYA_DESKTOP_PS4B_RINGBUFFER_H_
3. #define _HOME_SHREYA_DESKTOP_PS4B_RINGBUFFER_H_ 4.
5. #include <stdint.h>
6. #include <iostream>
7. #include <string>
8. #include <exception>
9. #include <stdexcept> 10. #include <queue>
11.
12. class RingBuffer {
13. public:
14. explicit RingBuffer(int capacity); // creates an empty buffe r
15. ~RingBuffer() { delete[] buffer; }
16. int size(); // returns number of items currently in the buff er
17. bool isEmpty(); // is the buffer empty (size equals zero)?
18. bool isFull(); // is the buffer full (size equals capacity) ?
19. void enqueue(int16_t x); // add item x to the end
20. int16_t dequeue(); // delete and return item from the front
21. int16_t peek(); // return (but do not delete) item from the front
22. void mt_buffer();
23. private:
24. double* buffer;
25. int buff_size;
26. int cap;
27. int initial;
28. int last; 29. };
30.
31. #endif // _HOME_SHREYA_DESKTOP_PS4B_RINGBUFFER_H_
```

RingBuffer.cpp:

```
1.  /*Copyright 2020 Shreya Patil*/
2.    #include "/home/shreya/Desktop/ps4b/RingBuffer.h"
3.    #include <string>
4.    using std::int16_t;
5.    using std::string;
6.    using std::invalid_argument;
7.    using std::runtime_error;
8.
9.    RingBuffer::RingBuffer(int capacity) {
10.     if ( capacity < 1 ) {
11.       string err_msg = "RingBuffer constructor: capacity must be > 0.";
12.         throw invalid_argument(err_msg);
13.         return;
14.     }
15.     buffer = new double[capacity];
16.     buff_size = 0;
17.     cap = capacity;
18.     initial = 0;
19.     last = 0;
20. }
21.
22. int RingBuffer::size() {
23.     return buff_size;
24. }
25.
26. bool RingBuffer::isEmpty() {
27.     if (buff_size == 0) return true;
28.     return false;
29. }
30.
31. bool RingBuffer::isFull() {
32.     if (buff_size >= cap) {
33.         return true;
34.     }
35.     return false;
36. }
37.
38. void RingBuffer::enqueue(int16_t x) {
39.     if (isFull()) {
40.         throw runtime_error("enqueue: can't enqueue to a full ring.");
41.         return;
42.     }
43.     buff_size++;
44.     buffer[last] = x;
45.     last++;
46.     if (last == cap) {
47. last = 0;}
48. }
49.
50.             int16_t RingBuffer::dequeue() {
51.                 if (isEmpty()) {
52.                     throw runtime_error("dequeue: can't dequeue to an empty ring.");
53.                     return -100;
54.                 }
55.                 buff_size--;
```

```
    int16_t temp = buffer[initial];
    buffer[initial] = 0;
    initial++;
    if (initial == cap) {
initial = 0;}
    return temp;
}

int16_t RingBuffer::peek() {
    if (isEmpty()) {
        throw runtime_error("peek: can't peek to a empty ring.");
        throw -100;
    }
    return buffer[initial];
}
void RingBuffer::mt_buffer()
{
    initial = 0;
    last = 0;
    buff_size = 0;
}
```

test.cpp:

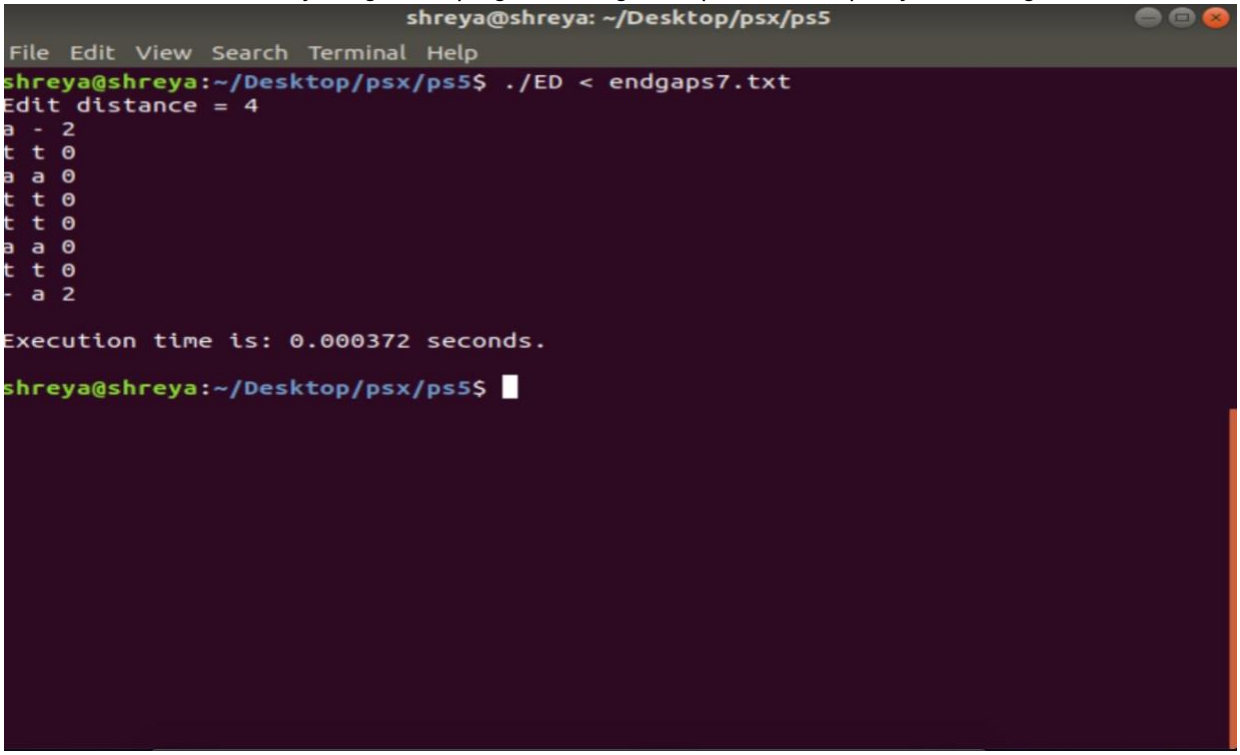
```
/*copyright 2020 Shreya Patil*/
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/included/unit_test.hpp>  5. #include "RingBuffer.h"  6.
        BOOST_AUTO_TEST_CASE(RingBufferConstructor) {
        BOOST_REQUIRE_NO_THROW(RingBuffer(100));  // initializes a construc tor
        BOOST_REQUIRE_THROW(RingBuffer(0), std::exception);  // throwing ex ception
        BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument);
        // throws an invalid argument exception
        }
 13.
        BOOST_AUTO_TEST_CASE(RingBuffer_enque_dequeue) {
        RingBuffer obj(100);
        obj.enqueue(2);  17.        obj.enqueue(1);
        obj.enqueue(0);
        BOOST_REQUIRE(obj.dequeue() == 2);  20.
         BOOST_REQUIRE(obj.dequeue() == 1);
    BOOST_REQUIRE(obj.dequeue() == 0);
    // throws a runtime_error exception
    BOOST_REQUIRE_THROW(obj.dequeue(), std::runtime_error);
    }  25.
    BOOST_AUTO_TEST_CASE(RingBuffer_BadCapacity) {
    BOOST_REQUIRE_THROW(RingBuffer(-1), std::invalid_argument);
    }  29.
        // exercising all other methods
        BOOST_AUTO_TEST_CASE(peek_size_Empty_Full) {
        RingBuffer obj(10);
        BOOST_REQUIRE(obj.isEmpty() == 1);
        obj.enqueue(2);
        BOOST_REQUIRE(obj.isEmpty() == 0);
        BOOST_REQUIRE(obj.peek() == 2);
        BOOST_REQUIRE(obj.size() == 1);
        BOOST_REQUIRE(obj.isFull() == 0);
        }
```

PS5 DNA SEQUENCE ALIGNMENT

The main task of this assignment is to implement a program to find optimal alignment between given 2 strings. To implement this assignment, I've used dynamic programming approach. Also, to use valgrind, a memory analysis tool and to understand time and space performance of the program.

The key to implement this algorithm of dynamic programming approach is by breaking the larger problem into smaller problems, optimized solution and solve the original problem. To do so, find the NxM matrix is implemented. The edit distances of the bottom most row and right most column are calculated, which in turn calculate the next round of edit distances. This process continues until [0][0] matrix is reached. The C-style arrays are used along with pointers to the current square and its neighbors and also flagging pointer points to the chosen square.
The alignment can be discovered by retracing the steps of the approach backwards and rediscovering the path of choices.
To analyze the amount of memory used, valgrind tool runs the code and produces a log file massif.out which can be examined for memory usage.

I learned about Needleman-Wunsch method to find the optimal solution for the given problem even though it was N-squared in space. Also, that using cstyled arrays give efficient memory usage. Learning about valgrind and understand the memory usage for a program on a given input was also pretty interesting.



Makefile:

```
CC = g++ -std=c++0x
CFLAGS = -Wall -Werror -ansi -pedantic -g

all: ED

ED: main.o ED.o
    $(CC) $(CFLAGS) -o ED main.o ED.o -lsfml-system

main.o: main.cpp ED.cpp

    $(CC) $(CFLAGS) -c main.cpp

ED.o: ED.cpp ED.hpp
    $(CC) $(CFLAGS) -c ED.cpp

clean:
    rm -rf ED *.o *.~
```

main.cpp:

```cpp
#include <SFML/System.hpp>
#include "ED.hpp"
#include <iostream>
#include <string>

int main(int argc, char* argv[])
{
    std::string s;
    std::string t;


        std::cin >> s;
        std::cin >> t;

        int edit_distance;
        std::string edit_distance_path;

        sf::Clock clock;
        sf::Time time;

        ED my_ed(s,t);


        edit_distance = my_ed.OptDistance();


        edit_distance_path = my_ed.Alignment();

        std::cout << "Edit distance = " << edit_distance << std::endl;
        std::cout << edit_distance_path << std::endl;

        time = clock.getElapsedTime();
        std::cout << "Execution time is: " << time.asSeconds() << " seconds.";
        std::cout << std::endl << std::endl;

        return 0;
    }
```

ED.hpp:

```
1. #include  <string       >
2. #include <  vector >
3.
4. class  ED
5. {
6. public :
7.
8.      ED(std::string, std::string);   //  a  constructor  that  accepts  2
       strin gs
9.      ~ED();
10.
11.           void       BaseMatrix();
12.           static   int   penalty(char      , char      ) ;  //  returns
              penalty  for  aligni ng  characters
13.           static  int   min(int  &, int  &, int  &);  //returns  minimum
              of 3   argu ments
14.           int   OptDistance();  //returns  optimal  distance
15.           std::string  Alignment();  //returns  a  string  to  display
              actual  alignment
16.
17.          // for debugging
18.          friend std::ostream& operator <<(std::ostream&, const ED&);
19.                  //used for debugging and overloading the input operator.
20.
21.         private:
23.           std::string s1;
24.           std::string s2;
25.
26.           unsigned int M, N;
```

```
27.                 int **Matrix;
28.
29.             };
```

ED.cpp:

```cpp
/*Copyright 2020 Shreya Patil*/
#include "/home/shreya/Desktop/psx/ps5/ED.h"
#include <string>
#include <vector>
#include <iostream>
using std::string;
const int GAP = 0;
const int MISMATCH = 1;
const int MATCH = 2;


ED::ED(std::string s, std::string t) : s1(s), s2(t) {
s1.append("-");
s2.append("-");
M = s1.length();
N = s2.length();
Matrix = new int*[M];
for (unsigned int i = 0; i < M; i++) {
    Matrix[i] = new int[N];
}
}

ED::~ED() {
for (unsigned int i = 0; i < M; i++) {
    delete [] Matrix[i];
}
delete Matrix;
}

void ED::BaseMatrix() {
std::string subs, subt;
for (unsigned int i = M; i > 0; i--) {
    subs = s1.substr(i);
    for (unsigned int j = N; j > 0; j--) {
        subt = s2.substr(j);
        if (i == M)
          Matrix[i-1][j-1] = (N - (N - subt.length()) ) * MATCH;
        if (j == N)
          Matrix[i-1][j-1] = (M - (M - subs.length()) ) * MATCH;
}
}
}
int ED::penalty(char a, char b) {
return ((a == b) ?  GAP : MISMATCH);
}
int ED::min_val(int& a, int& b, int& c) {
if(b < a) {



return (b < c) ? b : c;
}
```

```
return (a < c) ? a : c;
}

int ED::OptDistance() {
int opt_distance;
BaseMatrix();
for(unsigned int i = M-1; i > 0; i--) {
for(unsigned int j = N-1; j > 0; j--) {
    int bottom = Matrix[i][j-1] + MATCH;
    int right = Matrix[i-1][j] + MATCH;
    int diag = Matrix[i][j] + (penalty(s1.at(i-1), s2.at(j-1)));
    opt_distance = min_val(bottom, right, diag);
    Matrix[i-1][j-1] = opt_distance;
}
}
return opt_distance;
}

std::string ED::Alignment() {
std::string alignment;
unsigned int i = 0, j = 0;
int* currentptr = &Matrix[i][j];
int* rightptr = &Matrix[i][j+1];
int* diagptr = &Matrix[i+1][j+1];
int* bottomptr = &Matrix[i+1][j];
int* chosenptr;
while(i < M-1 || j < N-1) {
if(j < N-1 && (*currentptr - *rightptr) == MATCH) {
alignment.append("-");
alignment.append(" ");
alignment.push_back(s2.at(j));
alignment.append(" ");
alignment.append("2");
alignment.append("\n");
chosenptr = rightptr;
j++;
} else if ((*currentptr - *diagptr) == penalty(s1.at(i), s2.at(j))) {
alignment.push_back(s1.at(i));
alignment.append(" ");
alignment.push_back(s2.at(j));
alignment.append(" ");
if (*currentptr - *diagptr == GAP)
alignment.append("0");
if (*currentptr - *diagptr == MISMATCH)
alignment.append("1");
alignment.append("\n");
chosenptr = diagptr;
i++;
j++;
```

```
99.  } else if ((i < M-1) && (*currentptr - *bottomptr) == MATCH) {
100.alignment.push_back(s1.at(i));
101.alignment.append(" ");
102.alignment.append("-");
103.alignment.append(" ");
104.alignment.append("2");
105.alignment.append("\n");
106.chosenptr = bottomptr;
107.i++;
108.}
109.
110.currentptr = chosenptr;
111.rightptr = &Matrix[i][j+1];
112.diagptr = &Matrix[i+1][j+1];
113.bottomptr = &Matrix[i+1][j];
114.}
115.return alignment;
116.}
117.
118.std::ostream& operator <<(std::ostream& os, const ED& ed) {
119.std::cout << "    ";
120.for(unsigned int i = 0; i < ed.N; i++)
121.std::cout << "    " << ed.s2.at(i);
122.std::cout << std::endl;
123.for(unsigned int i = 0; i < ed.M; i++) {
124.std::cout << std::endl << ed.s1.at(i) << "   ";
125.for(unsigned int j = 0; j < ed.N; j++) {
126.if(ed.Matrix[i][j] > 9)
127.    std::cout << "   " << ed.Matrix[i][j];
128.else
129.std::cout << "    " << ed.Matrix[i][j];
130.}
131.}
132.std::cout << std::endl;
133.return os;
134.}
```

PS6 MARKOV MODEL OF NATURAL LANGUAGE

To implement this assignment, we've to build a model which generates reasonable random text, by analyzing the given text and producing a probabilistic model for transitions between k-grams. The main task of this assignment was to generate random text, using Markov model. I was able to complete 100% of this assignment. The text generator code takes an integer for building kgrams and an integer as length of text to be simulated as output for the given input text. Exceptions are thrown in case if k-gram length is not same as order, or if the input text is too small or when no such k-gram exists. The program also passes cpplint test and uses boost functions to test all function in the Model.

Initially, all the characters in the string were parsed and stored by a variable. In the constructor to parse the characters from the given input string I used sort method. Then, by the given input string all k-grams and k plus one-grams were created. After creating the k gram and k plus one gram, iterators were used to place them in map. Also, vectors were used to store characters that follow the k-gram depending on the number of times letter is found after k-gram.

For the wrap-around index, I appended the start of the text to final to form a circular string. Then the frequency functions determined the possibility of next character by iterating the map. Then kRand() generates random character by picking a character at random index in the vector. The generated function returns the generated string to the provided length by the user. The output generated by TextGenerator program defies that every function is working.

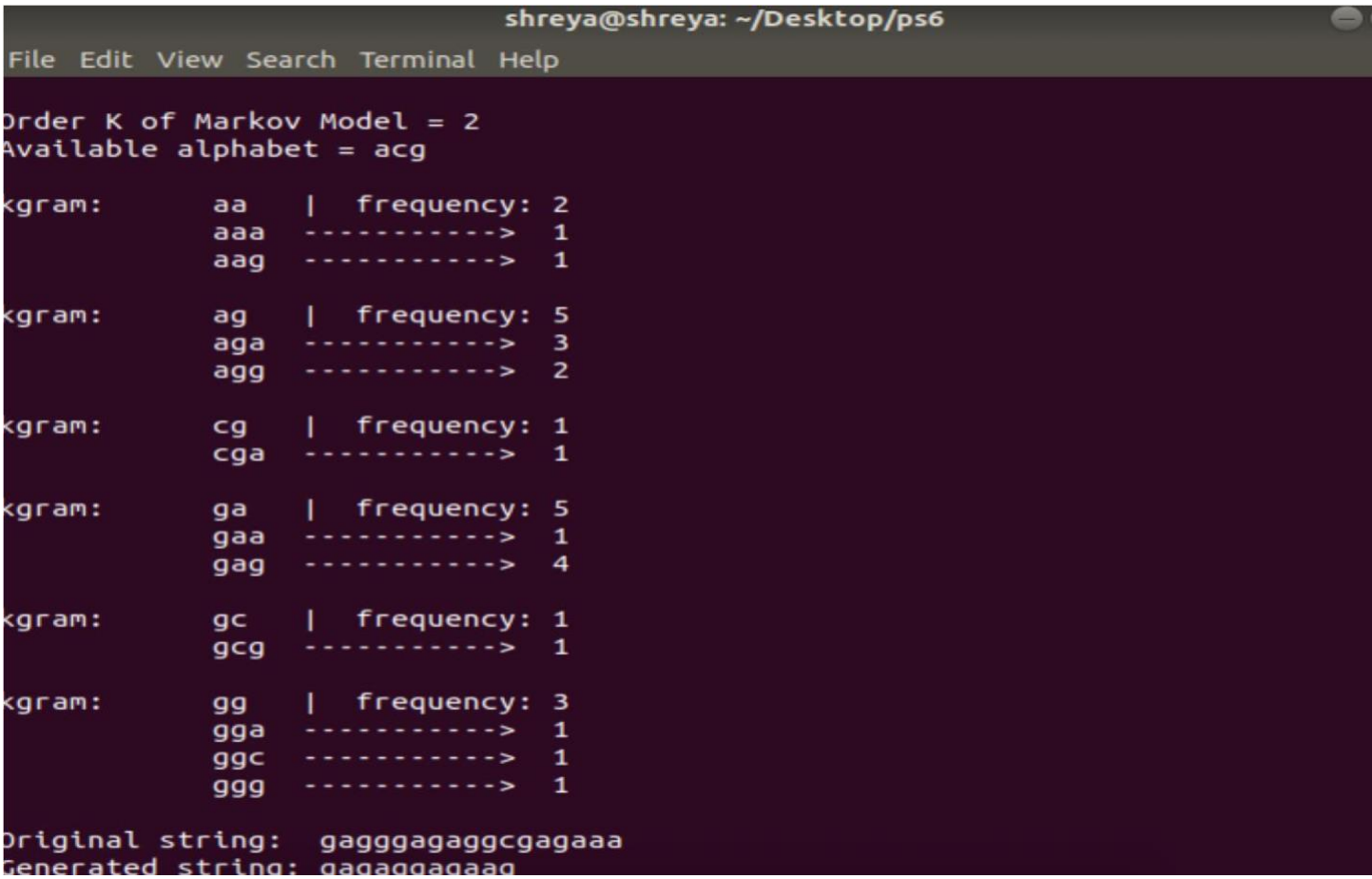
The main runtime exceptions were: frequency excption i.e., kgram length wasnt same as order, if ((unsigned)order != kgram.length()) throw std::runtime_error(frequency_excptn); constructor error i.e., Given input text wasnt greater than order

if ((unsigned)order >= text1.length()) throw std::runtime_error(constructor_error);} and kgram_exceptn when no such Kgram was found.

```
 if (freq(kgram) == 0) {
       throw std::runtime_error(kgram_exceptn);}
```

I learnt a lot about Markov modeling and its chains. Also how it is used by various text prediction programs for finding next probabilistic character and thus creating a sentence.

However, the code seems to throw an error when multiple lines are given as input.


shreya@shreya: ~/Desktop/ps6
File Edit View Search Terminal Help
Order K of Markov Model = 2
Available alphabet = acg
kgram: aa | frequency: 2
aaa ----------> 1
aag ----------> 1
kgram: ag | frequency: 5
aga ----------> 3
agg ----------> 2
kgram: cg | frequency: 1
cga ----------> 1
kgram: ga | frequency: 5
gaa ----------> 1
gag ----------> 4
kgram: gc | frequency: 1
gcg ----------> 1
kgram: gg | frequency: 3
gga ----------> 1
ggc ----------> 1
ggg ----------> 1
Original string: gagggagaggcgagaaa
Generated string: gagaggagaaag

Makefile:

```
CC = g++
CFLAGS = -g -Wall -W -ansi -pedantic
SFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio Boost = -lboost_unit_test_framework

all: TextGenerator boosttest

	TextGenerator: TextGenerator.o MModel.o
	$(CC) TextGenerator.o MModel.o -o TextGenerato r

	boosttest: boosttest.o MModel.o
	$(CC) boosttest.o MModel.o -o boosttest $(Boost)

TextGenerator.o:TextGenerator.cpp MModel.h $(CC) -c TextGenerator.cpp MModel.h $(CFLAGS)

	MModel.o:MModel.cpp MModel.h
	$(CC) -c MModel.cpp MModel.h $(CFLAGS)

boosttest.o:boosttest.cpp $(CC) -c boosttest.cpp $(Boost)

	clean:
	rm *.o

	rm TextGenerator
	rm boosttest
```

TextGenerator.cpp

```
1. /*Copyright 2020 Shreya Patil*/
2.
3. #include <time.h>
4. #include <stdlib.h>
5. #include <exception>
6. #include <stdexcept>
7. #include <string>
8. #include <iostream>
9. #include "/home/shreya/Desktop/ps6/MModel.h"
10.
11.       const int USAGE = 1;
12.       std::string improper_input = "STD INPUT:./TextGenerator <order> <
    kgram> <text>";  // NOLINT
13.
14.       int main(int argc, char* argv[]) {
15.           if (argc == USAGE) throw std::runtime_error(improper_input);
16.
17.           int k = atoi(argv[1]);
18.           int t = atoi(argv[2]);
19.           std::string input, kgram;
20.           std::cin >> input;
21.
22.           MModel mm(input, k);
23.         std::cout << mm << std::endl;
24.
25.             srand(time(NULL));
26.             kgram = input.substr(0, k);
```

```
    std::string generated_text = mm.generate(kgram, t);
27.
28.
29.     std::cout << "Original string:  " << input << std::endl;
30.     std::cout << "Generated string: " << generated_text << std::e ndl;
        std::cout << std::endl;
31.
32.
33.     return 0;
        }
34.
```

MModel.h:

```
1. /*Copyright 2020 Shreya Patil*/
2. #ifndef _HOME_SHREYA_DESKTOP_PS6_MMODEL_H_
3. #define _HOME_SHREYA_DESKTOP_PS6_MMODEL_H_
4. #include <map>
5. #include <string>
6.
7. class MModel {
8. public:
9.         MModel(std::string text, int k);
10.        ~MModel();
11.
12.        int kOrder();
13.        int freq(std::string kgram);
14.        int freq(std::string kgram, char c);
15.        char kRand(std::string kgram);
16.        std::string generate(std::string kgram, int L);
17.
18.    friend std::ostream& operator <<(std::ostream& os, MModel& mm );
19.
20.        private:
21.        int order;
22.        std::string alpha;
23.        std::string text1;
24.        std::map <std::string, int> kgrams;
25.        };
26.        #endif // _HOME_SHREYA_DESKTOP_PS6_MMODEL_H_
```

MModel.cpp

```
1.  /*Copyright 2020 Shreya  Patil*/
2.  #include <map>
3.  #include <vector>
4.  #include <exception>
5.  #include <stdexcept>
6.  #include <iostream>
7.  #include <string>
8.  #include <algorithm>
9.  #include "/home/shreya/Desktop/ps6/MModel.h"
10.
11.             std::string frequency_excptn =  " kgram length should be same as o rder"  ; //
                NOLINT
12.             std::string constructor_error =  "Given input text should be great er than order"  ; //
                NOLINT
13.             std::string kgram_exceptn =  "There is no such Kgram"  ; // NOLINT
14.             // implementing a constructor
15.             MModel::MModel(std::string text,  int k) : order(k), alpha(text) {
16.           int kgram_final;
17.
18.           text1 = text;
19.           if ((unsigned)order >= text1.length()) {
20.           throw std::runtime_error(constructor_error);}
21.
22.                       // Sorting the alphabet out of the input string
23.                     std::sort(alpha.begin(), alpha.end());
24.  alpha.erase(std::unique(alpha.begin(), alpha.end()), alpha.en d());
25.
26.                   // creating k and k -plus one grams
27.                       for (unsigned  int i = 1; i < text.length()+1; i++) {
28.                     std::string k_gram, k_plus_one, wrapped;
29.                   int lengthOfwrapped;
30.
31.                    kgram_final = i+k -1;
```

```
                                if ((unsigned)kgram_final >= text.length()) {
                                    lengthOfwrapped = kgram_final - text.length() + 1;
                                wrapped = text.substr(0, lengthOfwrapped);
                                    k_gram = text.substr(i, text.length()-1) + wrapped;
                                k_plus_one = k_gram + text.at(lengthOfwrapped);
                        } else {
                                k_gram = text.substr(i, k);
                                    if ((unsigned)kgram_final+1 >= text.length()) {
                                    k_plus_one = k_gram + text.at(0);
                                } else {
                                        k_plus_one = text.substr(i, k+1);
                                }
                        }

                                    std::map<std::string, int>::iterator kgram_iterator;
                                kgram_iterator = kgrams.find(k_gram);
                                  if (kgram_iterator != kgrams.end()) {
                                    kgram_iterator-> second += 1;
                                } else {kgrams[k_gram] = 1;}
                                        std::map<std::string, int>::iterator kplusone_iterator;
                                  kplusone_iterator = kgrams.find(k_plus_one);
                                   if (kplusone_iterator != kgrams.end()) {
                                    kplusone_iterator-> second += 1;
                                } else {kgrams[k_plus_one] = 1;}
                    }
            }

              MModel::~MModel() {
            }

                int MModel::kOrder() {
                   return order;
            }

                    int MModel::freq(std::string kgram) {
                        if ((unsigned)order != kgram.length()) {
                        throw std::runtime_error(frequency_excptn);}
std::map<std::string, int>::iterator i1 = kgrams.find(kgram);

                    if (i1 != kgrams.end())
                       return i1-> second;
                else
                        return 0;
            }

```

```
76.  int MModel::freq(std::string kgram, char c) {
77.      int frequency = 0;
78.
79.          if (order == 0) {
```

```
                    } return frequency;
                    }  84.
85.  if ((unsigned)order != kgram.length()) {  86.      throw
std::runtime_error(frequency_excptn);}
 87.
88.                 std::string follow_c(1, c);
89.                 std::string kplusone_c = kgram + follow_c;
90.                 std::map<std::string, int>::iterator i2 = kgrams.find(kpluson e_c);
91.                 if (i2 != kgrams.end()) {
92.                 return i2-> second;
93.                 } else {
94.                 return 0;
95.                 }  96.          }
 97.
98.                 char MModel::kRand(std::string kgram) {
99.                 std::vector<char> random_distribution;
100.                int random_index;
101.                int kgram_frequency;
102.                if (freq(kgram) == 0) {
103.                throw std::runtime_error(kgram_exceptn);
104.                } else {
105.                kgram_frequency = freq(kgram);
106.                }  107.
108.                    for (unsigned int i = 0; i < alpha.length(); i++) {
109.                    char kplusone_current = alpha.at(i);
110.                    int follow = freq(kgram, kplusone_current);
111.                    for (int j = 0; j < follow; j++) {
112.                    random_distribution.push_back(kplusone_current);
113.                    }
114.                    }
115.                    random_index = rand() % random_distribution.size();  // NOLIN T
116.                    return random_distribution.at(random_index);
```

```
80.                 for (unsigned int i = 0; i < text1.length(); i++) {
```

```
81.                     if (text1.at(i) == c) frequency++;
82.
83.
```

```
117.                    }  118.
119.          std::string MModel::generate(std::string kgram, int t) {
120.          std::string txt_generation = kgram;
 121.
122.          if (order == 0 && t == 0)
123.          return text1;   124.
125.                    while (txt_generation.length() < (unsigned)t) {
126.                    char c = kRand(kgram);
127.                    std::string str(1, c);
128.                    txt_generation.append(str);
129.                    if (order > 0) {
130.                    kgram = kgram.substr(1);
131.                    } else {kgram = kgram;}
```

```
                        std::string next_gram = kgram + str;
                    kgram = next_gram;
                }
                return txt_generation;
            }

                std::ostream& operator << (std::ostream& os, MModel& mm) {
                                                    std::cout << std::endl << "Order K of Markov Model = " << mm.
    order;
                std::cout << std::endl;
    std::cout << "Available alphabet = " << mm.alpha << std::endl ;

                                std::map<std::string, int>::iterator it;
                                for (it = mm.kgrams.begin(); it != mm.kgrams.end(); ++it) {
                                    if (it->first.length() == (unsigned)mm.order) {
                                    std::cout << std::endl;
                                        std::cout << "kgram:      ";
            std::cout << it->first << "   |  frequency: " << it>second;

                                        std::cout << std::endl;
                                } else {
                                            std::cout << "           ";
            std::cout << it->first << "  ----------->  " << it>second;

                                        std::cout << std::endl;
                            }
                    }

                  return os;
    }
```

Boosttest.cpp:

```
/*Copyright 2020 Shreya Patil*/
#include <boost/test/unit test.hpp>

#include <string>

#include <iostream>

#include <exception>
#include <stdexcept>

#include "/home/shreya/Desktop/ps6/MModel.h"

#define BOOST TEST DYN LINK
#define BOOST_TEST_MODULE Main



using namespace std; // NOLINT

BOOST_AUTO_TEST_CASE(order0) {
// normal constructor
BOOST_REQUIRE_NO_THROW(MModel("gagggagaggcgagaaa", 0));

MModel mm("gagggagaggcgagaaa", 0);

BOOST_REQUIRE(mm.kOrder() == 0);
BOOST_REQUIRE(mm.freq("") == 17);  // length of input in constructor
BOOST_REQUIRE_THROW(mm.freq("x"), std::runtime_error);

BOOST_REQUIRE(mm.freq("", 'g') == 9);
BOOST_REQUIRE(mm.freq("", 'a') == 7);
BOOST_REQUIRE(mm.freq("", 'c') == 1);
BOOST_REQUIRE(mm.freq("", 'x') == 0);
}

BOOST_AUTO_TEST_CASE(order1) {
// normal constructor
BOOST_REQUIRE_NO_THROW(MModel("gagggagaggcgagaaa", 1));

MModel mm("gagggagaggcgagaaa", 1);

BOOST_REQUIRE(mm.kOrder() == 1);
BOOST_REQUIRE_THROW(mm.freq(""), std::runtime_error);
BOOST_REQUIRE_THROW(mm.freq("xx"), std::runtime_error);

BOOST_REQUIRE(mm.freq("a") == 7);
BOOST_REQUIRE(mm.freq("g") == 9);
BOOST_REQUIRE(mm.freq("c") == 1);

BOOST_REQUIRE(mm.freq("a", 'a') == 2);
BOOST_REQUIRE(mm.freq("a", 'c') == 0);
BOOST_REQUIRE(mm.freq("a", 'g') == 5);
```

```
80.    BOOST_REQUIRE_NO_THROW(mm.freq("xx"));
81.    BOOST_REQUIRE_THROW(mm.freq("", 'g'), std::runtime_error);  // wrong length
82.    BOOST_REQUIRE_THROW(mm.freq("x", 'g'), std::runtime_error);  // wrong length   83.
       BOOST_REQUIRE_THROW(mm.freq("xxx", 'g'), std::runtime_error);  // wrong length
84.
85.
86.    BOOST_REQUIRE(mm.freq("aa") == 2);
87.    BOOST_REQUIRE(mm.freq("aa", 'a') == 1);   88.
       BOOST_REQUIRE(mm.freq("aa", 'c') == 0);
89.  BOOST_REQUIRE(mm.freq("aa", 'g') == 1);
90.
91.    BOOST_REQUIRE(mm.freq("ag") == 5);
92.    BOOST_REQUIRE(mm.freq("ag", 'a') == 3);   93.
       BOOST_REQUIRE(mm.freq("ag", 'c') == 0);
94.  BOOST_REQUIRE(mm.freq("ag", 'g') == 2);
95.
96.    BOOST_REQUIRE(mm.freq("cg") == 1);
97.    BOOST_REQUIRE(mm.freq("cg", 'a') == 1);   98.
       BOOST_REQUIRE(mm.freq("cg", 'c') == 0);
99.  BOOST_REQUIRE(mm.freq("cg", 'g') == 0);
100.
101.           BOOST_REQUIRE(mm.freq("ga") == 5);
102.           BOOST_REQUIRE(mm.freq("ga", 'a') == 1);
               BOOST_REQUIRE(mm.freq("ga", 'c') == 0);
103.
104.           BOOST_REQUIRE(mm.freq("ga", 'g') == 4);
105.
106.           BOOST_REQUIRE(mm.freq("gc") == 1);
107.           BOOST_REQUIRE(mm.freq("gc", 'a') == 0);
108.           BOOST_REQUIRE(mm.freq("gc", 'c') == 0);
109.           BOOST_REQUIRE(mm.freq("gc", 'g') == 1);
```

```
53.           BOOST_REQUIRE(mm.freq("c",  'a')  ==  0);          54.
BOOST_REQUIRE(mm.freq("c", 'c') == 0);
```

```
  BOOST_REQUIRE(mm.freq('c', 'g') == 1);

  BOOST_REQUIRE(mm.freq('g', 'a') == 5);
BOOST_REQUIRE(mm.freq('g', 'c') == 1);
  BOOST_REQUIRE(mm.freq('g', 'g') == 3);

  BOOST_REQUIRE_NO_THROW(mm.kRand('a'));
BOOST_REQUIRE_NO_THROW(mm.kRand('c'));
  BOOST_REQUIRE_NO_THROW(mm.kRand('g'));

  BOOST_REQUIRE_THROW(mm.kRand('x'), std::runtime_error);

  BOOST_REQUIRE_THROW(mm.kRand('xx'), std::runtime_error);
}
  BOOST_AUTO_TEST_CASE(order2) {
  // normal constructor
  BOOST_REQUIRE_NO_THROW(MModel('gagggagaggcgagaaa', 2));

MModel mm('gagggagaggcgagaaa', 2);

BOOST_REQUIRE(mm.kOrder() == 2);

  BOOST_REQUIRE_THROW(mm.freq(''), std::runtime_error);
  BOOST_REQUIRE_THROW(mm.freq('x'), std::runtime_error);


          BOOST_REQUIRE(mm.freq('gg') == 3);
        BOOST_REQUIRE(mm.freq('gg', 'a') == 1);
      BOOST_REQUIRE(mm.freq('gg', 'c')
                                == 1);
    BOOST_REQUIRE(mm.freq('gg', 'g') == 1);
    }
```