COMP IV: Project Portfolio

Brent Garey

FALL 2020

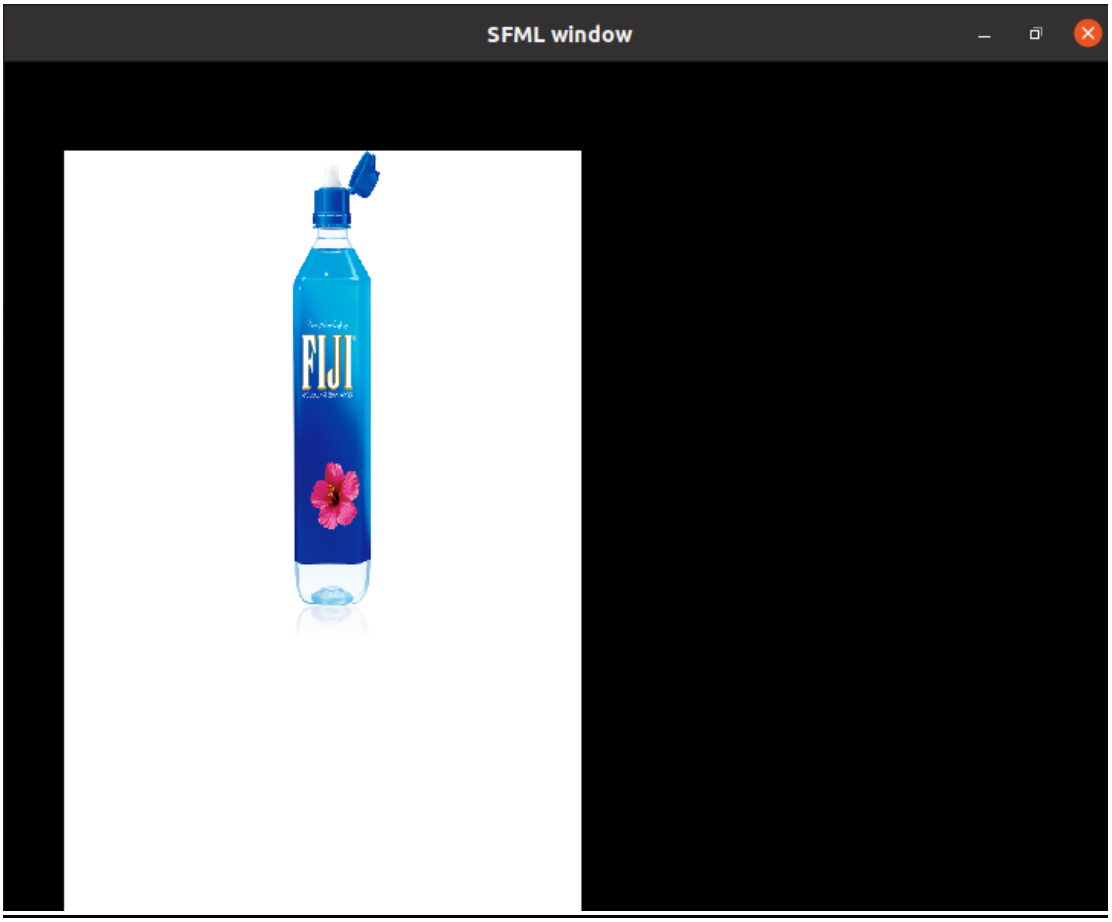
COMPUTING.2040

UMASS LOWELL

Table of Contents:

PS0: Hello World with SFML

Objective: Utilize SFML to demonstrate familiarity in controlling a sprite. The following sprite would move in the 4 directions corresponding to the arrow keys and rotate clockwise/counter clockwise.



Key algorithms, data structures, or OO designs that were central to this assignment:

SFML

What was learned:

SFML variables were created, specifically sf::Texture, sf::Sprite, and sf::RenderWindow.

I learned that sf::Sprite could be loaded off a sf::Texture. You can then utilize sf::Event in order to determine how to manipulate the sprite.

main.cpp Wed Sep 09 21:45:23 2020 1

```
 1: //Brent Garey ps0 Assignment
 2:
 3: #include <SFML/Graphics.hpp>
 4:
 5: int main()
 6: {
 7:         //position variables
 8:         int x = 100;
 9:         int y = 100;
10:         //Create the main window
11:         sf::RenderWindow window(sf::VideoMode(2400, 1800), "SFML window");
12:         //load closed water texture
13:         sf::Texture texture;
14:         texture.loadFromFile("sprite.png");
15:         if(!texture.loadFromFile("sprite.png"))
16:                 return EXIT_FAILURE;
17:         //create a sprite
18:         sf::Sprite sprite;
19:         sprite.setTexture(texture);
20:         sprite.setPosition(x,y);
21:
22:         //load open water texture
23:         sf::Texture texture1;
24:         texture.loadFromFile("sprite1.png");
25:         if(!texture.loadFromFile("sprite1.png"))
26:                 return EXIT_FAILURE;
27:
28:         //load some text
29:         sf::Font font;
30:         if(!font.loadFromFile("arial.ttf"))
31:                 return EXIT_FAILURE;
32:         sf::Text text("You are now hydrated!", font, 500);
33:
34:         while(window.isOpen())
35:         {
36:                 //process events
37:                 sf::Event event;
38:                 while(window.pollEvent(event))
39:                 {
40:                         //event = close
41:                         if(event.type == sf::Event::Closed)
42:                                 window.close();
43:                         //event move left
44:                         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
45:                         {
46:                                 x=x-75;
47:                                 sprite.setPosition(x,y);
48:                         }
49:
50:                         //event move up
51:                         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
52:                         {
53:                                 y=y-75;
54:                                 sprite.setPosition(x,y);
55:                         }
56:
57:                         //event move down
58:                         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
59:                         {
60:                                 y=y+75;
61:                                 sprite.setPosition(x,y);
```

```
62:                             }
63:
64:                             //event move right
65:                             if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
66:                             {
67:                                     x=x+75;
68:                                     sprite.setPosition(x,y);
69:                             }
70:
71:                             //event space bar drinks the water
72:                             if(sf::Keyboard::isKeyPressed(sf::Keyboard::Space))
73:                             {
74:                                     sprite.setTexture(texture1);
75:                                     window.draw(text);
76:                             }
77:
78:                             //event q and e rotates the water bottle
79:                             if(sf::Keyboard::isKeyPressed(sf::Keyboard::Q))
80:                                     sprite.rotate(-45);
81:
82:                             if(sf::Keyboard::isKeyPressed(sf::Keyboard::E))
83:                                     sprite.rotate(45);
84:                     }
85:
86:                     //clear the screen
87:                     window.clear();
88:                     //draw the sprite
89:                     window.draw(sprite);
90:                     //update the window
91:                     window.display();
92:             }
93:
94:             return EXIT_SUCCESS;
95: }
```

PS1: Linear Feedback Shift Register

Part A- Implement a 16bit LFSR

Part B- PhotoMagic Transformation

Objective:

Encrypt a .png file using a 16bit LFSR as a seed. The output will change the pixels of the image so that it cannot be viewed / restored unless you put in the same LFSR that was used as a seed to encrypt it.

Key algorithms, data structures, or OO designs that were central to this assignment:

Utilizing XOR allowed us to be able to revert the image back to its original form.

Understanding SFML and how it stores images and pixels allowed me to save the RGB value of each pixel and XOR it with the LFSR.

Correctly implementing the LFSR was critical since errors would lead to the "password" not doing it's intended functionality.

What was learned:

I learned how to access the pixels of an image stored in SFML. Manipulating the integer of RGB in the image with our LFSR allowed us to encode it (my first implementation of encoding an image)

```
#include <iostream>
#include <string>
#include <stdlib.h>
#include "FibLFSR.hpp"

//using declarations
using std::ostream;
using std::cout;
using std::endl;
using std::string;

FibLFSR::FibLFSR(string seed) {
        flfsr = seed;
}
//constructor to create LFSR with
//the given initial seed

FibLFSR::FibLFSR(){
        flfsr = '0';
}
FibLFSR::~FibLFSR() { cout << "FibLFSR destructor called" << endl; }

int FibLFSR::step() {

                char* s;
                s = &(flfsr.at(1));
                //store the first bit that will be shifted
                char first_bit = flfsr.at(0);

                //store each tap position
                char tap1 = flfsr.at(2);
                char tap2 = flfsr.at(3);
                char tap3 = flfsr.at(5);

                //convert them to int to compute xor
                int First_bit = ((int) first_bit) - 48;//atoi(&first_bit);
                int Tap1 = ((int) tap1) - 48;//atoi(&tap1);
                int Tap2 = ((int) tap2) - 48;//atoi(&tap2);
                int Tap3 = ((int) tap3) - 48;//atoi(&tap3);

                //compute the new bit
                int first_xor = First_bit ^ Tap1;
                int second_xor = first_xor ^ Tap2;
                int third_xor = second_xor ^ Tap3;

                //shift the bits
                flfsr = s;

                //convert from int to string
                string last_bit;
                if (third_xor == 1) {
                        last_bit = "1";
                        }
                else {
                        last_bit = "0";
                        }
                //append the last bit
                flfsr = flfsr + last_bit; //to_string(third_xor);

                return third_xor;
        }
```

```
62:         //simulate one step and return
63:         //the new bit as 0 or 1
64:
65:         int FibLFSR::generate(int k) {
66:                 int value = 0;
67:                 for (;k > 0; k--)
68:                 {
69:                         value = (value * 2) + step();
70:                         }
71:                 return value;
72:                 }
73:         //simulate k steps and return k-bit integer
74:
75:         string FibLFSR::getFLFSR() const { return flfsr; }
76:         //accessor function returns the register value
77:         //overload << in order to display current register value
78:         ostream& operator<< (ostream& out, const FibLFSR& f) {
79:         out << f.getFLFSR();
80:         return out;
81: }
82:
83:
```

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

void transform(sf::Image& image, FibLFSR* password);

int main(int argc, char* argv[])
{
 //lets open the image and store it
 sf::Image image;
 if(!image.loadFromFile("input-file.png"))
        return -1;
 FibLFSR password(argv[3]);

 transform(image, &password);

// sf::Color p;

sf::Vector2u size = image.getSize();
sf::RenderWindow window(sf::VideoMode(size.x, size.y), "Encryption / Decrypt
ion");

// for(int x = 0; x < size.x; x++){
//      for(int y = 0; y < size.y; y++){
//              p = image.getPixel(x, y);
//              p.r = 255 - p.r;
//              p.g = 255 - p.g;
//              p.b = 255 - p.b;
//              image.setPixel(x,y,p);
//              }
//      }

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

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

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

 if(!image.saveToFile("input-file.png"))
        return -1;
 if(!image.saveToFile("output-file.png"))
        return -1;

 return 0;
}
```

```
59: void transform(sf::Image& image, FibLFSR* password)
60: {
```



```
61:
62:         sf::Color p;
63:         sf::Vector2u size = image.getSize();
64:         for(int x = 0; x < size.x; x++){
65:                 for(int y = 0; y < size.y; y++){
66:                         p = image.getPixel(x,y);
67:                         p.r =  (p.r) ^ (password->generate(8)); //255 - p.r;
68:                         p.g =  (p.g) ^ (password->generate(8)); //255 - p.g;
69:                         p.b =  (p.b) ^ (password->generate(8)); //255 - p.b;
70:                         image.setPixel(x,y,p);
71:                         }
72:                 }
73:
74: }
```

PS2: N-Body Simulation

Pt a- CelestialBody.cpp & Universe.cpp

Pt b- NBody.cpp

Objective: Implement a physics simulation of planets in our solar system by creating classes for each CelestialBody. Create a Universe class that is full of each CelestialBody in our solar system that is affected by the physics

Key algorithms, data structures, or OO designs that were central to this assignment:

Object Oriented Design was crucial towards implementing the CelestialBody & Universe classes. Specifically, My Universe holds a vector of CelestialBody* and is able to edit each CelestialBody (after adding them to the vector).

What was learned:

This assignment solidified my familiarity with object orientated programming by creating two classes and constantly having to check and update their date members.

Specifically, the Universe class will constantly update the position of each CelestialBody due to the changing physics in NBody.

```
    1: #include "CelestialBody.cpp"
    2: #include <string>
    3: #include <vector>
    4: #include <math.h>
    5: using std::endl;
    6: using std::ostream;
    7: using std::istream;
    8: using std::string;
    9: using std::vector;
   10: using std::cin;
   11: using std::cout;
   12:
   13: class Universe
   14: {
   15: public:
   16: //default constructor
   17: Universe(sf::RenderWindow& window)
   18: {
   19:  cin >> numPlanets;
   20:  cin >> radius;
   21:  //for each planet..
   22:  for(; numPlanets > 0; numPlanets--)
   23:  {
   24:   cin >> xpos;
   25:   cin >> ypos;
   26:   cin >> xvelo;
   27:   cin >> yvelo;
   28:   cin >> mass;
   29:   cin >> filename;
   30:   size = window.getSize();
   31:
   32:   //create a new planet with the info got
   33:   CelestialBody* planet = new CelestialBody(xpos,ypos,xvelo,yvelo,mass,filen
ame, radius, size);
   34:   planets.push_back(planet);
   35:  }
   36: }
   37:
   38: virtual void draw(sf::RenderWindow& window)
   39: {
   40:  for(vector<CelestialBody*>::iterator it = planets.begin(); it!= planets.end
();it++)
   41:   {
   42:         (*it)->draw(window);
   43:   }
   44: }
   45:
   46: void step(double seconds){
   47:
   48: double deltaX;
   49: double deltaY;
   50: double deltaR;
   51: double force;
   52: double forceY;
   53: double forceX;
   54: double netForce;
   55: double xAccel;
   56: double yAccel;
   57: double xVeloNew;
   58: double yVeloNew;
   59: double xPosNew;
```

```
   60: double yPosNew;
   61:
   62:  for(int i = 0; planets[i+1] != NULL; i++)
   63: {
   64: //get the difference in positions from i to the next one
   65: deltaX = planets[i]->get_xPosition() - planets[i+1]->get_xPosition();
   66: deltaY = planets[i]->get_yPosition() - planets[i+1]->get_yPosition();
   67: deltaR = sqrt((deltaX * deltaX)+(deltaY * deltaY));
   68:
   69: //get and set netforce
   70: force = (Grav_const * (planets[i]->get_mass()) * (planets[i+1]->get_mass()))
 / (deltaR * deltaR);
   71: forceY = (force * deltaY) / (deltaR);
   72: forceX = (force * deltaX) / (deltaR);
   73: netForce = forceX + forceY;
   74: planets[i]->set_netForce(netForce);
   75: planets[i+1]->set_netForce(netForce);
   76:
   77: //get and set acceleration
   78: xAccel = forceX / planets[i]->get_mass();
   79: yAccel = forceY / planets[i]->get_mass();
   80: planets[i]->set_xAcceleration(xAccel);
   81: planets[i]->set_yAcceleration(yAccel);
   82:
   83: //calculate new velocity using the acceleration and time
   84: xVeloNew = planets[i]->get_xVelocity() + (seconds * xAccel);
   85: yVeloNew = planets[i]->get_yVelocity() + (seconds * yAccel);
   86: planets[i]->set_xVelocity(xVeloNew);
   87: planets[i]->set_yVelocity(yVeloNew);
   88:
   89: //calculate new position
   90: xPosNew = planets[i]->get_xPosition() + (seconds * planets[i]->get_xVelocity
());
   91: yPosNew = planets[i]->get_yPosition() + (seconds * planets[i]->get_yVelocity
());
   92: planets[i]->set_xPosition(xPosNew);
   93: planets[i]->set_yPosition(yPosNew);
   94: }
   95:
   96: }
   97:
   98: //friend istream& operator>>(istream& in, Universe universe){
   99: //in >> universe.numPlanets >> universe.radius;
  100: //in >> universe.xpos >> universe.ypos;
  101: //in >> universe.xvelo >> universe.yvelo;
  102: //in >> universe.mass >> universe.filename;
  103: //return in;
  104: //}
  105:
  106: double get_G(void){
  107: return Grav_const;}
  108: friend ostream& operator<<(ostream& out, Universe universe){
  109:  for(vector<CelestialBody*>::iterator it = universe.planets.begin(); it!= un
iverse.planets.end(); it++)
  110:  {
  111:         out << **it << endl;
  112:  }
  113: return out;
  114: }
  115: private:
  116: int numPlanets;
```

```
117: double radius;
118: sf::Vector2u size;
119: //make a vector of celestialbodies
120: vector<CelestialBody*> planets;
121:
122: //temp variables to hold scanned values
123: double xpos,ypos,xvelo,yvelo,mass;
124: string filename;
125:
126: //values for forces
127: const double Grav_const = (6.67*pow(10,-11));
128: };
```

```
    1: #include <iostream>
    2: #include <stdlib.h>
    3: #include <SFML/System.hpp>
    4: #include <SFML/Graphics.hpp>
    5: #include <SFML/Window.hpp>
    6: #include <string.h>
    7:
    8: using std::ostream;
    9: using std::endl;
   10: using std::istream;
   11: using std::string;
   12: using sf::Vector2f;
   13:
   14: //implement a celestialbody class
   15: class CelestialBody{
   16: public:
   17: //value constructor
   18: CelestialBody(double xPos, double yPos, double xVelo,
   19: double yVelo, double Mass, string fileName, double universeradius, sf::Vecto
r2u Size){
   20: xPosition = xPos;
   21: yPosition = yPos;
   22: xVelocity = xVelo;
   23: yVelocity = yVelo;
   24: mass = Mass;
   25: filename = fileName;
   26: universeRadius = universeradius;
   27: size = Size;
   28: //load image, texture and sprite
   29: image.loadFromFile(filename);
   30: texture.loadFromImage(image);
   31: sprite.setTexture(texture);
   32:
   33: //setPosition();
   34:
   35: }
   36:
   37: //setter for netForce
   38: void set_netForce(double force){
   39: netForce = force;}
   40:
   41: //empty constructor
   42: CelestialBody(void){
   43: xPosition = 0;
   44: yPosition = 0;
   45: xVelocity = 0;
   46: yVelocity = 0;
   47: mass = 0;
   48: filename = "empty";
   49: }
   50: //getter functions to return private members
   51: double get_xPosition(void){
   52: return xPosition;}
   53:
   54: double get_yPosition(void){
   55: return yPosition;}
   56:
   57: double get_xVelocity(void){
   58: return xVelocity;}
   59:
   60: double get_yVelocity(void){
```

```
   61: return yVelocity;}
   62:
   63: double get_mass(void){
   64: return mass;}
   65:
   66: double get_netForce(void){
   67: return netForce;}
   68:
   69: double get_xAcceleration(void){
   70: return xAcceleration;}
   71:
   72: double get_yAcceleration(void){
   73: return yAcceleration;}
   74: void setPosition(void){
   75: xPosition = ((xPosition / universeRadius) * (size.x / 2)) + (size.x / 2);
   76: yPosition = ((yPosition / universeRadius) * (size.y / 2)) + (size.y / 2);
   77: sprite.setPosition(xPosition, yPosition);
   78: }
   79:
   80: //mutator function to modify velocitys
   81: void set_xPosition(double position){
   82: xPosition = position;}
   83: void set_yPosition(double position){
   84: yPosition = position;}
   85: void set_xVelocity(double velocity){
   86: yVelocity = velocity;
   87: }
   88: void set_yVelocity(double velocity){
   89: xVelocity = velocity;
   90: }
   91: void set_xAcceleration(double accel){
   92: xAcceleration = accel;}
   93: void set_yAcceleration(double accel){
   94: yAcceleration = accel;}
   95: //calculate the net force (Fx and Fy) at current time t acting on
   96: //this celestial body object
   97: //void set_netForce(){
   98:
   99: //moves object due to velocity for given time
  100: void step(double seconds);
  101: //overload insertion operator to read in values
  102: friend istream& operator>>(istream& in, CelestialBody& celestialBody){
  103: in >> celestialBody.xPosition >> celestialBody.yPosition;
  104: in >> celestialBody.xVelocity >> celestialBody.yVelocity;
  105: in >> celestialBody.mass;
  106: in >> celestialBody.filename;
  107:
  108: if(!celestialBody.image.loadFromFile(celestialBody.filename))
  109:         return in;
  110: celestialBody.texture.loadFromImage(celestialBody.image);
  111: celestialBody.sprite.setTexture(celestialBody.texture);
  112: celestialBody.sprite.setPosition(celestialBody.xPosition,celestialBody.yPosi
tion);
  113:
  114: return in;
  115: }
  116:
  117: friend ostream& operator<<(ostream& out, CelestialBody& celestialBody){
  118: out << "Xposition: " << celestialBody.xPosition << endl;
  119: out << "Yposition: " << celestialBody.yPosition << endl;
  120: out << "xvelocity: " << celestialBody.xVelocity << endl;
```

```
121: out << "yvelocity: " << celestialBody.yVelocity << endl;
122: out << "mass: " << celestialBody.mass << endl;
123: return out;
124: }
125:
126: virtual void draw(sf::RenderWindow& window){
127:  window.draw(sprite);
128: }
129:
130: private:
131: double xPosition;
132: double yPosition;
133: double xVelocity;
134: double yVelocity;
135: double mass;
136: string filename;
137: sf::Image image;
138: sf::Texture texture;
139: sf::Sprite sprite;
140: double universeRadius;
141: sf::Vector2u size;
142: double netForce;
143: double xAcceleration;
144: double yAcceleration;
145: };
```

PS3: Synthesizing a Plucked String Sound

Part a-CircularBuffer
Part b-KSGuitarSim

Objective: Create a program that will generate Guitar string sounds based off input from the keyboard. You will need a CircularBuffer data structure to implement KSGuitarSim.cpp that will be used to open the corresponding sounds, hold the frequencies, and play the sound to the user.

Key algorithms, data structures, or OO designs that were central to this assignment:

Part a involved creating a custom CircularBuffer class that acted as a vector container but data was held in a ring buffer, where accessing data from one side and hopping from that will lead you to the other side of the buffer.

Part b involved utilizing memory to store the correct frequencies.

What was learned:

I learned that SFML is immensely helpful towards allowing us to load and store files into vectors. This was extremely helpful when it came to playing the "sound" of the corresponding frequency.

PS4: DNA Sequence Alignment

Objective: Write a program that will calculate the penalty for changing a DNA sequence into another.

Key algorithms, data structures, or OO designs that were central to this assignment:

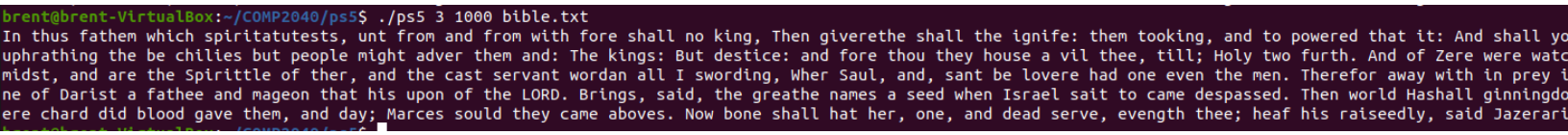
Allocating memory that allowed the user to treat it as a matrix. I attempted to create a matrix class, however I had trouble implementing and managing the memory so I was unable to calculate the penalty.

What was learned:

I learned how to store a matrix and access it via matrix[x][y]. This allowed me to allocate the memory based off the given [x] and [y] to determine distance from a given DNA sequence.

PS5: Markov Model of Natural Language

Objective: Implement a Markov Model structure that will keep track of kGrams, the subsequent character, and both of their frequencies. This will be used to calculate probability for each character whenever a kGram is found in order to give a random(yet accurate) sample of probable predictions (of a given string).



Key algorithms, data structures, or OO designs that were central to this assignment:

I created a map <string, int> that would store the string as a key for each map. It would scan a string of size k. The substring would be used to check the map if a key exists. If the substring representing the kGram was not found, it would initialize the map with a frequency of 1 to indicate this is the first time the kGram was found:

```
if (kGramFreq.find(kGram) == kGramFreq.end()) {
  // set the freq to 1(first time seeing)
  kGramFreq[kGram] = 1;
```

If the find function found the string, it would increment the integer representing the frequencies by 1:

```
else {  // if it gets this far.. its in the map
  kGramFreq[kGram] = ++kGramFreq[kGram];
```

For both cases, I stored the subsequent characters as a separate map to link them to the kgram (to find probability of each character occurring)

```
theoretical_char = word.substr(max + 1, 1);

kG[kGram].insert(std::make_pair(theoretical_char[0], 1));
```

What was learned:

Learning how to manipulate maps to store the kGrams and the alphabet (and their frequencies).

I specifically learned that the .find(kGram) will return .end() upon failing to find the map.

I then created a map to hold a <string, map <char, int> > to link the frequency of each subsequent character to each string. This implementation got me familiar with how to access and use the member functions associated with maps.

PS6: Kronos – Intro to Regular Expression

Objective:

Read in a .log file to evaluate whether a server started. If the server was successfully started, write into a .rpt file and report its duration time. If the server failed, report the failure in the .rpt file.

Utilize Regular Expressions (regex) to compare the input to evaluate whether the input matches the criteria of the regex variable. If it does, create a boost::datetime variable to calculate the uptime of the server.

112 (log.c.166) server started: 2013-10-02 18:42:38 Succeeded and ran for 165000ms

855 (log.c.166) server started: 2013-10-03 12:23:21 Succeeded and ran for 174000ms

1568 (log.c.166) server started: 2013-10-04 16:20:03 Succeeded and ran for 183000ms

31956 (log.c.166) server started: 2013-12-03 16:21:13 Succeeded and ran for 175000ms

33032 (log.c.166) server started: 2013-12-04 21:50:27 Succeeded and ran for 150000ms

33390 (log.c.166) server started: 2013-12-04 21:58:45 Succeeded and ran for 149000ms

33920 (log.c.166) server started: 2013-12-04 22:21:03 Succeeded and ran for 148000ms

45538 (log.c.166) server started: 2013-12-05 13:34:25 Succeeded and ran for 150000ms

45858 (log.c.166) server started: 2013-12-05 14:12:25 Succeeded and ran for 148000ms

46353 (log.c.166) server started: 2013-12-05 15:39:02 Succeeded and ran for 147000ms

46600 (log.c.166) server started: 2013-12-05 20:20:24 Succeeded and ran for 150000ms

47035 (log.c.166) server started: 2013-12-10 13:20:43 Succeeded and ran for 149000ms

47991 (log.c.166) server started: 2013-12-10 19:40:58 Succeeded and ran for 177000ms

48341 (log.c.166) server started: 2013-12-11 14:09:11 Succeeded and ran for 150000ms

48636 (log.c.166) server started: 2013-12-11 14:17:49 Succeeded and ran for 177000ms

Key algorithms, data structures, or OO designs that were central to this assignment:

Specifically, a regex variable was initialized to match with the criteria of the starting string:

25 String `" (log.c.166) server started: "`= regex start`"[(]log.c.[1 - 6][1 - 6][1 - 6][)][\\s]server[\\s]started[\\s]".`

This regex was used as an argument for regex_search:

```
42 regex_search(getline, smStart, start);
```

Which allowed me to compare the regex with a getline. If the two matched (meaning the getline evaluated as fitting the criteria of the regex) then we would store a substring of the getline to isolate and store the time in a string via

```
55 startTime = boost::posix_time::time_from_string(starttime);
```

Now with the time stored and the start message received, a regex that fit the criteria of the end string was created and subsequently matched with getline:

```
26 end = "[\\d][\\d][\\d][\\d]-[\\d][\\d]-
[\\d][\\d][\\s][\\d][\\d]:[\\d][\\d]:[\\d][\\d].[\\d][\\d][\\d]:INFO:oejs.AbstractConnect
or:Started[\\s]SelectChannelConnector@[\\d].[\\d].[\\d].[\\d]:[\\d][\\d][\\d][\\d]";

60 std::regex_search(getline, smEnd, end);
```

We implemented checks by checking if another potential start string was found before the end string, or if there was no end string found. This would indicate the server failed.

What was learned:

PS6 was my first implementation of Regular Expression (regex).

Regex manipulation was used to evaluate criteria of a start and end string.

Boost::datetime then allowed me to manipulate the string into a substring that stored a start time or end time.