2021

Fall 2021 Portfolio : COMP 2040

HAWKINS, SANDRA V

Preface:

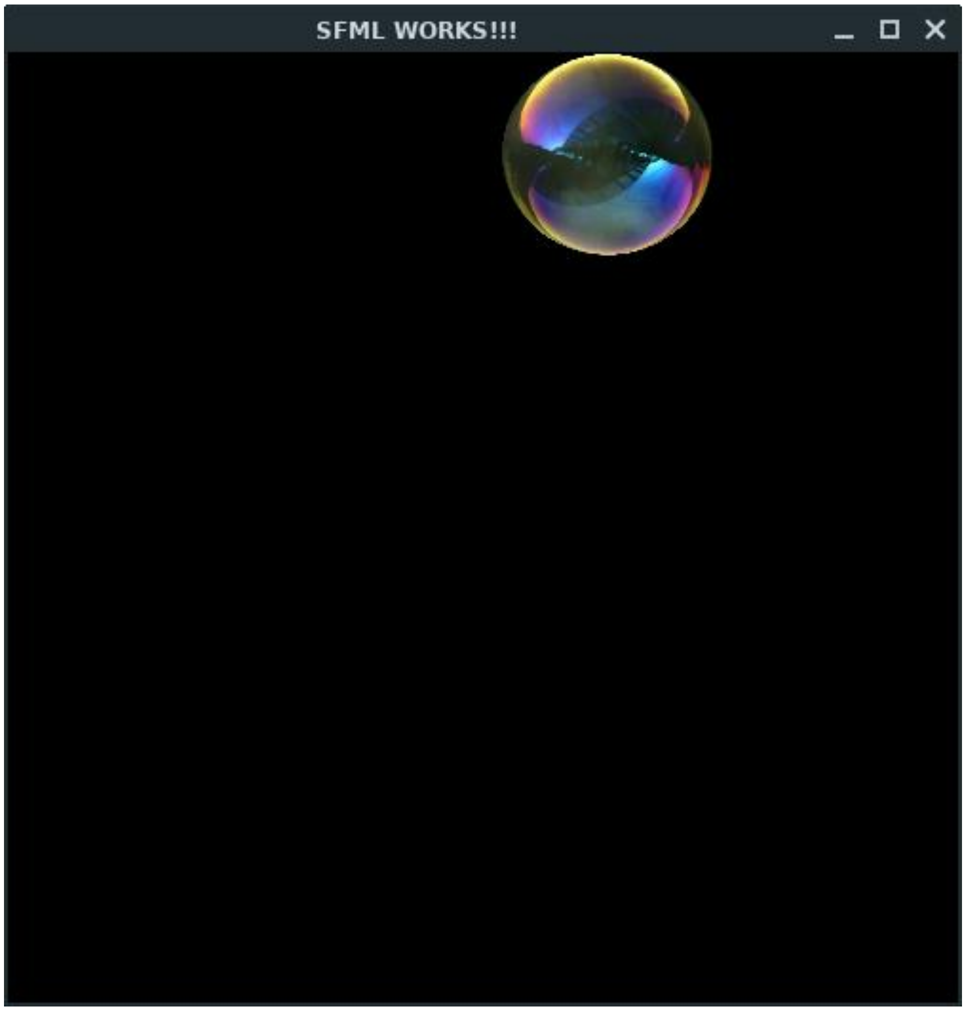
This is a portfolio consisting of the assignments I have completed for one of my programming classes in the Fall 2021 semester during my junior year at University of Massachusetts Lowell. The course being “Computing 4” (COMP 2040 by its class code), where advanced C++ is taught. There were eleven total assignments, numbered from zero through seven where few were split into two parts: A and B respectively. All of the assignments are done in C++, where I am the sole writer of them, with the exception of one (I was a cowriter). As of December 31st, 2021, what is shown here is my current ability and competency in C++ programming.

Sandra Hawkins Portfolio: COMP2040 Fall 2021

Contents:

PS0: Getting to know SFML

This assignment was all about setting up SFML and really just testing it out, see if it worked, see how it worked. The goal of the assignment was to modify the starter code in order to make the SFML object a sprite. As well as make it move in the window it is displayed on. My sprite moved in 2 ways: either by the arrow keys, or by WASD keys. It also had an additonal plus that it always stayed within the window it occupied (so it had always stayed 'on-screen' so to speak). The sprite was able to be any image, but I chose for it be a bubble. In the assignment I learned how to make a sprite in SFML, which I did not know before, though I had experience with SFML in a prior class, so it was not entirely foreign to me. I also learned how to use a new text editor, nano, on a virtual machine. What challenged me in this assignment was actually the Makefile (of all things). I had to ask for help with that, as I had completely forgotten how to do one.



SFML window showing the sprite, indicating that SFML does, indeed, work.

Makefile   Sun Sep 05 02:16:54 2021   1

```
compile: ./main.cpp
	g++ -Wall -Werror -pedantic -c ./main.cpp
	g++ main.o -o app -lsfml-graphics -lsfml-window -lsfml-system

run:
	./app
```

```cpp
/***
main.cpp -  PS0 (SFML installation, texture/movement experimentation)

Date: 09.05.2021

Created by: Sandra Hawkins


*/




#include <iostream>
#include <SFML/Graphics.hpp>
using namespace std;
using namespace sf;

const int WINDOW_X = 500;
const int WINDOW_Y = 500;
const int DISTANCE = 10;
const int SIZE = 100;

void move_bubble(Sprite& bubble);
int main() {
        //cout << "Hello!" << endl;
        //testing
        RenderWindow window(VideoMode(WINDOW_X, WINDOW_Y), "SFML WORKS!!!");
        /*CircleShape shape(100.f);
        shape.setFillColor(Color::Magenta);
        shape.setPosition(Vector2f(150, 150));*/

        Texture texture;
        if (!texture.loadFromFile("sprite.jpg")) {return EXIT_FAILURE;}
        Sprite sprite(texture);
        sprite.setPosition(Vector2f(150,150));
        sprite.scale(Vector2f(0.125, 0.125)); //shear both dimensions to scale

        while(window.isOpen()) {
                Event event;
                while (window.pollEvent(event)) {
                        if (event.type == Event::Closed) {window.close();}
                        //move if there has been a keystroke (arrowkeys/wasd)
                        //DO ME: function for 4-way movement
                        if (event.type == Event::KeyPressed) {
                                move_bubble(sprite);}
                }

                window.clear();
                window.draw(sprite);
                window.display();

        }

        return 0;
}
bool validate_position(const Sprite& bubble) {
        bool valid =  true;
        Vector2f pos = bubble.getPosition();

        if (pos.x < 0 || pos.x > (WINDOW_X- SIZE) ||
        pos.y < 0 || pos.y > (WINDOW_Y- SIZE)) {
                valid = false;
        }

        return valid;
}
void realign(Sprite& bubble) {
        Vector2f pos = bubble.getPosition();
        if (pos.x < 0) { bubble.setPosition(0,  pos.y); }
        if (pos.x > WINDOW_X - 100) { bubble.setPosition(WINDOW_X - SIZE, pos.y);}
        if (pos.y < 0) { bubble.setPosition(pos.x, 0);}
        if (pos.y > WINDOW_Y -100) { bubble.setPosition(pos.x, WINDOW_Y - SIZE);}
}
void move_bubble(Sprite& bubble) {
        //move via arrow keys OR wasd

        //up
        if (Keyboard::isKeyPressed(Keyboard::Key::Up) ||
        Keyboard::isKeyPressed(Keyboard::Key::W)) {
                bubble.move(0, -DISTANCE);
        }
        //down
        if (Keyboard::isKeyPressed(Keyboard::Key::Down) ||
        Keyboard::isKeyPressed(Keyboard::Key::S)) {
                bubble.move(0, DISTANCE);
        }
        //left
        if (Keyboard::isKeyPressed(Keyboard::Key::Left) ||
        Keyboard::isKeyPressed(Keyboard::Key::A)) {
                bubble.move(-DISTANCE, 0);
        }
        //right
        if (Keyboard::isKeyPressed(Keyboard::Key::Right) ||
        Keyboard::isKeyPressed(Keyboard::Key::D)) {
                bubble.move(DISTANCE, 0);
        }
        //verify
        if(!validate_position(bubble)) { realign(bubble);}
}
```

PS1A: Fibonacci Linear Feedback Shift Register (FibLFSR)

This assignment was all about implementing the class FibLFSR and its member functions: constructor(s), step() and generate(). Its main goal was to be able to generate random 16 bit numbers (that have been converted to base 10) using the Fibonacci Linear Feedback Shift Register, where the bit string, and the elements at indices corresponding to the first few values of the Fibonacci sequence (0, 2, 3, and 5) were put into a series of XOR operations in order to get a randomized bit in order to feed back at the end of the bit string. Generating a pseudo-random number required calling step() multiple times. An important design aspect in this assignment was encapsulation; putting functionality and variables into a class to be able to contained, allowing for further abstraction. String manipulation was also used, where the bit string (done via std::string) needed to be reversed in order to simulate a left shift.

I also learnt about boost testing, as well as how to link libraries in the Makefile, and boost testing facilitates so much in code testing. Much easier than having a main() program to test the functionality manually. What challenged me, however, was to figure out the best implementation of the container for the bit string in the FibLFSR object, I was torn between a linked list or a vector. They each had their pros and cons (easy insertion/deletion or random access), and was very much decided on ease to implement and what was prioritsed (random access).

Output:

```
seed: 1011011000110110
step 10 times:
0110110001101100    0
1101100011011000    0
1011000110110000    0
0110001101100001    1
1100011011000011    1
1000110110000110    0
0001101100001100    0
0011011000011001    1
0110110000110011    1
1101100001100110    0

generate(5) 7 times:
1100011011000011     3
1101100001100110     6
0000110011001110     14
1001100111011000     24
0011101100000001     1
0110000000101101     13
0000010110111100     28
```

Makefile   Sat Sep 25 09:16:16 2021   1

```
CFLAGS=  -Wall -Werror -std=c++11 -pedantic -c
###-c: dont link, execetubales need be linked!
all: ps1a test
#executable
ps1a: FibLFSR.o main.o
	g++ -Wall -Werror -pedantic FibLFSR.o main.o -o ps1a
test: test.o FibLFSR.o
	g++ -Wall -Werror -pedantic FibLFSR.o test.o -o test -lboost_unit_test_framework
#objects
test.o: test.cpp
	g++ $(CFLAGS) test.cpp
FibLFSR.o: FibLFSR.h FibLFSR.cpp
	g++ $(CFLAGS) FibLFSR.cpp
main.o: FibLFSR.h
	g++ $(CFLAGS) main.cpp
FibLFSR.cpp:
	g++ $(CFLAGS) FibLFSR.cpp
#other
clean:
	\rm *.o ps1a test
cleanbk: \rm *~
superclean: \rm *.o *~ ps1a test

```

main.cpp Sun Sep 05 01:51:47 2021 1

```cpp
/*
main.cpp - testing/debugging via print statements for FibLFSR class

09.15.2021
09.17.2021: Added code to test new fixed length of 16

 Sandra Hawkins

*/



#include <iostream>
#include "FibLFSR.h"
#include <cmath>
using namespace std;

int main () {
        string str = "1011011000110110";
        FibLFSR f(str);
        FibLFSR fff(str);
        FibLFSR ff(str);
        FibLFSR ffff(str);
        cout << "seed: " << f << endl;
        cout << "step 10 times: " << endl;
        int result;
        for (int i = 0; i < 10; i++) {
                //step and show result (int)
                result = f.step();
                cout << f << "\t" << result << endl;
        }
        cout <<"\n";
        cout << "generate(5) 7 times:" << endl;
        for (int i = 0; i < 7; i++) {
                result = fff.generate(5);
                cout << fff << "\t" << result << endl;
        }

        int x = pow(2, str.length()) - 1;
        cout << "seed  : [" << ff << "]\n";
        int y = ff.generate(x);
        cout << "result: [" << ff << "]\n";
        x = y;
        FibLFSR s("1100");
        cout << "pad   : [" << s << "]" << endl;
        FibLFSR l("11000110111101111110");
        cout << "splice: [" << l << "]" << endl;

        return 0;

}

```

FibLFSR.h Sun Sep 05 04:27:01 2021 1

```
/*
FibLFSR.h - class declaration file for FibLFSR class

09.15.2021
09.17.2021: Added get_length()

Sandra Hawkins


*/


#pragma once
#include <string>
using namespace std;

const int NUM_TAP = 4;
class FibLFSR {
        public:
                FibLFSR(string seed);
                FibLFSR();
                int step();
                int generate(int k);
                friend ostream& operator<<(ostream& out, const FibLFSR& f);
                //used in testing
                int get_length() { return bit_string.length();}
                friend bool operator==(const FibLFSR& left, const FibLFSR& r
ight);
        private:
                string bit_string;
                //initialized here cuz everywhere else was giving me issues
                int tap[NUM_TAP] {0,2,3,5};
};
```

```
/*
FibLFSR.cpp - Implementation file for FibLFSR class
(and other helpful functions)

09.15.2021
09.17.2021: Added functions to keep bit string fixed at 16 bits
            Refactored reverse() function.

Sandra Hawkins

*/


#include "FibLFSR.h"
#include <cmath>
const int NUM_BITS = 16;

//other functions

//size adjusting
void swap(char& left, char& right) {
       char hold;
       hold = left; left = right; right = hold;
}
void reverse(string& str) {
       int i = 0, j = str.length() -1, len = str.length() /2;
       for (; i < len; i++, j--) { swap(str[i], str[j]); }
}
void pad_zero(string& bit_string) {
       //until up to num bits pad with zeroes (’0’)
       //"1100"->"0000 0000 0000 1100"
       reverse(bit_string);
       while (bit_string.length() < NUM_BITS) {bit_string.push_back(’0’);}
       reverse(bit_string);
}
void splice(string& bit_string) {
       //only include 1st 16 characters entered (leftmost)
       while (bit_string.length() > NUM_BITS) {bit_string.pop_back();}
}
void adjust(string& bit_string) {
       if (bit_string.length() < NUM_BITS) { pad_zero(bit_string); }
       else if (bit_string.length() > NUM_BITS) { splice(bit_string); }
}

//shifting
int char_to_int(char x) { return x - ’0’;}
char int_to_char(int x) { return x + ’0’; }
bool exor(bool l, bool r) {return ((l && !r) || (!l && r));}
char chexor(char left, char right) {
       int result = exor(char_to_int(left), char_to_int(right));
       return int_to_char(result);
}
void left_shift(string& str) {
```

FibLFSR.cpp Fri Sep 24 20:09:57 2021 2

```
        reverse(str);
        str.pop_back(); //now at the end
        reverse(str);
}
int decimalize(const string& str, int base) {
        int sum = 0, len = str.length() -1;
        for (int i = 0; i <= len; i++) {
                sum += (char_to_int(str[i]) * pow(base, len - i));
        }
        return sum;
}

//operators
ostream& operator<<(ostream& out, const FibLFSR& f) {
        out << f.bit_string; return out;
}
bool operator==(const FibLFSR& left, const FibLFSR& right) {
        return left.bit_string == right.bit_string;
}

//construction
FibLFSR::FibLFSR(string seed): bit_string(seed) {adjust(bit_string);}
FibLFSR::FibLFSR(): FibLFSR("0000000000000000"){}

//public member functions
int FibLFSR::step() {
        char xorval = bit_string[0];
        for (int i = 1; i < NUM_TAP; i++) {
                xorval = chexor(xorval, bit_string[tap[i]]);
        }
        left_shift(bit_string);
        bit_string.push_back(xorval);
        return char_to_int(xorval);
}
int FibLFSR::generate(int k) {
        string result;
        const int BASE_2 = 2;
        for (int i = 0; i < k; i++) {
                result.push_back(int_to_char(step()));
        }
        return  decimalize(result, BASE_2);
}
```

test.cpp   Sat Sep 25 10:51:25 2021   1

```
// Dr. Rykalova
// test.cpp for PS1a
// updated 1/31/2020

#include <iostream>
#include <string>
#include <cmath>

#include "FibLFSR.h"

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>


//tests for correct output for the step and generate with the
//seed 1011 0110 0011 0110
BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {

FibLFSR l("1011011000110110");
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 1);
BOOST_REQUIRE(l.step() == 1);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 1);

 FibLFSR l2("1011011000110110");
 BOOST_REQUIRE(l2.generate(9) == 51);
}

//given some 16-bit string f,  after (2^n) -1 steps, the resulting
//string should cycle back to the form of the original seed
//the seed string is also 14 bits to test if it adjusts correctly.
BOOST_AUTO_TEST_CASE(cycle_back_to_seed) {
        string str = "10110010101111";
        FibLFSR before(str);
        FibLFSR after(str);
        int x = pow(2, before.get_length()) - 1;
        after.generate(x);
        BOOST_REQUIRE(before == after);
}

//like on the doc, just shows that after each generate(5), the correct
//value is given.
//made the seed tring length be greater than 16 to see if it adjusts
//correctly
BOOST_AUTO_TEST_CASE(generate_gives_correct_output) {
        string str = "101101100011011011";
        FibLFSR seed(str);

        const int STEP = 5;
```

test.cpp   Sat Sep 25 10:51:25 2021   2

```
        BOOST_REQUIRE(seed.generate(STEP) == 3); //5 steps
        BOOST_REQUIRE(seed.generate(STEP) == 6); //10 steps
        BOOST_REQUIRE(seed.generate(STEP) == 14); //15 steps
        BOOST_REQUIRE(seed.generate(STEP) == 24); //20 steps
        BOOST_REQUIRE(seed.generate(STEP) == 1); //25 steps
        BOOST_REQUIRE(seed.generate(STEP) == 13); //30 steps
        BOOST_REQUIRE(seed.generate(STEP) == 28); //35 steps

        //do the strings match even after 35 steps (compared to 7 5steps?)
        FibLFSR comp(str);
        comp.generate(35);

        BOOST_REQUIRE(seed == comp);
}
```

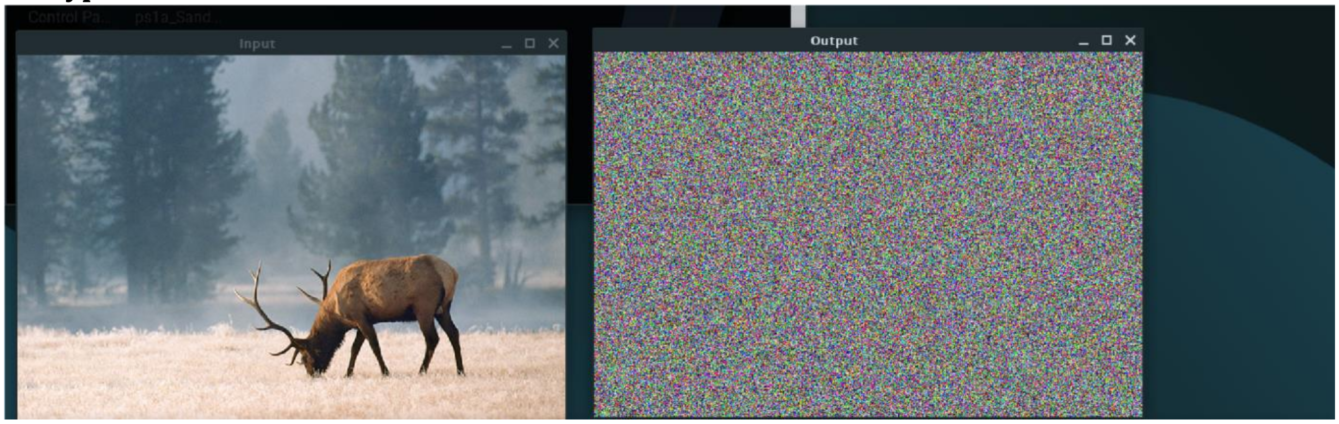
PS1B: PhotoMagic with FibLFSR: Image Encryption and Decryption

By using the FibLFSR implemented from the previous assignment, one could encrypt an image by randomizing its RGB components (values 0-255), for every pixel the image has. The end result would make it look similar to TV static. Decryption is the exact same process: using the same FibLFSR seed from the encryption, an encrpyted image can be decrypted back to its orginal state (component XOR number XOR number is just component XOR 0, which is just the component).

The implementation of this assignment was to implement the transform() function, given a FibLFSR object and sf::Image argument that was passed by reference, further functions were made to make the transform() function be more readable/be delegated to only one task. Abstraction was used in order to acheive this. Transform() called some smaller functions, such as from_base_10() and to_base_10() and convert_component(). My implementation used manipulation of std::string objects, making integers strings, and strings integers, in order to interact with the FibLFSR object as well as be reversible. What I learned was how to use the sf::Image class in order to access pixels, and thus make the encryption/decryption process even possible, as well as learn how to use command line arguments (in lieu of asking the user while the program is being run with std::cin).

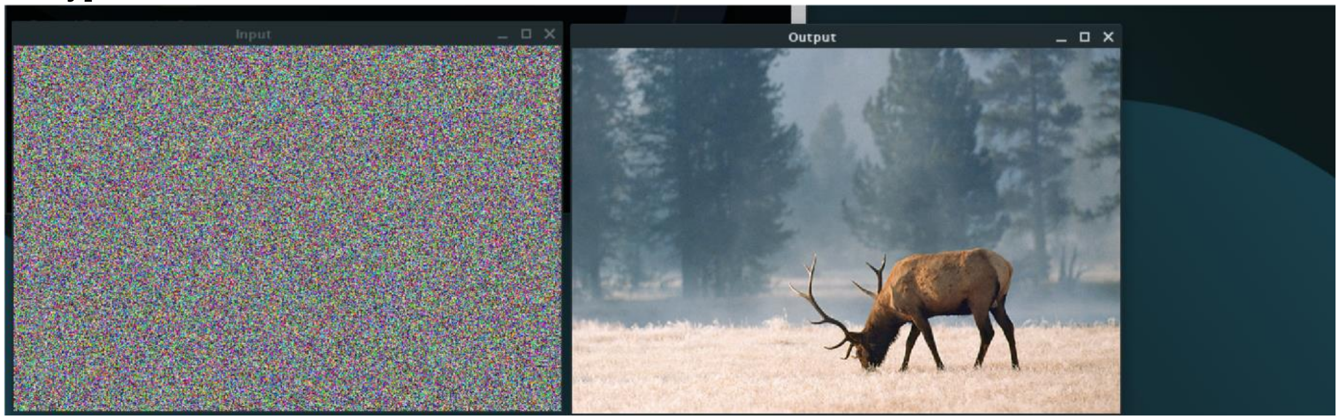
Output:

Encryption:



Input image file (left window) with it then encrypted (right window).

Decryption:



The left window is the input image, encrypted, and the right window is the decrypted image, back in its original state. It is identical to the input image from the encryption.

Makefile Sat Sep 25 08:43:30 2021 1

```
CFLAGS = -Wall -Werror -std=c++11 -pedantic -c
LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system

all: ./PhotoMagic
#executable
PhotoMagic: FibLFSR.o PhotoMagic.o
	g++ FibLFSR.o PhotoMagic.o -o PhotoMagic $(LFLAGS)
#objects
FibLFSR.o: FibLFSR.cpp FibLFSR.h
	g++ $(CFLAGS) FibLFSR.cpp
PhotoMagic.o: FibLFSR.h
	g++ $(CFLAGS) PhotoMagic.cpp
#other
FibLFSR.cpp: FibLFSR.h
	g++ $(CFLAGS) FibLFSR.cpp
clean:
	rm *.o PhotoMagic
cleanbk:
	rm *~
superclean:
	rm *~ *.o PhotoMagic
```

PhotoMagic.cpp Sun Sep 26 03:05:05 2021 1

```
#include <iostream>
#include <SFML/Graphics.hpp>
#include <string>
#include "FibLFSR.h"
using namespace std;
using namespace sf;
const int EIGHT = 8;

void transform(Image& image, FibLFSR& f);
string from_base_10(int num, int base);
int convert_component(int component, int num);
int main(int argc, char** argv) {
        FibLFSR f(argv[3]);
        Image input_image;
        Image output_image;

        if(!input_image.loadFromFile(argv[1])) { return -1;}
        if(!output_image.loadFromFile(argv[1])) { return -1;}


        //transformation
        transform(output_image, f);

        //set the sprites
        Texture input_texture;
        Texture output_texture;
        Sprite input_sprite;
        Sprite output_sprite;


        input_texture.loadFromImage(input_image);
        output_texture.loadFromImage(output_image);
        input_sprite.setTexture(input_texture);
        output_sprite.setTexture(output_texture);


        Vector2u size = input_image.getSize();
        RenderWindow input_window(VideoMode(size.x, size.y), "Input");
        RenderWindow output_window(VideoMode(size.x, size.y), "Output");

        Color color = Color::White;
        while(input_window.isOpen() && output_window.isOpen()) {
                Event event;
                while(input_window.pollEvent(event)) {
                        if (event.type == Event::Closed){input_window.close(
);}
                }
                while(output_window.pollEvent(event)) {
                        if (event.type == Event::Closed){output_window.close
();}
                }
                input_window.clear(color); output_window.clear(color);
                input_window.draw(input_sprite);
```

PhotoMagic.cpp   Sun Sep 26 03:05:05 2021   2

```
                output_window.draw(output_sprite);
                input_window.display(); output_window.display();
        }

        //save the image
        if (!output_image.saveToFile(argv[2])) { return -1;}

        return 0;
}

void transform(Image& image, FibLFSR& f) {
        Color p;
        Vector2u size = image.getSize();
        for (unsigned int x = 0; x < size.x; x++) {
                for (unsigned int y = 0; y < size.y; y++) {
                        p = image.getPixel(x, y);
                        p.r = convert_component(p.r, f.generate(EIGHT));
                        p.g = convert_component(p.g, f.generate(EIGHT));
                        p.b = convert_component(p.b, f.generate(EIGHT));
                        image.setPixel(x , y , p);
                }
        }
}
string from_base_10(int num, int base) {
        string result;
        int remainder;
        while (num > 0) {
                remainder = num % base;
                num = (num - remainder) / base;
                result.push_back(int_to_char(remainder));
        }
        while (result.length() < EIGHT) {result.push_back('0');}
        return result;
}
int convert_component(int component, int num) {
        const int BASE_2 = 2;
        string compon;
        string new_gen;
        compon = from_base_10(component, BASE_2);
        new_gen = from_base_10(num, BASE_2);
        for (int i = 0; i < EIGHT; i++) {
                compon[i] = chexor(compon[i], new_gen[i]);
        }
        reverse(compon);
        return to_base_10(compon, BASE_2);
}
```

FibLFSR.h Sun Sep 05 07:44:09 2021 1

```cpp
/*
FibLFSR.h - class declaration file for FibLFSR class

09.15.2021
09.17.2021: Added get_length()

Sandra Hawkins


*/


#pragma once
#include <string>
using namespace std;

const int NUM_TAP = 4;
class FibLFSR {
        public:
                FibLFSR(string seed);
                FibLFSR();
                int step();
                int generate(int k);
                friend ostream& operator<<(ostream& out, const FibLFSR& f);
                //used in testing
                int get_length() { return bit_string.length();}
                friend bool operator==(const FibLFSR& left, const FibLFSR& right);
        private:
                string bit_string;
                //initialized here cuz everywhere else was giving me issues
                int tap[NUM_TAP] {0,2,3,5};
};

//other functions used
void swap(char& left, char& right);
void reverse(string& str);
char int_to_char(int x);
int char_to_int(char x);
bool exor(bool l, bool r);
char chexor(char left, char right);
int to_base_10(const string& str, int base);
```

FibLFSR.cpp   Sun Sep 05 07:31:07 2021   1

```cpp
/*
FibLFSR.cpp - Implementation file for FibLFSR class
(and other helpful functions)

09.15.2021
09.17.2021: Added functions to keep bit string fixed at 16 bits
           Refactored reverse() function.

Sandra Hawkins

*/


#include "FibLFSR.h"
#include <cmath>
const int NUM_BITS = 16;

//other functions

//size adjusting
void swap(char& left, char& right) {
        char hold;
        hold = left; left = right; right = hold;
}
void reverse(string& str) {
        int i = 0, j = str.length() -1, len = str.length() /2;
        for (; i < len; i++, j--) { swap(str[i], str[j]); }
}
void pad_zero(string& bit_string) {
        //until up to num bits pad with zeroes ('0')
        //"1100"->"0000 0000 0000 1100"
        reverse(bit_string);
        while (bit_string.length() < NUM_BITS) {bit_string.push_back('0');}
        reverse(bit_string);
}
void splice(string& bit_string) {
        //only include 1st 16 characters entered (leftmost)
        while (bit_string.length() > NUM_BITS) {bit_string.pop_back();}
}
void adjust(string& bit_string) {
        if (bit_string.length() < NUM_BITS) { pad_zero(bit_string); }
        else if (bit_string.length() > NUM_BITS) { splice(bit_string); }
}

//shifting
int char_to_int(char x) { return x - '0';}
char int_to_char(int x) { return x + '0'; }
bool exor(bool l, bool r) {return ((l && !r) || (!l && r));}
char chexor(char left, char right) {
        int result = exor(char_to_int(left), char_to_int(right));
        return int_to_char(result);
}
void left_shift(string& str) {
        reverse(str);
        str.pop_back(); //now at the end
        reverse(str);
}
int to_base_10(const string& str, int base) {
        int sum = 0, len = str.length() -1;
        for (int i = 0; i <= len; i++) {
                sum += (char_to_int(str[i]) * pow(base, len - i));
        }
        return sum;
}

//operators
ostream& operator<<(ostream& out, const FibLFSR& f) {
        out << f.bit_string; return out;
}
bool operator==(const FibLFSR& left, const FibLFSR& right) {
        return left.bit_string == right.bit_string;
}

//construction
FibLFSR::FibLFSR(string seed): bit_string(seed) {adjust(bit_string);}
FibLFSR::FibLFSR(): FibLFSR("0000000000000000"){}

//public member functions
int FibLFSR::step() {
        char xorval = bit_string[0];
        for (int i = 1; i < NUM_TAP; i++) {
                xorval = chexor(xorval, bit_string[tap[i]]);
        }
        left_shift(bit_string);
        bit_string.push_back(xorval);
        return char_to_int(xorval);
}
int FibLFSR::generate(int k) {
        string result;
        const int BASE_2 = 2;
        for (int i = 0; i < k; i++) {
                result.push_back(int_to_char(step()));
        }
        return  to_base_10(result, BASE_2);
}
```

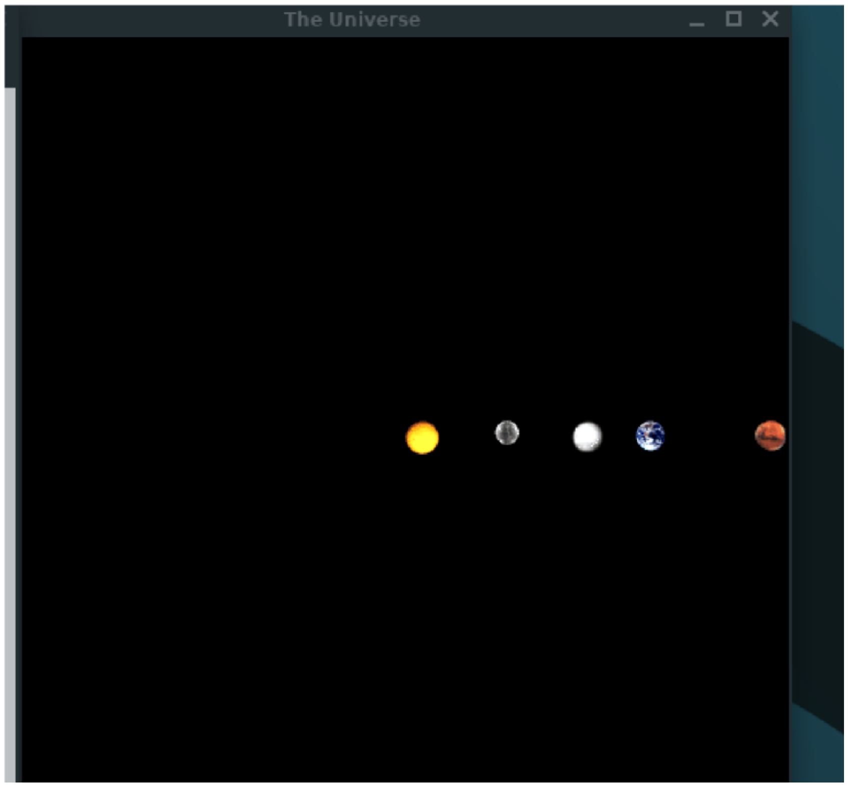
PS2A: NBody Simulation: Universe and CelestialBody Implementation

The overall goal of this assignment was to be able to simulate the (innner) solar system by using SMFL sprites. With a given text file (also with correct formatting), the program would be able to display the given celestial body from the input file in the correct position. The implementation of this was done by the class CelestialBody, which, as the name suggests, holds data for a single CelestialBody. Extraction was overloaded to accommodate the CelestialBody class where it would be able to read all of its necessary members with ease. A container to hold all of the pointers to the CelestialBody objects (given by the input file) was implemented by the class Universe, which had the member std::vector<CelestialBody*> to hold the bodies. The CelestialBody class also had to be sf::Drawable(), so it could be able to be drawn to the SFML window like any other native SFML object.

What made this assignment diffiicult was making the sprites show on the window. I had many problems from it not displaying properly, whether by segmentation faults or only showing a white square. I eventually realized that I was not saving the texture and image data in order to make a proper sprite for each instance of a CelestialBody object. I also learnt about input redirection, which is much more streamlined than using <fstream> for reading and writing to files.

<u>Output:</u>

The expected output is for all bodies from planets.txt to be in proper order based on their position: The sun, Mercury, Venus, Earth, and Mars.



SFML window titled "The Universe" displaying the inner solar system in their correct positions via the use of SFML sprites, drawn as CelestialBody objects to the screen.

Makefile   Thu Sep 23 14:39:36 2021   1

```
CFLAGS = -Wall -Werror -std=c++11 -pedantic -c
LFLAGS = -lsfml-system -lsfml-window -lsfml-graphics

#
all: ./NBody

#executable(s)
NBody: Universe.o CelestialBody.o NBody.o
       g++ Universe.o CelestialBody.o NBody.o -o NBody $(LFLAGS)
#objects
Universe.o: CelestialBody.cpp Universe.cpp Universe.h
       g++ $(CFLAGS) CelestialBody.cpp Universe.cpp
CelestialBody.o: CelestialBody.cpp CelestialBody.h
       g++ $(CFLAGS) CelestialBody.cpp
Nbody.o: Universe.h Nbody.cpp
       g++ $(CLFAGS) Nbody.cpp
#implementations
Universe.cpp: Universe.h
       g++ $(CFLAGS) Universe.cpp
CelestialBody.cpp: CelestialBody.h
       g++ $(CFLAGS) CelestialBody.cpp
#removal
clean:
       rm *.o NBody
cleanbk:
       rm *~
superclean:
        rm *.o *~ NBody
```

NBody.cpp Sat Sep 25 18:04:03 2021 1

```
/*
NBody.cpp
main program file for the universe simulation.

09.27.2021

Sandra Hawkins

*/
#include <iostream>
#include "Universe.h"
#include <SFML/Graphics.hpp>
#include <string>
using namespace std;
using namespace sf;

double CelestialBody::RADIUS;
int main(int argc, char** argv) {
        //read in 1st 2 values in file
        int nBodies;
        cin >> nBodies >> CelestialBody::RADIUS;
        //set up universe
        Universe U(nBodies, CelestialBody::RADIUS);

        //display loop (draw the CelestialBodies)
        RenderWindow window(VideoMode(U.get_window_size(), U.get_window_size
()),
         "The Universe");

        while(window.isOpen()) {
                Event event;
                while (window.pollEvent(event)) {
                        if(event.type == Event::Closed) {window.close();}
                }
                window.clear();
                window.draw(U);
                window.display();
        }
        return 0;
}
```

Universe.h Thu Sep 23 22:28:48 2021 1

```
/*
Universe.h : header file for Universe class
provides functionality for usage of many CelestialBody objects at once

09.29.2021

 Sandra Hawkins

*/

#include "CelestialBody.h"
class Universe: public Drawable {
        public:
                Universe(int n, double r);
                Universe();
                ~Universe();
                int get_window_size() { return window_size;}
        private:
                int nBodies;
                double radius;
                int window_size;
                CelestialBody** pBodies;
                virtual void draw(RenderTarget& target, RenderStates states)
const;
};

```

Universe.cpp Thu Sep 23 22:29:18 2021 1

```
/*
Universe.cpp : Implementation file for the Universe
class

09.29.021

Sandra Hawkins

*/
#include "Universe.h"
#include <iostream>
void clear_buffer(istream& in) {
        in.clear();
        in.ignore(numeric_limits<streamsize>::max(), '\n');
}

//struction
Universe::Universe(int n, double r) : Drawable(),
        nBodies(n), radius(r), window_size(2*radius*SCALE_FACTOR) {
        pBodies = new CelestialBody*[nBodies];
        for (int i = 0; i < nBodies; i++) {
                pBodies[i] = nullptr;
                pBodies[i] = new CelestialBody;
                cin >> *pBodies[i];
                clear_buffer(cin);
                //cout << *pBodies[i];
        }
}
Universe::Universe() : Universe(0, 0) {}
Universe::~Universe() {
        for (int i = 0; i < nBodies; i++) { delete pBodies[i]; }
        delete [] pBodies; pBodies = nullptr;
}
void Universe::draw(RenderTarget& target, RenderStates states) const {
        for (int i = 0; i <  nBodies; i++) {
                target.draw(*pBodies[i]);
        }
}
```

CelestialBody.h Fri Sep 24 20:31:10 2021 1

```
/*
CelestialBody.h
        header file for the CelestialBody class.

        09.27.2021

        Sandra Hawkins
*/
#include <SFML/Graphics.hpp>
#include <iostream>
#include <string>
using namespace std;
using namespace sf;

const double SCALE_FACTOR = 1e-9;
class CelestialBody : public Drawable  {
        public:
        static double RADIUS;
        CelestialBody(double xp, double yp, double xv, double yv,
                        double m, string f);
        CelestialBody() : CelestialBody(0, 0, 0, 0, 0, "") {}
        friend istream& operator>>(istream& in, CelestialBody& body);

        //debugging, shows to screen data from text file
        friend ostream& operator<< (ostream& out, const CelestialBody& body)
;
        private:
                double xpos;
                double ypos;
                double xvel;
                double yvel;
                double mass;
                string filename;
                Image image;
                Texture texture;
                Sprite sprite;
                virtual void draw(RenderTarget& target, RenderStates states)
const;
                void set_sprite(Sprite& sprite, Texture& texture, Image& ima
ge);
                float transform(double component);
};
```

CelestialBody.cpp Fri Sep 24 20:31:21 2021 1

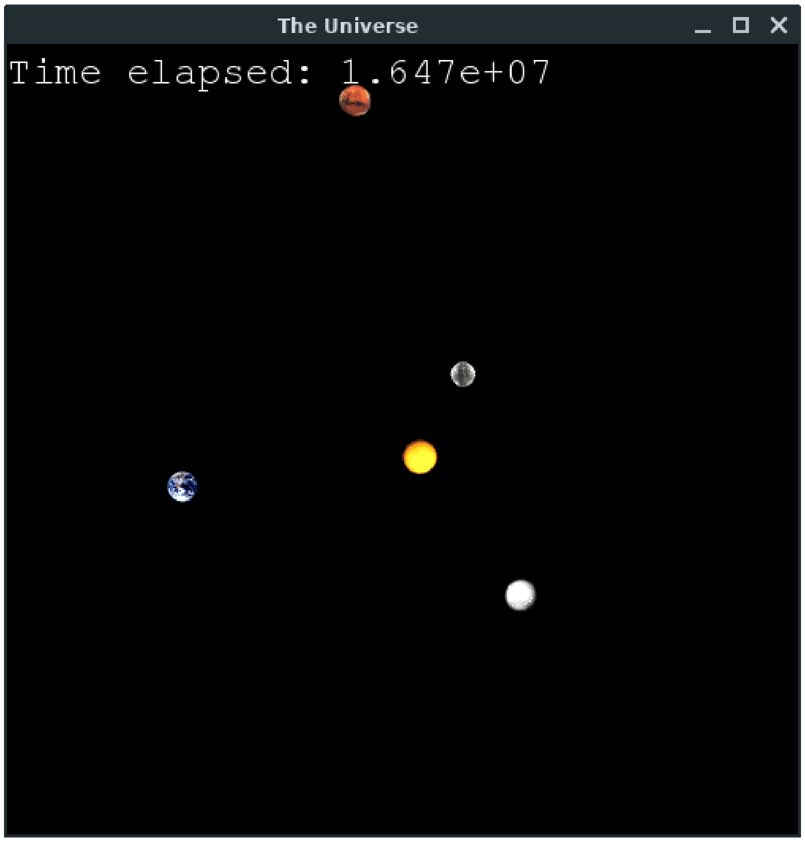
```
/*
CelestialBody.cpp
implementation file for the class declared in CelestialBody.h

09.27.2021

Sandra Hawkins
*/
#include "CelestialBody.h"
#include <cmath>
float CelestialBody::transform(double component) {
        component += RADIUS;
        component *= SCALE_FACTOR;
        component = round(component);
        return component;
}
void CelestialBody::set_sprite(Sprite& sprite, Texture& texture, Image& imag
e) {
        if(!image.loadFromFile(filename)) { exit(-1);}
        texture.loadFromImage(image);
        sprite.setTexture(texture);
        sprite.setPosition(transform(xpos), transform(ypos));
}

CelestialBody::CelestialBody(double xp, double yp, double xv, double yv,
                                double m, string f): Drawable(),
                                xpos(xp), ypos(yp), xvel(xv), yvel(yv),
                                mass(m), filename(f) {}
//operators
istream& operator>>(istream& in, CelestialBody& body) {
        in >> body.xpos >> body.ypos >>  body.xvel >> body.yvel >>
        body.mass >> body.filename;
        body.set_sprite(body.sprite, body.texture, body.image);
        return in;
}
ostream& operator<<(ostream& out, const CelestialBody& body) {
        cout << "SI position: (" << body.xpos << " , " << body.ypos << ")\n"
;
        cout << "SI velocity: (" << body.xvel << " , " << body.yvel << ")\n"
;
        cout << "Mass (kg)  : " << body.mass << endl;
        cout << "Body filename: [" << body.filename << "]" << endl;
        cout << "Pixel Position: (" << body.sprite.getPosition().x << ", "
        << body.sprite.getPosition().y << ")\n" << endl;
        return out;
}
void CelestialBody::draw(RenderTarget& target, RenderStates states) const {
        target.draw(sprite);
}

```

PS2B: NBody Simulation: Animating Orbit

Using the sprites and other data extracted from the previous assignment, animate their realistic orbit according to the laws of physics. That is, accurately determine their change in position and motion based on gravitational forces amongst all involved bodies (in this case the inner solar system and the sun). The animation was implemented via the class Universe, as well as via the use of functions to calculate the exact physics. The Universe class held an std::vector of std::unique_ptr<CelestialBody> objects, as well as a step() function to simulate motion for a given amount of seconds.

What challenged me in this assignment was the implementation of the physics; I had difficulty generalising it and still make it be somewhat efficient (hard to think through). Another challenge was the implementation of the smart pointers. It was hard to eventually get right. In the process I learnt their importance, as well as the overall preference of pointers to components rather than the actual component of an object within class declarations. I also learned how to display SFML text to the SFML window by using a font file (.ttf extension), which I did not know how to do before.

Output:



```
Time elapsed: 1.647e+07
```

Screenshot of NBody program being run with command line arguments 25000000 and 10000, where 2.5e7 is the total running time of the simulation, and 10000 is the time interval for each call of step().

<u>**Output from insertion:**</u>

```
3.7457e+10 1.4514e+11 2.8792e+04 -7.4489e+03 5.9740e+24 earth.gif (287, 395)

-2.0060e+11 -1.0596e+11 -1.1306e+04 2.1407e+04 6.4190e+23 mars.gif (49, 144)

-1.2468e+10 -5.6406e+10 -4.6948e+04 1.0066e+04 3.3020e+23 mercury.gif (238, 194)

8.1705e+05 -4.9093e+06 8.2250e-03 -1.9829e-01 1.9890e+30 sun.gif (250, 250)

-2.6710e+10 -1.0456e+11 -3.4012e+04 8.6486e+03 4.8690e+24 venus.gif (223, 145)
```

elements:
final x-position
final y-position
final x-velocity
final y-velocity
mass
filename
pixel position (relative to window)

Makefile Sat Sep 25 18:33:13 2021 1

```
CFLAGS = -Wall -Werror -std=c++14 -pedantic -c
LFLAGS = -lsfml-system -lsfml-window -lsfml-
graphics

#
all: ./NBody

#executable(s)
NBody: Universe.o CelestialBody.o NBody.o
	g++ Universe.o CelestialBody.o NBody.o -o NBody $(LFLAGS)
#objects
Universe.o: CelestialBody.cpp Universe.cpp Universe.h
	g++ $(CFLAGS) CelestialBody.cpp Universe.cpp
CelestialBody.o: CelestialBody.cpp CelestialBody.h
	g++ $(CFLAGS) CelestialBody.cpp
Nbody.o: Universe.h Nbody.cpp
	g++ $(CLFAGS) Nbody.cpp
#implementations
Universe.cpp: Universe.h
	g++ $(CFLAGS) Universe.cpp
CelestialBody.cpp: CelestialBody.h
	g++ $(CFLAGS) CelestialBody.cpp
#removal
clean:
	rm *.o NBody
cleanbk:
	rm *~
superclean:
	 rm *.o *~ NBody
```

NBody.cpp Sat Nov 13 18:13:09 2021 1

```
/*
NBody.cpp
main program file for the universe simulation.

09.27.2021

Sandra Hawkins

*/
#include <iostream>
#include "Universe.h"
#include <SFML/Graphics.hpp>
#include <string>
#include <sstream>
#include <cmath>
using namespace std;
using namespace sf;

double CelestialBody::RADIUS;
int main(int argc, char** argv) {
        //read in 1st 2 values in file
        int nBodies;
        cin >> nBodies >> CelestialBody::RADIUS;
        //set up universe
        Universe U(nBodies, CelestialBody::RADIUS);
        double T =atof(argv[1]);
        double dt = atof(argv[2]);

        //display loop (draw the CelestialBodies)
        RenderWindow window(VideoMode(U.get_window_size(), U.get_window_size
()),
         "The Universe");


        //showing text
        Font font;
        Text text;
        double totalT = 0;
        string str = "Time elapsed: ", strT;
        stringstream ss; ss << totalT; ss >> strT;
        if(!font.loadFromFile("cour.ttf")) { return -1;}
        text.setFont(font); text.setString(str + strT);
        text.setCharacterSize(25); text.setFillColor(Color::White);
        text.setPosition(0, 0);

        Time interval = milliseconds(50);
        Clock clock;
        while(window.isOpen()) {
                Event event;
                Time t = clock.getElapsedTime();
                if (t >= interval && totalT < T) {
                        totalT += dt;
                        if (totalT > T) {dt -= (totalT - T); totalT = T; }
```

NBody.cpp Sat Nov 13 18:13:09 2021 2

```
                        U.step(dt);
                        stringstream ss; ss << totalT; ss >> strT;
                        text.setString(str + strT);
                        clock.restart(); }
                while (window.pollEvent(event)) {
                        if(event.type == Event::Closed) {window.close();}
                }
                window.clear();
                window.draw(text);
                window.draw(U);
                window.display();
        }
        cout << U << endl;
        return 0;
}
```

Universe.h   Fri Sep 24 17:05:14 2021   1

```
/*
Universe.h : header file for Universe class
provides functionality for usage of many CelestialBody objects at once

09.29.2021

10.04.2021: overloaded << operator for the Universe class in order to
        print all of the CelestialBody objects in its container.

            changed the "pointers to CelestialBody objects" container
            into a vector from a dynamic array.

            made "pointers to CelestialBody objects" be smart pointers
            (unique pointers). As result, removed the destructor, as
            no longer raw pointers are used for memory allocation.

10.06.2021:
           Implemented step() function.

Sandra Hawkins

*/

#include "CelestialBody.h"
#include <vector>
#include <memory>
class Universe: public Drawable {
        public:
                Universe(int n, double r);
                Universe();
                int get_window_size() { return window_size;}
                friend ostream& operator<<(ostream& out, const Universe& U);
                void step(double seconds);
        private:
                int nBodies;
                double radius;
                int window_size;
                vector<unique_ptr<CelestialBody>> pBodies;
                virtual void draw(RenderTarget& target, RenderStates states)
const;
};

```

Universe.cpp Wed Oct 27 22:40:25 2021 1

```
/*
Universe.cpp : Implementation file for the Universe class

09.29.021

Sandra Hawkins

*/
#include "Universe.h"
#include <iostream>
#include <cmath>

const double G = 6.67e-11; //universal gravitational constant
void clear_buffer(istream& in) {
        in.clear();
        in.ignore(numeric_limits<streamsize>::max(), '\n');
}

//struction
Universe::Universe(int n, double r) : Drawable(),
        nBodies(n), radius(r), window_size(2*radius*SCALE_FACTOR) {
        pBodies.resize(nBodies);
        for (int i = 0; i < nBodies; i++) {
                pBodies[i] = make_unique<CelestialBody>();
                cin >> *pBodies[i];
                clear_buffer(cin);
        }
}
Universe::Universe() : Universe(0, 0) {}
void Universe::draw(RenderTarget& target, RenderStates states) const {
        for (int i = 0; i <  nBodies; i++) {
                target.draw(*(pBodies[i]));
        }
}


//operators
ostream& operator<<(ostream& out, const Universe& U) {
        for (int i = 0; i < U.nBodies; i++) {
                out << *(U.pBodies[i]) << endl;
        }
        return out;
}


//physics
Vector2f get_force(const unique_ptr<CelestialBody>& obj1,
                   const vector<unique_ptr<CelestialBody>>& obj2) {
        Vector2f gForce(0, 0);
        double dr, dx, dy, force;
        //get netforce here
        for (int i = 0; i < (int)(obj2.size()); i++) {
                if (obj1 == obj2[i]) continue;
```

Universe.cpp Wed Oct 27 22:40:25 2021 2

```
                dx = obj2[i]->get_xpos() - obj1->get_xpos();
                dy = obj2[i]->get_ypos() - obj1->get_ypos();
                dr = sqrt(pow(dx, 2) + pow(dy, 2));
                force = (G * obj1->get_mass() * obj2[i]->get_mass()) / pow(dr, 2);
                gForce.x += (float)((dx/dr) * force);
                gForce.y += (float)((dy/dr) * force);
        }
        return gForce;

}
void Universe::step(double seconds) {
        //for each body:
        //calculate net force acting onto it
        //calculate resulting acceleration
        //calculate veclocities

        Vector2f gForce;
        Vector2f acceleration;
        double vx, vy;
        for (int i = 0; i < nBodies; i++) {
                gForce = get_force(pBodies[i], pBodies);
                acceleration.x = (float)(gForce.x / pBodies[i]->get_mass());
                acceleration.y = (float)(gForce.y / pBodies[i]->get_mass());
                vx = pBodies[i]->get_xvel() + (acceleration.x * seconds);
                vy = pBodies[i]->get_yvel() + (acceleration.y * seconds);
                pBodies[i]->set_xvel(vx); pBodies[i]->set_yvel(vy);
        }

        //go through loop again for position.
        double px, py;
        for (int i = 0; i < nBodies; i++) {
                px = pBodies[i]->get_xpos() + (pBodies[i]->get_xvel() * seconds);
                py = pBodies[i]->get_ypos() + (pBodies[i]->get_yvel() * seconds);
                pBodies[i]->set_xpos(px); pBodies[i]->set_ypos(py);
                pBodies[i]->set_sprite_position();
        }
}
```

CelestialBody.h   Fri Sep 24 20:51:50 2021   1

```cpp
/*
CelestialBody.h
        header file for the CelestialBody class.

        09.27.2021

        10.06.2021:
                  added accessor functions to retrieve private
                  member variables.

                  added mutator functions to change private memeber
                  variables.

 added public member function to update sprite position from Universe class

        Sandra Hawkins
*/
#include <SFML/Graphics.hpp>
#include <iostream>
#include <string>
using namespace std;
using namespace sf;

const double SCALE_FACTOR = 1e-9;
class CelestialBody : public Drawable  {
        public:
        static double RADIUS;
        CelestialBody(double xp, double yp, double xv, double yv,
                        double m, string f);
        CelestialBody() : CelestialBody(0, 0, 0, 0, 0, "") {}
        friend istream& operator>>(istream& in, CelestialBody& body);
        friend ostream& operator<< (ostream& out, const CelestialBody& body);
        double get_xpos() const {return xpos;}
        double get_ypos() const {return ypos;}
        double get_xvel() const {return xvel;}
        double get_yvel() const {return yvel;}
        double get_mass() const {return mass;}
        void   set_xpos(double x) {xpos = x;}
        void   set_ypos(double y) {ypos = y;}
        void   set_xvel(double x) {xvel = x;}
        void   set_yvel(double y) {yvel = y;}
        void   set_sprite_position() {sprite.setPosition(transform(xpos), transform(ypos));}
        private:
                double xpos;
                double ypos;
                double xvel;
                double yvel;
                double mass;
                string filename;
                Image image;
                Texture texture;
```

CelestialBody.h Fri Sep 24 20:51:50 2021 2

```
                    Sprite sprite;
                    virtual void draw(RenderTarget& target, RenderStates states)
 const;
                    void set_sprite(Sprite& sprite, Texture& texture, Image& ima
 ge);
                    float transform(double component);
 };
```

CelestialBody.cpp Fri Sep 24 20:52:04 2021 1

```
/*
CelestialBody.cpp
implementation file for the class declared in CelestialBody.h

09.27.2021

10.04.2021: redid the overloading of the << operator to more accurately
           resemble the input line from planets.txt
           added one extra element: pixel position (position relative
to the window).

Sandra Hawkins
*/
#include "CelestialBody.h"
#include <cmath>
#include <iomanip>
float CelestialBody::transform(double component) {
        component += RADIUS;
        component *= SCALE_FACTOR;
        component = round(component);
        return component;
}
void CelestialBody::set_sprite(Sprite& sprite, Texture& texture, Image& image) {
        if(!image.loadFromFile(filename)) { exit(-1);}
        texture.loadFromImage(image);
        sprite.setTexture(texture);
        sprite.setPosition(transform(xpos), transform(ypos));
}

CelestialBody::CelestialBody(double xp, double yp, double xv, double yv,
                             double m, string f): Drawable(),
                             xpos(xp), ypos(yp), xvel(xv), yvel(yv),
                             mass(m), filename(f) {}
//operators
istream& operator>>(istream& in, CelestialBody& body) {
        in >> body.xpos >> body.ypos >>  body.xvel >> body.yvel >>
        body.mass >> body.filename;
        body.set_sprite(body.sprite, body.texture, body.image);
        body.yvel *= -1;
        return in;
}
ostream& operator<<(ostream& out, const CelestialBody& body) {
        out.precision(4);
        scientific(out);
        out << body.xpos << " " << body.ypos << " " <<
        body.xvel << " " << body.yvel << " " << body.mass << " " <<
        body.filename;
        out << " (" << (int)body.sprite.getPosition().x << ", " <<
        (int)body.sprite.getPosition().y << ")";
        return out;
}
```

CelestialBody.cpp Fri Sep 24 20:52:04 2021 2

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

```

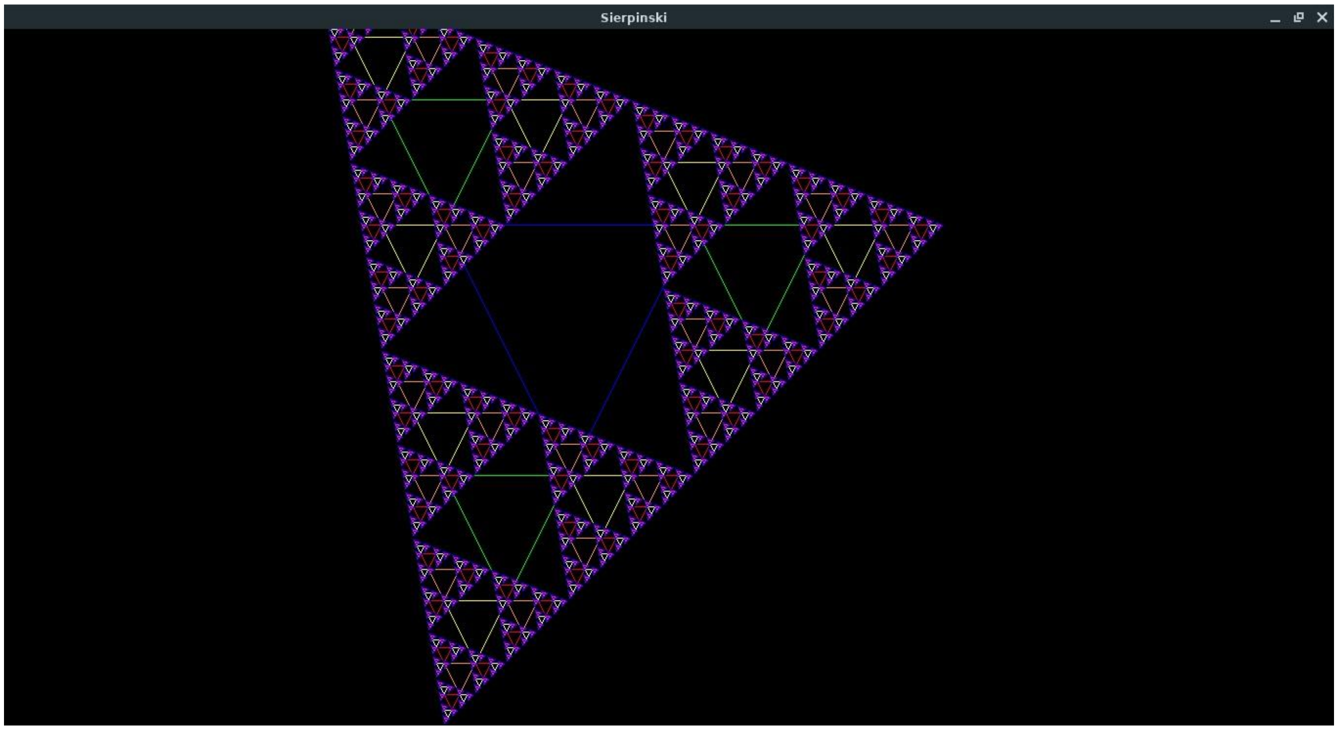
PS3: Recursive Graphics : Sierpinski Triangle Fractal

By using recursion to draw continuous triangles to the window, a fractal is eventually drawn. In this case, drawing a Sierpinski triangle, given recursion depth level and side length, from the command line using SFML graphics libraries. Recursion was the key to success in this assignment. Not only that it is neater than an iterative implementation, but also that the pattern was recursive: at every vertex another triangle is drawn until a basis is reached.

Drawing the triangle itself was done via a class known as Triangle, that was inherited from sf::Drawable. Where the exact method of the implementation of the class was not specified. In my case, my Triangle class only had constructors with up to four parameters: side length, originating vertex (constant), current point, and color constant. This allowed for easy construction of the triangle where it would need not be modified at any later point in time, which allowed for it to be drawn easily to the window.

I also learned about cpplint, coding standards to make code look nicer/readable, as well as learnt how to use the sf::Vertex class on drawing polygons from lines and points alone. What challenged me was the recursion actually. It was very difficult for me to figure out (and visualize) the best way to implement the fTree() function, the function where the recursion occurs.

__Output:__



Screenshot of TFractal program with starting side length of 256 and recursion depth of 13.
An extra feature was added so that the coloring of the fractal would vary with depth, where each level is differently colored.

Makefile Sun Sep 26 12:13:25 2021 1

```
CFLAGS = -Wall -Werror -pedantic -std=c++14 -c
LFLAGS = -lsfml-system -lsfml-window -lsfml-graphics

all: ./TFractal
TFractal: TFractal.o Triangle.o
	g++ TFractal.o Triangle.o -o TFractal $(LFLAGS)
TFractal.o: TFractal.cpp Triangle.cpp
	g++ $(CFLAGS) TFractal.cpp Triangle.cpp
Triangle.o: Triangle.cpp Triangle.h
	g++ $(CFLAGS) Triangle.cpp
Triangle.cpp: Triangle.h
	g++ $(CFLAGS) Triangle.cpp
clean:
	rm *.o TFractal
cleanbk:
	rm *~
superclean:
	rm *~ *.o TFractal
```

TFractal.cpp   Sun Sep 26 12:13:43 2021   1

```
// Copyright
#include <iostream>
#include<SFML/Graphics.hpp>
#include "Triangle.h"

sf::RenderWindow window(sf::VideoMode(2000, 2000), "Sierpinski");
void fTree(int d, int n, sf::Vector2f p, const int ref);
int main(int argc, const char* argv[]) {
 int L = atoi(argv[1]); int N = atoi(argv[2]);
 while (window.isOpen()) {
   sf::Event event;
   while (window.pollEvent(event)) {
     if (event.type == sf::Event::Closed) { window.close();}
   }
   window.clear();
   fTree(N, L, sf::Vector2f(450, 200), LEFT);
   window.display();
 }
 return 0;
}

void fTree(int d, int n, sf::Vector2f p, const int ref) {
 if (d <= 0) { return;}
 // draw the triangle according to the reference, color is chosen according
 // to depth.
 Triangle t(n, p, ref, (d % NUM_COLOR));
 window.draw(t);
 n /= 2; d--;
 // call fTree for the next triangles
 fTree(d, n, t.get_right(), LEFT);
 fTree(d, n, t.get_mid(), RIGHT);
 fTree(d, n, t.get_left(), MID);
}
```

Triangle.h Sun Nov 21 17:06:54 2021 1

```
// Copyright 2021
#pragma once
#include <SFML/Graphics.hpp>
/*
Triangle.h: header file for the Drawable Triangle class

10.11.2021

Sandra Hawkins


*/

enum sides {LEFT, RIGHT, MID, NUM_VERTEX};
enum points {START, END, NUM_POINT};
enum colors {WHITE, RED, ORANGE, YELLOW, GREEN, BLUE, INDIGO, MAGENTA ,
NUM_COLOR = 8};
const int TRIANGLE = 3;

class Triangle: public sf::Drawable {
public:
   Triangle();
   explicit Triangle(int n);
   Triangle(int n, sf::Vector2f p);
   Triangle(int n, sf::Vector2f p, int reference, int c);
   sf::Vector2f get_left() {return left_pos;}
    sf::Vector2f get_right() {return right_pos;}
   sf::Vector2f get_mid() {return mid_pos;}
private:
   virtual void draw(sf::RenderTarget &target, sf::RenderStates states) con
st; // NOLINT
   int side_length;
   sf::Vector2f origin;
   sf::Vector2f left_pos;
   sf::Vector2f right_pos;
   sf::Vector2f mid_pos;
   sf::Vertex sides[TRIANGLE][NUM_POINT];
   sf::Color color;
};
```

Triangle.cpp Sun Nov 21 17:06:46 2021 1

```
// Copyright 2021
#include "Triangle.h"
/*
Triangle.cpp: Implementation file of the Drawable Triangle class.

10.11.2021

Sandra Hawkins

*/


/*
constructs an equilateral triangle of sidelength n, originating
from Vector p, and how it is drawn is dependent on the reference:
LEFT, RIGHT, or MID, such that any given vertex in a triangle can
act as its origin.

if not all parmaters are given, it defaults to 0.

The color of the triangle (coolor of its sides), varies depending
on its given color state constant. With the fTree() function, the
color chosen varies upon the depth of the recursion.
*/

sf::Color set_color(const int color_state) {
 sf::Color color;
 sf::Color orange; orange.r = 255; orange.g = 150; orange.b = 50;
 sf::Color indigo; indigo.r = 100; indigo.g = 0; indigo.b = 200;
 switch (color_state) {
   case WHITE: color = sf::Color::White; break;
   case RED: color = sf::Color::Red; break;
   case ORANGE: color = orange; break;
   case YELLOW: color = sf::Color::Yellow; break;
   case GREEN: color = sf::Color::Green; break;
   case BLUE: color = sf::Color::Blue; break;
   case INDIGO: color = indigo; break;
   case MAGENTA: color = sf::Color::Magenta; break;
   default: color = sf::Color::White; break;
 }
 return color;
}
Triangle::Triangle(int n, sf::Vector2f p, int reference, int c) :
 sf::Drawable(), side_length(n), origin(p), color(set_color(c)) {
    switch (reference) {
     case LEFT:
       left_pos = p;
       right_pos = sf::Vector2f(p.x + n, p.y);
       mid_pos = sf::Vector2f(p.x + (n/2), p.y + n);
       // left->right
       sides[LEFT][START].position = left_pos;
       sides[LEFT][END].position = right_pos;
       // right->mid
       sides[RIGHT][START].position = sides[LEFT][END].position;
```

Triangle.cpp Sun Nov 21 17:06:46 2021 2

```
        sides[RIGHT][END].position = mid_pos;
        // mid->left
        sides[MID][START].position = sides[RIGHT][END].position;
        sides[MID][END].position = sides[LEFT][START].position;
      break;
      case RIGHT:
        right_pos = p;
        mid_pos = sf::Vector2f(p.x - (n/2), p.y + n);
        left_pos = sf::Vector2f(p.x - n, p.y);
        // right->mid
        sides[RIGHT][START].position = right_pos;
        sides[RIGHT][END].position = mid_pos;
        // mid->left
        sides[MID][START].position = sides[RIGHT][END].position;
        sides[MID][END].position = left_pos;
        // left->right
        sides[LEFT][START].position = sides[MID][END].position;
        sides[LEFT][END].position = sides[RIGHT][START].position;
      break;
      case MID:
        mid_pos = p;
        left_pos = sf::Vector2f(p.x - (n/2), p.y - n);
        right_pos = sf::Vector2f(p.x + (n/2), p.y - n);
        // mid->left
        sides[MID][START].position = mid_pos;
        sides[MID][END].position = left_pos;
        // left->right
        sides[LEFT][START].position = sides[MID][END].position;
        sides[LEFT][END].position = right_pos;
        // right->mid
        sides[RIGHT][START].position = sides[LEFT][END].position;
        sides[RIGHT][END].position = sides[MID][START].position;
      break;
      }
      // color-setting:
      for (int i = 0; i < NUM_VERTEX; i++) {
        for (int j = 0; j < NUM_POINT; j++) {
          sides[i][j].color = color;
         }
      }
}
Triangle::Triangle(int n, sf::Vector2f p) : Triangle(n, p, LEFT, WHITE) {}
Triangle::Triangle(int n): Triangle(n, sf::Vector2f(0, 0)) {}
Triangle::Triangle() : Triangle(0) {}

void Triangle::draw(sf::RenderTarget &target, sf::RenderStates states) const
{
 target.draw(sides[LEFT], NUM_POINT, sf::Lines);
 target.draw(sides[RIGHT], NUM_POINT, sf::Lines);
 target.draw(sides[MID], NUM_POINT, sf::Lines);
}
```

PS4A: SFML Audio Output: CircularBuffer Implementation

The purpose of this assignment was to implement the class CircularBuffer, which was a Circular Array to hold int16_t integers. It also had to be tested using boost's unit test framework. The CircularBuffer was implemtened by using a vector and having the front and back indices not be at the true beginning and end of the vector, but rather at any index, updated accordingly with each enqueue() and dequeue(). Using a class, CircularBuffer, helped with the abstraction as well as encapsulation (kept all details hidden in one object). Lambda expressions were (optionally) implementable, but were welcome (so I used several).

What challenged me was how to best implement the CircularBuffer, as what I had on paper was not entirely working. It was difficult to figure out the process of enqueing and dequeing a circular array, so I had to some additional research/googling regarding it. What I learnt was how to use lambda expressions properly, std::chrono for timekeeping, std::algorithm and as a result got better with iterators, as well as learn how to use std::exceptions instead of always throwing empty classes.

Output:

```
0    1    2    3    4    5    6    7    8    9

1    2    3    4    5    6    7    8    9    0

2    3    4    5    6    7    8    9    0    -1

3    4    5    6    7    8    9    0    -1   -2

4    5    6    7    8    9    0    -1   -2   -3

5    6    7    8    9    0    -1   -2   -3   -4

6    7    8    9    0    -1   -2   -3   -4   -5

7    8    9    0    -1   -2   -3   -4   -5   -6

8    9    0    -1   -2   -3   -4   -5   -6   -7

9    0    -1   -2   -3   -4   -5   -6   -7   -8

0    -1   -2   -3   -4   -5   -6   -7   -8   -9
```

Shown above is the output from a lambda expression that dequeues the current front, negates it, and then feeds it back into the buffer via enqueue(). Insertion was also overloaded to print the proper order of elements in the buffer. This lambda was called for the entire length of the buffer (in this case 10), in order to show how it 'circles back' eventually.

Makefile Fri Nov 05 18:20:38 2021 1

```
CFLAGS = -Wall -Werror -pedantic -std=c++14 -c -O1
LBOOST = -lboost_unit_test_framework

all: ps4a test

#exectuables
ps4a: main.o CircularBuffer.o
	g++ main.o CircularBuffer.o -o ps4a
test: test.o CircularBuffer.o
	g++ test.o CircularBuffer.o -o test $(LBOOST)
#objects
main.o: main.cpp CircularBuffer.cpp
	g++ $(CFLAGS) main.cpp CircularBuffer.cpp
test.o: test.cpp
	g++ $(CFLAGS) test.cpp
CircularBuffer.o: CircularBuffer.cpp CircularBuffer.h
	g++ $(CFLAGS) CircularBuffer.cpp
#implementations
CircularBuffer.cpp: CircularBuffer.h
	g++ $(CFLAGS) CircularBuffer.cpp

#removal
clean:
	rm *.o ps4a test
cleanbk:
	rm *~
superclean:
	rm *~ *.o ps4a test
```

main.cpp Thu Oct 28 13:43:42 2021 1

```
// Copyright 2021 Sandra Hawkins

/*
main.cpp: testing/main program file for CircularBuffer class
10.18.2021

*/

#include <iostream>
#include <chrono>  // NOLINT
#include "CircularBuffer.h"

int main(void) {
 int cap = 10;
 CircularBuffer b(cap);


// loaded lambda, it just checks the time it takes to dequeue and enqueue
// (nanoseconds)
 auto negate = [&]() {
   auto start = std::chrono::steady_clock::now();
   int16_t hold = b.dequeue();
   auto stop = std::chrono::steady_clock::now();
   auto duration = std::chrono::duration_cast<std::chrono::nanoseconds>
   (stop - start);
   std::cout << "d:" <<  duration.count() << "\t";
   hold *= -1;
   start = std::chrono::steady_clock::now();
   b.enqueue(hold);
   stop = std::chrono::steady_clock::now();
   duration = std::chrono::duration_cast<std::chrono::nanoseconds>
   (stop - start);
   std::cout << "e: " << duration.count() << std::endl;
 };


 // printing test: from begin to end, based on FRONT and LAST
 for (int i = 0; i < cap; i++) {b.enqueue(i);}

 std::cout << b << std::endl;
 for (int i = 0; i < b.size(); i++) {
   negate(); std:: cout << b << std::endl << std::endl;
 }
 return 0;
}
```

CircularBuffer.h Sat Sep 25 14:18:17 2021 1

```
// Copyright 2021 Sandra Hawkins

/*
CircularBuffer.h: header file for the CircularBuffer class

*/

#pragma once
#include <stdint.h>
#include <iostream>
#include <vector>
#include <string>

class CircularBuffer {
public:
 explicit CircularBuffer(int capacity);
 int size() { return SIZE; }
 bool isEmpty() { return SIZE == 0;}
 bool isFull() { return SIZE  == cap;}
 void enqueue(int16_t x);
 int16_t dequeue();
 int16_t peek();
 friend std::ostream& operator<< (std::ostream& out, const CircularBuffer&
b);
private:
 std::vector<int16_t> q;  // im not spelling queue
 int cap;
 int SIZE;
 int FRONT;
 int LAST;
};
```

CircularBuffer.cpp   Sat Nov 06 15:09:35 2021   1

```
// Copyright 2021 Sandra Hawkins

/*
CircularBuffer.cpp: implementation file for the CircularBuffer class


10.18.2021

*/

#include <iostream>
#include <algorithm>
#include "CircularBuffer.h"

// lambdas
auto print = [](auto a) { std::cout << a << "\t";};

// error messages
const char* bad_cap =
"CircularBuffer Constructor: capacity must be greater than zero.\n";
const char* full_ring =
"Enqueue: cannot enqueue to a full ring.\n";
const char* empty_ring =
"Dequeue/Peek: there are no elements to properly continue the operation\n";

const int MIN = 1;

// struction
CircularBuffer::CircularBuffer(int capacity) : cap(capacity),
SIZE(0), FRONT(0), LAST(0) {
 if (cap < MIN) { throw std::invalid_argument(bad_cap); }
 q.resize(cap);
}

// operators
std::ostream& operator<<(std::ostream& out, const CircularBuffer& b) {
 // print each element using a generic lambda (in order)
 auto q_begin = b.q.begin() + b.FRONT; auto q_end = q_begin;
 std::for_each(q_begin, b.q.end(), print);
 std::for_each(b.q.begin(), q_end, print);
 return out;
}

void CircularBuffer::enqueue(int16_t x) {
 // push back
 if (SIZE == cap) { throw std::runtime_error(full_ring);}
 ++SIZE; LAST = (LAST + 1) % cap; if (SIZE == MIN) {LAST = FRONT;}
 q[LAST] = x;
}

int16_t CircularBuffer::dequeue() {
 // pop front
 if (!SIZE) { throw std::runtime_error(empty_ring);}
```

CircularBuffer.cpp Sat Nov 06 15:09:35 2021 2

```
 int16_t hold = q[FRONT];
 --SIZE; FRONT = (FRONT + 1) % cap; return hold;
}

int16_t CircularBuffer::peek() {
 if (isEmpty()) {throw std::runtime_error(empty_ring);}
 return q[FRONT];
}
```

test.cpp   Sat Nov 06 17:30:23 2021   53

```
// Copyright 2021 Sandra Hawkins

/*
test.cpp : test program for functionality of CircularBuffer class


10.18.2021

*/

#include <iostream>
#include "CircularBuffer.h"
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include<boost/test/unit_test.hpp>



// exception checking

/*

tests the exception with invalid constructor arguments
as well as NO_THROW if valid

*/
BOOST_AUTO_TEST_CASE(ZeroCapacityException) {
 BOOST_REQUIRE_THROW(CircularBuffer(0), std::invalid_argument);
 BOOST_REQUIRE_THROW(CircularBuffer(-493), std::invalid_argument);
 for (int i = 1; i < 10; i++) {
   BOOST_REQUIRE_NO_THROW(CircularBuffer(static_cast<int>(i)));
 }
}

/*

tests the exception thrown if full.
it shouldnt be thrown if there is still space left in the CircularBuffer

*/
BOOST_AUTO_TEST_CASE(FullException) {
 CircularBuffer b(10); int16_t x = 493;

 // not full
 for (int i = 0; i < 10; i++) {
   BOOST_REQUIRE_NO_THROW(b.enqueue(x));
 }
 // full
 BOOST_REQUIRE_THROW(b.enqueue(x), std::runtime_error);
}


```

```
/*

tests the exception thrown when empty done by dequeue() or peek().
it must not throw when nonempty.

*/
BOOST_AUTO_TEST_CASE(EmptyException) {
 int cap = 2;
 CircularBuffer b(cap); int16_t x = 493;

 // nonempty
 for (int i = 0; i < cap; i++) { b.enqueue(x);}
 for (int i = 0; i < cap; i++) {
   BOOST_REQUIRE_NO_THROW(b.peek());
   BOOST_REQUIRE_NO_THROW(b.dequeue());
 }
 // empty
 BOOST_REQUIRE_THROW(b.dequeue(), std::runtime_error);
 BOOST_REQUIRE_THROW(b.peek(), std::runtime_error);
}



// function testing



/*

tests if empty

*/
BOOST_AUTO_TEST_CASE(isEmptyTrue) {
 CircularBuffer b(2);
 BOOST_REQUIRE(b.isEmpty() == true);
}


/*

tests to see if enqueue is correct.
enqueue changes a lot so other factors change as well
such as the FRONT element (known by peek)

size is incremented
no longer empty (if was previously)
potential to be full

in this case, the capacity is 1, so it is either full or not full,
or empty or nonempty

*/
BOOST_AUTO_TEST_CASE(enqueueCorrect) {
```

test.cpp Sat Nov 06 17:30:23 2021 3

```
 CircularBuffer b(1); int16_t x = 493;
 b.enqueue(x);
 BOOST_REQUIRE(b.peek() == x);
 BOOST_REQUIRE(b.size() == 1);
 BOOST_REQUIRE(b.isEmpty() == false);
 BOOST_REQUIRE(b.isFull() == true);
}

/*
tests if dequeue works properly.
dequeue changes a lot.

capacity is set to 1 such that the Buffer must be either full or not full
so isEmpty() or isFull() is always true.


*/
BOOST_AUTO_TEST_CASE(dequeueCorrect) {
 CircularBuffer b(1); int16_t x = 493, y;
 b.enqueue(x); y = b.dequeue();
 BOOST_REQUIRE(b.size() == 0);
 BOOST_REQUIRE(b.isEmpty() == true);
 BOOST_REQUIRE(b.isFull() == false);
 BOOST_REQUIRE(y == x);
}
```

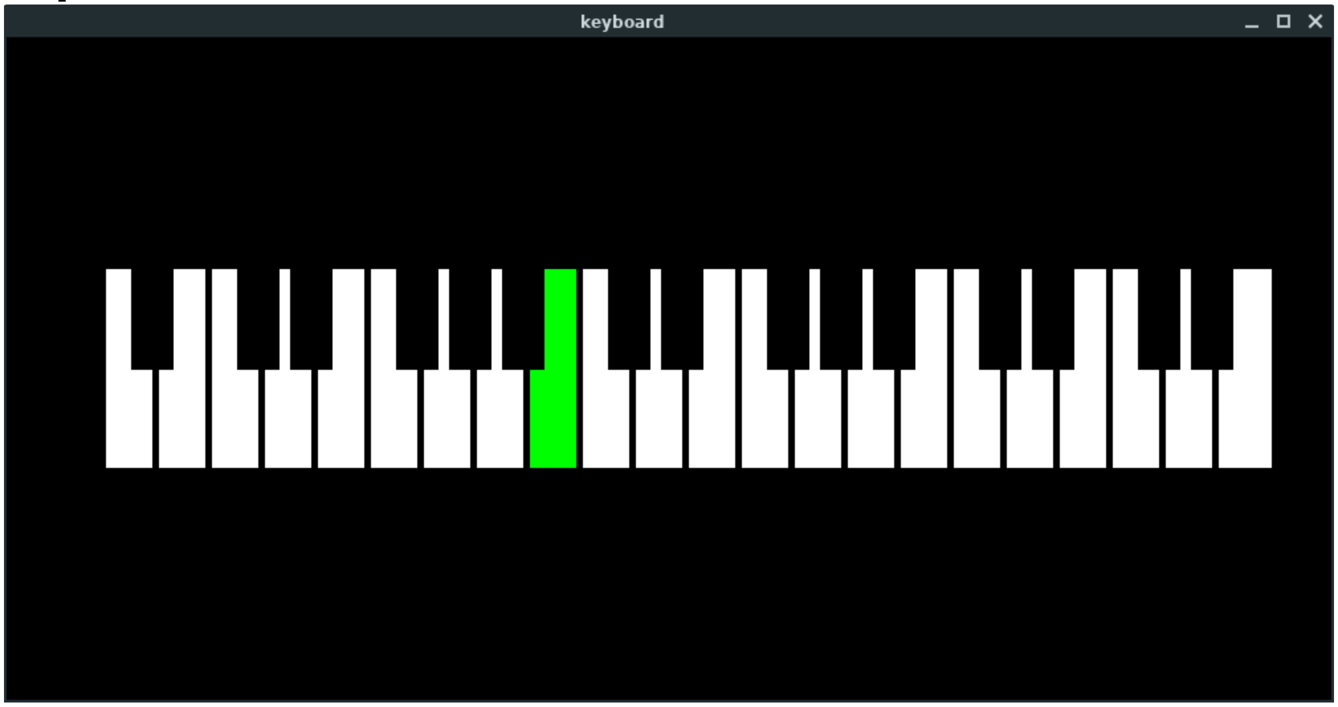
PS4B: SFML Audio Output: StringSound Implementation

Purpose of this assignment was to implement the CircularBuffer class in order to produce a sound-wave (or an imitation of one) through the Karplus-Strong algorithm. The Karplus-Strong simply removed the current front element, and with the next top element, averages them together, and then scales by some decaying factor, and then fed back into the buffer in order to simulate energy loss over time with sound. The implementation of this was done via the class StringSound, which utilised abstraction by keeping all functions and values pertaining to making a sound sample stay within the object. StringSound used 2 functions in order to make a sample: pluck(), which filled the buffer with random values, and tic(), which implemented the karplus-strong algorithm for a single increment of time.

To produce a sound, however, required using parallel vectors of the SFML objects of sf::Int16 in order to contain the samples, sf::SoundBuffer constructed from said samples, to finally have a valid vector of sf::Sound objects that used the buffers. The assignment was challenging for me as it was all very tedious; if one feature did not work perfectly it would all fall apart (no sound is generated if the buffers are bad, and if the samples are bad, then the buffers are bad, etc). I did, however, learn a lot in the assignment: SFML audio functionality, better practice with lambda expressions, as well as learn how to use a randomizing engine to produce random numbers (for pluck() I used a mersene twister).

I also added an additional feature by making a Keyboard object to hold all of the parallel vectors as well as make it drawable. Such that once run, the SFML window would display a keyboard, and when a valid note is played, the corresponding note on the keyboard is colored green.

<u>Ouput:</u>



SFML window titled "keyboard" displaying the drawable Keyboard object. Notes are colored green when they are being played.

Makefile Sun Sep 26 13:39:10 2021 1

```
CFLAGS = -Wall -Werror -pedantic -std=c++14 -c -O1
LFLAGS = -lsfml-window -lsfml-system -lsfml-graphics -lsfml-audio

all: KSGuitarSim

KSGuitarSim: StringSound.o KSGuitarSim.o CircularBuffer.o Keyboard.o
	g++ StringSound.o KSGuitarSim.o CircularBuffer.o Keyboard.o -o KSGuitarSim $(LFLAGS)
KSGuitarSim.o: KSGuitarSim.cpp StringSound.h
	g++ $(CFLAGS) KSGuitarSim.cpp
Keyboard.o: Keyboard.cpp StringSound.h Keyboard.h
	g++ $(CFLAGS) Keyboard.cpp StringSound.cpp
StringSound.o: StringSound.cpp CircularBuffer.cpp
	g++ $(CFLAGS) StringSound.cpp CircularBuffer.cpp
CircularBuffer.o: CircularBuffer.cpp CircularBuffer.h
	g++ $(CFLAGS) CircularBuffer.cpp
StringSound.cpp: StringSound.h
	g++ $(CFLAGS) StringSound.cpp
CircularBuffer.cpp: CircularBuffer.h
	g++ $(CFLAGS) CircularBuffer.cpp
clean:
	rm *.o KSGuitarSim
cleanbk:
	rm *~
superclean:
	rm *.o *~ KSGuitarSim

```

KSGuitarSim.cpp Sun Sep 26 20:19:49 2021 1

```
// Copyright
#include <iostream>
#include <vector>
#include <SFML/Audio.hpp>
#include <SFML/Graphics.hpp>
#include "Keyboard.h"

int main() {
 Keyboard keyboard(10);

 // SFML loop
 sf::RenderWindow window(sf::VideoMode(1000, 500), "keyboard");
 while (window.isOpen()) {
   sf::Event event;
   while (window.pollEvent(event)) {
     if (event.type == sf::Event::Closed) { window.close();}
     if (event.type == sf::Event::KeyPressed) {
       keyboard.play(event.key.code);
     }
     window.clear(); window.draw(keyboard);
     window.display();
   }
 }
 return 0;
}
```

StringSound.h Sun Sep 26 16:39:07 2021 1

```
// Copyright
#pragma once
#include <vector>
#include <SFML/Audio.hpp>
#include "CircularBuffer.h"

const int SAMPLING_RATE = 44100;
class StringSound {
public:
 explicit StringSound(double frequency);  // N (capacity) = samprate/freq
 explicit StringSound(std::vector<sf::Int16> init);
 StringSound(const StringSound& obj) {}
 ~StringSound();
 void pluck();
 void tic();
 sf::Int16 sample() { return _cb->peek();}
 int time() { return _time;}
 friend std::ostream& operator<<(std::ostream& out, const StringSound& s);
private:
 CircularBuffer* _cb;
 int _time;
 int N;
};
```

StringSound.cpp Sat Nov 06 13:31:51 2021 1

```
// Copyright
#include <iostream>
#include <vector>
#include <cmath>
#include "StringSound.h"


// constants + lambdas + other stuff
const double DECAY = 0.996;
auto average = [](double a, double b) { return ((a + b) / 2);};
const char* nonpos = "StringSound constructor: frequency must be positive.";


std::ostream& operator<<(std::ostream& out, const StringSound& s) {
 out << *(s._cb) << std::endl; return out;
}

// struction
StringSound::StringSound(double frequency) : _time(0) {
 if (frequency <= 0) { throw std::invalid_argument(nonpos);}
 N = ceil(SAMPLING_RATE/frequency);
 _cb = new CircularBuffer(N);
}

StringSound::StringSound(std::vector<sf::Int16> init): _time(0) {
 if (init.size() <= 0) { throw std::invalid_argument(nonpos);}
 N = init.size();
 _cb = new CircularBuffer(N);
 for (int i = 0; i < static_cast<int>(init.size()); i++) {
   _cb->enqueue(init[i]);}
}
StringSound::~StringSound() { delete _cb; _cb = nullptr;}


// simulation

/*

simulates the plucking of a guitar string with no time yet passed

fills the buffer with random signed 16 bit integers if not done
so already.
it must be empty prior to this, so empty() may be called.

*/
void StringSound::pluck() {
 if (!_cb->isEmpty()) { _cb->empty(); }
 _cb->fill();
}
/*

simulates time after plucking and how the sound (amplitude)
gradually dissipates via the Karplus-Strong algorithm
*/
```

StringSound.cpp Sat Nov 06 13:31:51 2021 2

```

void StringSound::tic() {
 auto karplus_strong = [&]() {
 double  a = _cb->dequeue(); a = average(a, _cb->peek()) * DECAY;
 _cb->enqueue(a);};
 ++_time; karplus_strong();
}
```

CircularBuffer.h Sun Sep 26 04:48:45 2021 1

```
// Copyright 2021 Sandra Hawkins

/*
CircularBuffer.h: header file for the CircularBuffer class

//10.25.2021: changed vector to hold doubles (to accommodate decimals from
    tic()), instead of int16_t (dont worry, pluck will still use int16_t )
*/

#pragma once
#include <stdint.h>
#include <random>
#include <iostream>
#include <vector>
#include <string>

class CircularBuffer {
public:
 explicit CircularBuffer(int capacity);
 int size() { return SIZE; }
 bool isEmpty() { return SIZE == 0;}
 bool isFull() { return SIZE  == cap;}
 void enqueue(double x);
 void empty();
 void fill();
 double dequeue();
 double peek();
 friend std::ostream& operator<< (std::ostream& out, const CircularBuffer&
b);
private:
 std::vector<double> q;  // im not spelling queue
 int cap;
 int SIZE;
 int FRONT;
 int LAST;
};

```

CircularBuffer.cpp Sat Nov 06 14:40:54 2021 1

```
// Copyright 2021 Sandra Hawkins

/*
CircularBuffer.cpp: implementation file for the CircularBuffer class

10.18.2021

10.24.2021:
        implemented empty() function
10.25.2021:
        implemented fill() function

*/

#include <iostream>
#include <algorithm>
#include "CircularBuffer.h"

// lambdas
auto print = [](auto a) { std::cout << a << "\t";};

// error messages
const char* bad_cap =
"CircularBuffer Constructor: capacity must be greater than zero.\n";
const char* full_ring =
"Enqueue: cannot enqueue to a full ring.\n";
const char* empty_ring =
"Dequeue/Peek: there are no elements to properly continue the operation\n";

const int MIN = 1;

// struction
CircularBuffer::CircularBuffer(int capacity) : cap(capacity),
SIZE(0), FRONT(0), LAST(0) {
 if (cap < MIN) { throw std::invalid_argument(bad_cap); }
 q.resize(cap);
}

// operators
std::ostream& operator<<(std::ostream& out, const CircularBuffer& b) {
 // print each element using a generic lambda (in order)
 auto q_begin = b.q.begin() + b.FRONT; auto q_end = q_begin;
 std::for_each(q_begin, b.q.end(), print);
 std::for_each(b.q.begin(), q_end, print);
 return out;
}

void CircularBuffer::enqueue(double x) {
 // push back
 if (SIZE == cap) { throw std::runtime_error(full_ring);}
 ++SIZE; LAST = (LAST + 1) % cap; if (SIZE == MIN) {LAST = FRONT;}
 q[LAST] = x;
}

```

CircularBuffer.cpp Sat Nov 06 14:40:54 2021 2

```
double CircularBuffer::dequeue() {
 // pop front
 if (!SIZE) { throw std::runtime_error(empty_ring);}
 double hold = q[FRONT];
 --SIZE; FRONT = (FRONT + 1) % cap; return hold;
}

double CircularBuffer::peek() {
 if (isEmpty()) {throw std::runtime_error(empty_ring);}
 return q[FRONT];
}


void CircularBuffer::empty() {
 FRONT = 0; LAST = 0; SIZE = 0;
}

// fill up to capacity with random values
void CircularBuffer::fill() {
 std::random_device rd; std::mt19937 mt(rd());
 std::uniform_int_distribution<int16_t> range(INT16_MIN, INT16_MAX);
 for (auto it = q.begin(); it != q.end(); ++it) {
   enqueue(range(mt));
 }
}
```

Keyboard.h Sun Sep 26 20:33:02 2021 1

```
// Copyright

#pragma once
#include <vector>
#include <SFML/Graphics.hpp>
#include
<SFML/Audio.hpp>
#include "StringSound.h"

/*

constructs a drawable keyboard that plays notes given event.key.type
it is only constructed once.

*/


class Keyboard: public sf::Drawable {
public:
 explicit Keyboard(int seconds);
 Keyboard();
 virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const
;  // NOLINT
 void play(int key_code);
private:
 std::vector<std::vector<sf::Int16>> samples;
 std::vector<sf::SoundBuffer> buffers;
 std::vector<sf::Sound> sounds;
 std::vector<int> notes;
 std::vector<sf::RectangleShape> keys;
};

```

```cpp
// Copyright
#include "Keyboard.h"
#include <string>

const sf::Vector2f START_WHITE = sf::Vector2f(75, 175);
const sf::Vector2f SIZE_WHITE = sf::Vector2f(40, 150);
const sf::Vector2f SIZE_BLACK = sf::Vector2f(30, 75);
const int CHROMATIC = 37;
const int MIDDLE = 110;


auto freq = [](int i) { return MIDDLE * pow(2, (i-12)/24);};
const char* bad_load = "loadFromSamples: failed to load vector to buffer.";
const char* nonpossec = "Keyboard constructor: argument must be positive.";

bool is_black(int i) {
 bool black = false;
 switch (i) {
   case 1: case 4: case 6: case 9: case 11: case 13: case 16: case 18:
   case 21: case 23: case 25: case 28: case 30: case 33: case 35:
   black = true; break;
 }
 return black;
}


std::vector<sf::RectangleShape> make_keys(std::vector<sf::RectangleShape> v) {
 sf::Vector2f posw = START_WHITE; sf::Vector2f posb; int numw = 1;
 for (int i = 0; i < CHROMATIC; i++) {
   if (is_black(i)) {
     v[i].setFillColor(sf::Color::Black); v[i].setSize(SIZE_BLACK);
     posb.y = START_WHITE.y;
     posb.x = (numw * SIZE_WHITE.x) + (SIZE_BLACK.x/2);
     v[i].setPosition(posb); v[i].setOutlineColor(sf::Color::Black);
     v[i].setOutlineThickness(1);
   } else {
     v[i].setOutlineColor(sf::Color::Black); v[i].setOutlineThickness(5);
     v[i].setFillColor(sf::Color::White); v[i].setSize(SIZE_WHITE);
     v[i].setPosition(posw); posw.x += SIZE_WHITE.x; ++numw;
   }
 }
 return v;
}

std::vector<sf::RectangleShape> restore(std::vector<sf::RectangleShape> v) {
 for (int i = 0; i < CHROMATIC; i++) {
   if (!is_black(i) && (v[i].getFillColor() != sf::Color::White)) {
     v[i].setFillColor(sf::Color::White);
   } else if (is_black(i) && (v[i].getFillColor() != sf::Color::Black)) {
     v[i].setFillColor(sf::Color::Black);
   }
 }
 return v;
```

Keyboard.cpp Sun Sep 26 20:31:34 2021 2

```
}
void make_samples(std::vector<sf::Int16>* sample, StringSound* string,
int seconds) {
 for (int i = 0; i < seconds * SAMPLING_RATE; i++) {
   string->tic(); sample->push_back(string->sample());
 }
}

std::vector<int> get_notes(std::vector<int> v) {
 v = {sf::Keyboard::Q, sf::Keyboard::Num2, sf::Keyboard::W,
 sf::Keyboard::E, sf::Keyboard::Num4, sf::Keyboard::R, sf::Keyboard::Num5,
 sf::Keyboard::T, sf::Keyboard::Y, sf::Keyboard::Num7, sf::Keyboard::U,
 sf::Keyboard::Num8, sf::Keyboard::I, sf::Keyboard::Num9, sf::Keyboard::O,
 sf::Keyboard::P, sf::Keyboard::Hyphen, sf::Keyboard::LBracket,
 sf::Keyboard::Equal, sf::Keyboard::Z, sf::Keyboard::X, sf::Keyboard::D,
 sf::Keyboard::C, sf::Keyboard::F, sf::Keyboard::V, sf::Keyboard::G,
 sf::Keyboard::B, sf::Keyboard::N, sf::Keyboard::J, sf::Keyboard::M,
 sf::Keyboard::K, sf::Keyboard::Comma, sf::Keyboard::Period,
 sf::Keyboard::Semicolon, sf::Keyboard::Slash, sf::Keyboard::Quote,
 sf::Keyboard::Space};
 return v;
}


// struction

Keyboard::Keyboard(int seconds) : Drawable() {
 samples.resize(CHROMATIC); buffers.resize(CHROMATIC);
 sounds.resize(CHROMATIC); notes.resize(CHROMATIC);
 keys.resize(CHROMATIC);
 if (seconds <= 0) { throw std::invalid_argument(nonpossec);}
 for (int i = 0; i < CHROMATIC; i++) {
   StringSound string(freq(i)); string.pluck();
   make_samples(&(samples[i]), &string, seconds);
   if (!buffers[i].loadFromSamples(&samples[i][0], samples[i].size(), 2,
   SAMPLING_RATE)) { throw std::runtime_error(bad_load);}
   sounds[i].setBuffer(buffers[i]);
 }
 notes = get_notes(notes);
 keys = make_keys(keys);
}

Keyboard::Keyboard() : Keyboard(1) {}

//  note playing (based on key)
void Keyboard::play(int key_code) {
 keys = restore(keys);
 int i = 0;
 for (i = 0; i < CHROMATIC; i++) {
   if (notes[i] == key_code) { break;}
 }
 if (i >= CHROMATIC) { return;}
 sounds[i].play();
 keys[i].setFillColor(sf::Color::Green);
```

Keyboard.cpp Sun Sep 26 20:31:34 2021 3

```
}

// drawing
void Keyboard::draw(sf::RenderTarget& target, sf::RenderStates states) const
{
 for (int i = 0; i < CHROMATIC; i++) {
   if (is_black(i)) { continue;}
   target.draw(keys[i]);
 }
 for (int i = 0; i < CHROMATIC; i++) {
    if (!is_black(i)) { continue;}
     target.draw(keys[i]);
 }
}
```

PS5: Edit Distance: Finding Optimal Alignment

The purpose of this assignment was to figure out the optimal edit distance between two strings and display the the resulting alignment and cost of each step to the screen. Figuring out the optimal distance was done via the Needleman-Wunsch algorithm, which requires memoization, or keeping track of costs, by filling a matrix. It required additonal traversal after the matrix is filled in order to do the most optimal alignment by going in one of three directions at any given cell : left, right, or diagonal. The Needleman-Munsch is just one of the ways to do so. It requires a lot of space to hold all the values (especially if the input is large).

Needleman-Wunsch was the key algorithm for this assignment, as well as using a vector of a vector of integers of constant size in order to hold the costs for all alignments possible given the input strings. A challenging aspect of the assignment was defiitely figuring out the algorithm. I had to do some additonal research/googling to really understand what the algorithm was doing. Though I also learnt further about space-time complexity for program runtime, as well as how to estimate it through the doubling method. And realizing how very large data sets can have a really, really, really, slow runtime and waste a lot of resources in the process.

Ouptut (input file: endgaps7.txt):

```
Edit distance: 4
Time: 0s
a - 2
t t 0
a a 0
t t 0
t t 0
a a 0
t t 0
- a 2
```

Makefile Thu Oct 28 13:47:26 2021 1

```
CFLAGS = -pedantic -Werror -Wall -std=c++14 -c -O1
LFAGS = -lsfml-system

all: EDistance

EDistance: main.o EDistance.o
	g++ main.o EDistance.o -o EDistance $(LFLAGS)
main.o: main.cpp EDistance.h
	g++ $(CFLAGS) main.cpp
EDistance.o: EDistance.cpp EDistance.h
	g++ $(CFLAGS) EDistance.cpp
EDistance.cpp: EDistance.h
	g++ $(CFLAGS) EDistance.cpp
clean:
	rm *.o EDistance
cleanbk:
	rm *~
superclean:
	rm *~ *.o EDistance
```

main.cpp Sat Nov 06 03:51:17 2021 1

```
// Copyright 11.02.2021 Sandra Hawkins
#include <iostream>
#include <chrono>  // NOLINT, having problems with sf
#include "EDistance.h"

int main() {
 std::string x; std::string y;
 // x = "RELATE"; y = "SENSITIVE";
 std::cin >> x; std::cin >> y;
 std::cout << "Edit distance: ";

 auto start = std::chrono::steady_clock::now();
 EDistance e(x, y);
 std::cout << e.optDistance() << std::endl;
 auto stop = std::chrono::steady_clock::now();
 auto duration = std::chrono::duration_cast<std::chrono::milliseconds>
 (stop -start);
 std::cout << "Time: " << duration.count() << "s" << std::endl;

 std::cout << e.Alignment() << std::endl;
 return 0;
}
```

EDistance.h Fri Nov 05 13:52:23 2021 1

```
// Copyright 11.02.2021 Sandra Hawkins
#pragma once
#include <vector>
#include <algorithm>
#include <string>

class EDistance {
public:
 EDistance(const std::string& x, const std::string& y);
 EDistance();
 int optDistance();
 std::string Alignment();
 friend std::ostream& operator<<(std::ostream& out, const EDistance& ed);
private:
 static int penalty(char a, char b) { return a != b;}
 static int min(int a, int b, int c);
 std::string _x;
 std::string _y;
 std::string _table;
 std::vector<std::vector<int>> _matrix;
 int M;  // num rows
 int N;  // num columns
};


```

EDistance.cpp   Fri Nov 05 15:53:58 2021   1

```
// Copyright 11.02.2021 Sandra Hawkins
#include <iostream>
#include <utility>
#include "EDistance.h"

/*

implemenation file of the EDistance class

*/

// misc
auto cost = [](int entry, int g) { return entry + g;};
auto fill = [](int row, int column) { return 2 * (row - column);};

// static
int EDistance::min(int a, int b, int c) {
 if ( a <= b && a <= c) return a;
 return ( b <= c) ? b : c;
}
std::ostream& operator<<(std::ostream& out, const EDistance& ed) {
 for (int i = 0; i <= ed.M; i++) {
   for (int j = 0; j <= ed.N; j++) { out << ed._matrix[i][j] << " "; }
   out << std::endl;
 }
 return out;
}


// struction
/*

given the input, constructs a matrix (vector of rows) of size
x.size() (N) and y.size() (M).

gives bases for final row and final column

empty constructor assumes empty strings.

*/
const char* empty = "Constructor arguments must be non-empty";
EDistance::EDistance(const std::string& x, const std::string& y):
_x(x), _y(y), _table(""), M(x.length()), N(y.length()) {
 if (x.empty() || y.empty()) {
   throw std::invalid_argument(empty);
 }
 _matrix.resize(M + 1, std::vector<int>(N+1));
 for (int i = 0; i <= M; i++) { _matrix[i][N] = fill(M, i);}
 for (int j = 0; j <= N; j++) { _matrix[M][j] = fill(N, j);}
}

```

EDistance.cpp Fri Nov 05 15:53:58 2021 2

```
EDistance::EDistance() : EDistance("", "") {}

/*

Needleman-Wunsch:

for all entries indexed by i and j  and cost matrix M:
(starting from bottom right)

M(i, j) = minimum:
       M[++i][++j] + S(x[i], y[j]); // diagonal movement, no indel
       M[++i][j] + g; // vertical movement, indel
       M[i][++j] + g; // horizontal movement, indel
where g is the penalty cost

return M[0][0]
*/

int EDistance::optDistance() {
 const int g = 2;  // indel penalty cost
 for (int i = M - 1; i >= 0; i--) {
   for (int j = N - 1; j >= 0; j--) {
     int a = cost(_matrix[i + 1][j + 1], penalty(_x[i], _y[j]));
     int b = cost(_matrix[i][j+1], g);
     int c = cost(_matrix[i + 1][j], g);
     _matrix[i][j] = min(a, b, c);
   }
 }
 return _matrix[0][0];
}


/*
Traversal:
starting from the TOP LEFT, go in the direction with minimum cost:

Diagonal movement: M[++i][++j] +1;
Horizontal movement: M[i][++j] + g;
Vertical movement: M[++i][j] + g;
(where g is the penalty cost)

Non-diagonal movements assume insertion in either _x or _y string
(_x may be smaller in size than _y)

keep track of the operations done to see cost of each.
*/

void make_line(char left, char right, char cost, std::string* str) {
 str->push_back(left); str->push_back(' ');
 str->push_back(right); str->push_back(' ');
 str->push_back(cost); str->push_back('\n');
}
```

EDistance.cpp Fri Nov 05 15:53:58 2021 3

```
std::string EDistance::Alignment() {
 if (!_table.empty()) { return _table;}
 std::string prLine;
 char cost = 'g', left = 'x', right = 'y';
 int diag = 493, vert = 493, horz = 493, m = -493;
 int len = (_x.length() >= _y.length()) ? _x.length() : _y.length();
 for (int i = 0, j = 0, k = 0; k <= len; ++k) {
   cost = '2';
   if (i < M && j < N) {
     diag = _matrix[i+1][j+1] + 1;
     vert = _matrix[i+1][j] + 2;
     horz = _matrix[i][j+1] + 2;
     m = min(diag, vert, horz);
     if (m == diag) {
       cost = '0' + penalty(_x[i], _y[j]); left = _x[i]; right = _y[j];
       ++i; ++j;
     } else if (m == vert) { left = _x[i]; right = '-'; ++i;
     } else if (m == horz) { left = '-'; right = _y[j]; ++j;
     } } else {
       if (i >= M && j < N) { left = '-'; right = _y[j]; ++j;
       } else if (j >= N && i < M) { left = _x[i]; right = '-'; ++i;
       } else { break;}
     }
   make_line(left, right, cost, &prLine);
   _table += prLine; prLine.clear();
 }
 return _table;
}
```

PS6: Random Writer : Markov Models in Text Generation

A markov model essentially lists probabilities of some next event to occur given some current event. By using markov models with characters, (where each instance of a character is an event) one could generate realistic random text imitating (near) natural language from its given input file. The implemenation in order to do this was done via the class RandWriter, which contained functions to generate text, get a random character, and return particular keys and values. The actual models themselves were done with a nested map: by associating a string (kgram) to characters, which would then be associated to integers (frequencies) (so std::map<std::string, std::map<char, int>>), such that once read from an input string, text generation only requires 'hopping' through the character->int maps given a starting k-gram (which generates the maps), and going to a randomly selected character based on its frequency and continuing for the desired length of the text.

Overall for me, this assignment was difficult. I had great difficulty on how to implement the maps, I was even considering using a transition matrix with each column being an ASCII character, more or less a hash-table, but soon realized that would leave a lot of unused space. Working with a partner in an act of pair programming helped to alleviate the difficulty. My programming partner helped me figure out a more optimal solution, and we worked on the assignment together. He wrote the code while I mainly worked as copilot, providing structure and design of the code to follow.

Aside from learning about pair programming, I also learnt about how to use different distributions in order to accurately produce a random character for the text generation (in this case a weighted vector and discrete_distribution<int> alongside a mersene twister engine). One downside, however, is the runtime and space needed to hold the map(s). It fails at large input files (larger than 1 MB). Regardless of that, I was really impressed with how it turned out. It exceeded my expectations.

<u>Output (given kgram length of 10 and 1000 characters to generate):</u>

<u>input file: aesop.txt</u>

```
me provide the wine to pour over you when it is time
to visit their king.  He
was neither wrathful, cruel, nor tyrannical, but just as he was about
to fly off, he made some excuse to send her
home on a visit to her father died, as there was no danger
threatening from a long journey lay down, overcome with me
instantly."  The Dog, wagging his tale and believing what he said,
"if you will soon
let you know that you have done to yourself the trouble," said the
Satyr, "a
fellow who with the help of a pole endeavored to carry out the least
fear you,
nor are you stronger than herself, saying that if the Horseman got
the hare, he rode
```

```
off as fast as he could run.  A neighbor, seeing the preparations for
a universal applause.  A Countryman

A RICH NOBLEMAN once opened the
faggot, took their spades and mattocks and carefully provided nothing
was found, they cheered the actor, and
loaded him with their teeth.  A Fox, who had never been
brought out on a journey, driving his design, she climbed
```

input file: trump-clinton3.txt:

```
TRUMP: No, it's going to take huge cuts in their homes takes
appropriate precautions.

But the leaders -- they're all gone because the next 10 years. Mr.
Trump back into the court, including healthier for kids, including
healthier lunches...

WALLACE: Well, that's because she was born in this topic. You got
about 45 seconds.

TRUMP: ... and admirals, 21 endorsing me, 21 congressional Medal of
Honor recipients. As far as Japan and Germany, and I'm -- not to
single them out, but South Korea, we have 33,000 people a year who
die from guns. I think we need comprehensive background checks, need
to close the gun show loophole. There's about four minutes between
the order, it must be expected of anyone standing on a debate stage
during a general election." Today your daughter, Ivanka, said the
Republicans, and independent experts which said that years ago, three
years ago, Iran is taking over Iraq.

WALLACE: Time.

TRUMP: ... and the health of the economy...

TRUMP: Right.

WALLACE: Please,
```

Makefile   Mon Nov 15 16:20:58 2021   1

```
# By Jae Choi generic recipe
EXE := TextWriter
objs := main.o RandWriter.o
test := test
testobjs := RandWriter.o test.o
SFML-LIBS := -lsfml-system

CXX := g++
CPPFLAGS := -std=c++17 -Wall -Werror -pedantic -g

ifneq ($(V),1)
Q = @
endif

all: $(EXE) test

#all objs has its .o replaced with .d for dependency tracking
deps := $(patsubst %.o,%.d,$(objs))
-include $(deps)
DEPFLAGS = -MMD -MF $(@:.o=.d)

all: $(EXE) test

$(EXE): $(objs)
        @echo "LINKING  $(EXE)"
        $(Q) $(CXX) $(objs) -o $(EXE) $(SFML-LIBS)

.PHONY: test
test: $(testobjs)
        @echo "LINKING  $(test)"
         $(Q) $(CXX) $(testobjs) -o $(test)

%.o: %.cpp
        @echo "CXX      $@"
        $(Q) $(CXX) $(CPPFLAGS) -c $< $(DEPFLAGS)

clean:
        @echo "CLEANING"
        $(Q) rm -f $(objs) $(EXE) $(testobjs) $(test)
```

main.cpp Mon Nov 15 15:28:55 2021 1

```
/**
* @file TextWriter.cpp
* @author Jae Choi       ID: 01998621
* @author Sandra Hawkins ID: 01843958
* @brief
* @version 0.1
* @date 2021-11-10
*
* @copyright Copyright (c) 2021
*
*/
#include <cstdio>
#include <exception>
#include <memory>
#include <string>
#include <algorithm>

#include "RandWriter.h"

int main(int argc, const char* argv[]) {
 if (argc != 3) {
   std::__throw_invalid_argument("Not enough arguments.");
 }
 std::string str; char ch;
 ch = std::char_traits<char>::eof();
 std::getline(std::cin, str, ch);

 int k = atoi(argv[1]);
 int L = atoi(argv[2]);
 RandWriter r(str, k);
 std::string gram = r.get_kgram();
 std::string word = r.generate(gram, L);
 std::cout << word << std::endl;
 std::cout << r << std::endl;
}
```

RandWriter.h   Sat Nov 13 20:08:21 2021   1

```
/**
* @file randomWriter.h
* @author Jae Choi       ID: 01998621
* @author Sandra Hawkins ID: 01843958
* @brief
* @version 0.1
* @date 2021-11-10
*
* @copyright Copyright (c) 2021
*
*/

#pragma once

#include <string>
#include <map>
#include <memory>
#include <iostream>
#include <random>
#include <utility>

using markov_table_t = std::map<std::string, std::map<char, int>>;

class RandWriter {
public:
 RandWriter(const std::string &text, int k);  // text length min k
 RandWriter(RandWriter &rw) = delete;  // NOLINT
 RandWriter() = delete;
 int order_k() { return m_k; }  // order k of Markov model
 int freq(std::string k_gram);  // throw except if gram length < k
 int freq(std::string k_gram, char c);
 char k_Rand(std::string k_gram);
 std::string generate(std::string k_gram, int L);
 std::string get_kgram() { return m_kgram; }
 friend std::ostream& operator<<(std::ostream &o, RandWriter &rw);  // NOLI
NT
private:
 int m_k;
 std::string m_kgram;  // current kgram string
 std::unique_ptr<std::mt19937> m_rmachine;
 markov_table_t m_markov_map;  // has frequency for next chars

 std::string getKG(const std::string &text, int i, int k);
};
```

RandWriter.cpp Mon Nov 15 15:27:55 2021 81

```
/**
* @file randomWriter.cpp
* @author Jae Choi       ID: 01998621
* @author Sandra Hawkins ID: 01843958
* @brief
* @version 0.1
* @date 2021-11-10
*
* @copyright Copyright (c) 2021
*
*/

#include <vector>
#include "RandWriter.h"

const char* small = "length of kgram is smaller than k.";

RandWriter::RandWriter(const std::string &text, int k)
       : m_k(k), m_kgram("") {
 // check for valid arguments.
 if (static_cast<int>(text.size()) < k || k < 0) {
   const char* e = "RandWriter: invalid arguments.";
   throw std::invalid_argument(e);
 }
 // set random machine
 std::random_device rd;
 m_rmachine = std::make_unique<std::mt19937>(rd());

 // go through the text and fill out our markov model map
 std::string kg;
 auto grab_next_char = [&](int index) {
   return text.at((index + k) % text.size());
 };

 for (auto i = 0u; i < text.size(); i++) {
   kg = getKG(text, i, k);
   m_markov_map[kg][grab_next_char(i)]++;
   kg.clear();
 }

 // Find the first kgram string.
 if (m_k >= 0) {
   std::vector<int> kgram_weights;
   for (const auto & m : m_markov_map) {
     kgram_weights.push_back(freq(m.first));
   }
   std::discrete_distribution<int> dist(kgram_weights.begin(),
                                        kgram_weights.end());
   int kg_index = dist(*m_rmachine);
   auto itr = m_markov_map.begin();
   for (auto i = 0; i < kg_index; i++)
     itr++;
```

RandWriter.cpp Mon Nov 15 15:27:55 2021 2

```
    m_kgram = (*itr).first;
  }
}

/**
* @brief Grabs k length kgram string from text. Starting at index i
*
* @param text base pool of text
* @param i starting index
* @param k
* @return std::string kgram substring
*/
std::string RandWriter::getKG(const std::string &text, int i, int k) {
 std::string ret;

 for (auto j = 0; j < k; ++j)
   ret.push_back(text.at([=] {
     return (i + j) % text.size(); }()));

 return ret;
}

/**
* @brief returns frequency of k_gram
* if length of k_gram is less than m_k throws invalid error.
*
* @param k_gram
* @return int
*/
int RandWriter::freq(std::string k_gram) {
 if (static_cast<int>(k_gram.length()) < m_k)
   throw std::invalid_argument(small);

 int ret = 0;
 if (m_markov_map.find(k_gram) != m_markov_map.end())
   for (auto e : m_markov_map[k_gram])
     ret += e.second;

 return ret;
}

int RandWriter::freq(std::string k_gram, char c) {
 if (static_cast<int>(k_gram.length()) < m_k)
   throw std::invalid_argument(small);
 return m_markov_map[k_gram][c];
}

/**
* @brief Returns error if k_gram isn’t k length and if it cannot be found
* on the markov map
*
* @param k_gram
* @return next character
*/
char RandWriter::k_Rand(std::string k_gram) {
 std::vector<int> wei_ve;
 // Check for valid condition
 if (static_cast<int>(k_gram.length()) <  m_k
 ||  m_markov_map.find(k_gram) == m_markov_map.end()) {
   std::invalid_argument e("k_Rand: Invalid k_gram");
   throw e;
 }

 // populate the wei_ve
 for (const auto &val : m_markov_map[k_gram])
   wei_ve.push_back(val.second);

 std::discrete_distribution<int> dist(wei_ve.begin(), wei_ve.end());
 int index = dist(*m_rmachine);
 auto itr = m_markov_map[k_gram].begin();
 for (int i = 0; i < index; i++)
   itr++;
 // Change internally stored kgram.
 if (m_k > 0) {
   m_kgram.erase(0, 1);
   m_kgram.push_back((*itr).first);
 }

 return (*itr).first;
}

/**
* @brief Generate and return string until L length string is created.
*
* @param k_gram
* @param L
* @return std::string
*/
std::string RandWriter::generate(std::string k_gram, int L) {
 std::string ret;
 m_kgram = k_gram;
 for (auto i = 0; i < L; ++i) {
   ret.push_back(k_Rand(m_kgram));
 }
 return ret;
}

/**
* @brief
* overload the stream insertion operator and display
* the internal state of the Markov Model. Print out
* the order, the alphabet, and the frequencies of
* the k-grams and k+1-grams.
*
```

RandWriter.cpp Mon Nov 15 15:27:55 2021 4

```
*/
std::ostream& operator<<(std::ostream &o, RandWriter &rw) {
 o << "kgram: " << rw.m_kgram << " frequency: ";
 o << rw.freq(rw.m_kgram) << std::endl;
 // loop through the map of current kgram and then print.
 for (const auto &itr_ref : rw.m_markov_map[rw.m_kgram]) {
   o << "next possible: " << itr_ref.first << " frequency: ";
   o << itr_ref.second << std::endl;
 }
 return o;
}
```

```
/**
* @file unit_test.cpp
* @author Jae Choi       ID: 01998621
* @author Sandra Hawkins ID: 01843958
* @brief
* @version 0.1
* @date 2021-11-13
*
* @copyright Copyright (c) 2021
*
*/

#include "RandWriter.h"
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>
#include <boost/test/included/unit_test.hpp>

BOOST_AUTO_TEST_CASE(Exception_Tests) {
 const std::string str = "gagggagaggcgagaaa";
 // int k = 3;
 BOOST_REQUIRE_THROW(RandWriter(str, -2),
                         std::invalid_argument);
 BOOST_REQUIRE_THROW(RandWriter("12", 3),
                         std::invalid_argument);
 BOOST_REQUIRE_NO_THROW(RandWriter(str, 3));
}

BOOST_AUTO_TEST_CASE(Functionality_Tests) {
 std::string test_str = "gagggagaggcgagaaa";
 int k = 2;

 RandWriter rw(test_str, k);
 BOOST_REQUIRE_EQUAL(rw.order_k(), k);
 BOOST_REQUIRE_EQUAL(rw.freq("aa", 'a'), 1);
 BOOST_REQUIRE_EQUAL(rw.freq("aa", 'c'), 0);
 BOOST_REQUIRE_EQUAL(rw.freq("aa", 'g'), 1);

 BOOST_REQUIRE_EQUAL(rw.freq("aa"), 2);

 BOOST_REQUIRE_EQUAL(rw.freq("ag", 'a'), 3);
 BOOST_REQUIRE_EQUAL(rw.freq("ag", 'c'), 0);
 BOOST_REQUIRE_EQUAL(rw.freq("ag", 'g'), 2);
 BOOST_REQUIRE_EQUAL(rw.freq("ag"), 5);

 BOOST_REQUIRE_EQUAL(rw.freq("cg", 'a'), 1);
 BOOST_REQUIRE_EQUAL(rw.freq("cg", 'c'), 0);
 BOOST_REQUIRE_EQUAL(rw.freq("cg", 'g'), 0);
 BOOST_REQUIRE_EQUAL(rw.freq("cg"), 1);

 BOOST_REQUIRE_EQUAL(rw.freq("ga", 'a'), 1);
 BOOST_REQUIRE_EQUAL(rw.freq("ga", 'c'), 0);
```

test.cpp Sat Nov 13 21:49:35 2021 2

```
 BOOST_REQUIRE_EQUAL(rw.freq("ga", ’g’), 4);
 BOOST_REQUIRE_EQUAL(rw.freq("ga"), 5);

 BOOST_REQUIRE_EQUAL(rw.freq("gc", ’a’), 0);
 BOOST_REQUIRE_EQUAL(rw.freq("gc", ’c’), 0);
 BOOST_REQUIRE_EQUAL(rw.freq("gc", ’g’), 1);
 BOOST_REQUIRE_EQUAL(rw.freq("gc"), 1);

 BOOST_REQUIRE_EQUAL(rw.freq("gg", ’c’), 1);
 BOOST_REQUIRE_EQUAL(rw.freq("gg", ’g’), 1);
 BOOST_REQUIRE_EQUAL(rw.freq("gg", ’g’), 1);
 BOOST_REQUIRE_EQUAL(rw.freq("gg"), 3);
}
```

PS7: Introduction to Regular Expression Parsing

Given .log files, the goal was to produce a report detailing the number of (un)successful boots for a Kronos Intouch device. Regular expressions had to be used, namely those belonging to a boost library.

The exact solution/implementation method/guidelines were not wholly specified. I thought it best to implement it as a class in order to hold all of the variables, functions, and other data within a single object. Helps with the abstraction. My class was called Boots, where its only purpose was to record and keep track of boots of the device. Regex was used in order to parse the lines in the .log file searching for the correct expression, as well as finding the timestamp of when it occurred once the correct expression is found. The lines to be written for the report were held in parallel vectors: one to hold the strings of boot initiation and another to hold the strings of boot completion. Other items were kept track, such as number of successes, lines read, and input file (file i/o was done with <fstream>). Components of each line of the report were concatenated together.

Out of all the assignments, this one took me the longest. Mainly because in the beginning my regex expressions were not entirely correct so they had to all be tweaked ever so slightly. I also learnt about regex and how to use regex using a boost library, and realizing how useful and powerful they are.

Output (output file: device2_intouch.log.rpt):
```
Device Boot Report

InTouch log file: device2_intouch.log
Lines Scanned: 538352 Device boot count:

initiated = 1, completed: 1



=== Device Boot ===
498921(device2_intouch.log): 2014-03-11 15:42:26 Boot Start
499030(device2_intouch.log): 2014-03-11 15:45:08 Boot Completed
     Boot time: 162000ms
```

Makefile Tue Nov 23 19:04:07 2021 1

```
CFLAGS = -pedantic -Wall -Werror -c -std=c++14 -O1
LFLAGS = -lboost_regex -lboost_date_time

all: ps7

ps7: Boots.o main.o
	g++ Boots.o main.o -o ps7 $(LFLAGS)
Boots.o: Boots.cpp Boots.h
	g++ $(CFLAGS) Boots.cpp
main.cpp: Boots.h
	g++ $(CFLAGS) main.cpp
Boots.cpp:
	g++ $(CFLAGS) Boots.cpp
clean:
	rm *.o ps7
cleanbk:
	rm *~
superclean:
	rm *.o *~ ps7

```

main.cpp Tue Nov 23 12:37:11 2021 1

```
// Copyright 2021.11.21 Sandra Hawkins
#include <iostream>
#include <fstream>
#include "Boots.h"


const char* command = "Insufficient command line arguments\n";
const char* bad_file = "Invalid file to read from\n";
int main(int argc, const char* argv[]) {
 if (argc != 2) { throw std::runtime_error(command);}
 std::string arg(argv[1]); 12:
 boost::regex expr{"device[1-6]_intouch.log"};
 if (!boost::regex_match(arg, expr)) {
   throw std::invalid_argument(bad_file);
 }

 Boots b(arg); b.generate();
 return 0;
}
```

Boots.h   Tue Nov 23 12:04:07 2021   1

```
// Copyright 2021.11.21 Sandra Hawkins

#pragma once
#include <iostream>
#include <string>
#include <vector>
#include <boost/regex.hpp>



/*
Class to keep track (and store) number of boots from a stream.

*/
class Boots {
private:
 static boost::regex _boot_start;
  static boost::regex _boot_success;
 static boost::regex _timestamp;
public:
 explicit Boots(std::string filename);
 Boots();
 int lines_scanned() const { return _line;}
 int total_boots() const { return _start.size();}
 int total_success() const { return _nsuccess;}
 std::string filename() const { return _filename;}
 void generate();
private:
 int _line;
 int _nsuccess;
 std::vector<std::string> _start;
 std::vector<std::string> _success;
 std::string _filename;
 void print();
};
```

Boots.cpp Tue Nov 23 13:24:23 2021 91

```cpp
// Copyright 2021.11.21 Sandra Hawkins
#include "Boots.h"
#include <sstream>
#include <utility>
#include <fstream>
#include "boost/date_time/gregorian/gregorian.hpp"
#include "boost/date_time/posix_time/posix_time.hpp"

// constants, lambda, etc

// exceptions
const char* stream = "Constructor: invalid file stream";
const char* file_fail = "Generate/print: Failed to open file for stream";

// report
const char* START = "Boot Start\n";
const char* COMPLETE = "Boot Completed\n";
const char* BOOT = "=== Device Boot ===\n";
const char* FAILURE = "**** Incomplete Boot ****\n\n";
const char* TIME = "\tBoot time: ";

boost::regex Boots::_boot_start{"(\\(log.c.166\\) server started)"};
boost::regex Boots::_boot_success{"oejs.AbstractConnector:Started SelectChan nelConnector@0.0.0.0:9080"};  // NOLINT
boost::regex Boots::_timestamp{"\\d{4}-\\d{2}-\\d{2}\\s\\d{2}:\\d{2}:\\d{2}"};

enum {q0, q1, NUM_STATE};  // states differentiated by regex expression
/*
the state is determined by previous regex found, by default it is
at q1 (start and complete  statements found).

if the start statement is found while remaining in q0, a pairing
complete statement was never found.
*/


//  number->string via stringstream
auto string = [](auto a) {
 std::string str;
 std::stringstream ss; ss << a; ss >> str;
 return str;
};

// provides a header/introduction to boot report
std::string header(const Boots& b) {
 std::string str;
 str += "Device Boot Report\n\n";
 str += ("InTouch log file: " + b.filename() + "\n");
 str += ("Lines Scanned: " + string(b.lines_scanned()) + "\n\n");
 str += "Device boot count: ";
 str += ("initiated = " + string(b.total_boots()) + ", ");
```

Boots.cpp Tue Nov 23 13:24:23 2021 2

```
 str += ("completed: " + string(b.total_success()) + "\n\n\n");
 return str;
}

// returns time elpased, given 2 strings (timestamps).
auto time_elapsed = [](std::string a, std::string b) {
 using boost::gregorian::date;
 using boost::posix_time::ptime;
 using boost::posix_time::time_duration;
 using boost::posix_time::duration_from_string;

 date d1(boost::gregorian::from_simple_string(a));
 date d2(boost::gregorian::from_simple_string(b));

 boost::regex expr{"\\d{2}:\\d{2}:\\d{2}"}; boost::smatch what;

 boost::regex_search(a, what, expr);
 ptime t1(d1, time_duration(duration_from_string(what[0])));

 boost::regex_search(b, what, expr);
 ptime t2(d2, time_duration(duration_from_string(what[0])));

 boost::posix_time::time_duration td = t2 - t1;
 return td.total_milliseconds();
};


// concatenation lambda, concatenates elements into 1 string for a line
// of the report. n: line_number, f: filename, t:timestamp
auto concat = [](int n, std::string f, std::string t, const int state) {
 std::string status = START;
 if (state == q1) { status = COMPLETE;}
 return (string(n) + "(" + f + "): " + t + " " + status);
};



/*
Constructs the Boots object, it must have a string paramter.
*/

// struction
Boots::Boots(std::string filename) : _line(0), _nsuccess(0),
_filename(filename)
{ if (filename.empty()) { throw std::invalid_argument(stream);}}
Boots::Boots() : Boots("") { _filename.clear();}

/*
Goes through the given file stream and finds the
corresponding strings, given regular expressions.
if no file stream is given, std::cin is used.

```

Boots.cpp Tue Nov 23 13:24:23 2021 3

```
The resulting string element for any of the parallel vectors is
a concatenation of
-line number where encountered
-filename
-timestamp
-boot status
-time elapsed (if complete)
and additional formatting (newline appended)

by going through the entire file (until extraction failure) stream and
analysing each line for matching expressions. line by line.

*/
void Boots::generate() {
 std::string str; int state = q1;
 std::string t1; std::string t2;

 std::ifstream in; in.open(_filename);
 if (in.fail()) { throw std::runtime_error(file_fail);}

 while (!in.fail()) {
   std::getline(in, str, '\n'); if (in.eof()) { break;}
   boost::smatch what; _line++;
   if (boost::regex_search(str, what, _boot_start)) {
     if (!state) { _success.push_back(""); }

     boost::regex_search(str, what, _timestamp);
     t1 = what[0]; state = q0;
     _start.push_back(concat(_line, _filename, t1, state));

   } else if (boost::regex_search(str, what, _boot_success)) {
       boost::regex_search(str, what, _timestamp); t2 = what[0];
       state = q1; _nsuccess++;

       std::string t = TIME + string(time_elapsed(t1, t2))  + "ms\n\n";
       _success.push_back(concat(_line, _filename, t2, state) + t);

       t1.clear(); t2.clear();
   }
   str.clear();
 }
 in.close(); print();
}


// overload
/*
prints report generated to the filestream of [filename].rpt

prints:

-header/introduction to report (and additional parameters)
```

Boots.cpp Tue Nov 23 13:24:23 2021 4

```
-elements from the corresponding parallel vectors (concatenated strings)

*/
void Boots::print() {
 std::ofstream out; out.open(_filename + ".rpt");
 if (out.fail()) { throw std::runtime_error(file_fail);}
 out << header(*this) << std::endl;
 for (int i = 0; i < static_cast<int>(_start.size()); i++) {
   std::string str = (_success[i].empty()) ? FAILURE : _success[i];
   out << BOOT << _start[i] << str;
 }
 out.close();
}

```