Computing IV Section number: 201: Project Portfolio

Marvens Luc

12/14/2024

Contents

Time to Complete Portfolio: time to complet: 5 days

1 ps0: Hello SFML

1.1 Overview

Ps0 is a project that tests if you have set up sfml correctly. When it's set up you have to copy a line of code that displays a green circle. In the second part of the project, I have to set up and display a sprite. The sprite has to move based on a keystrokes.

1.2 End Product

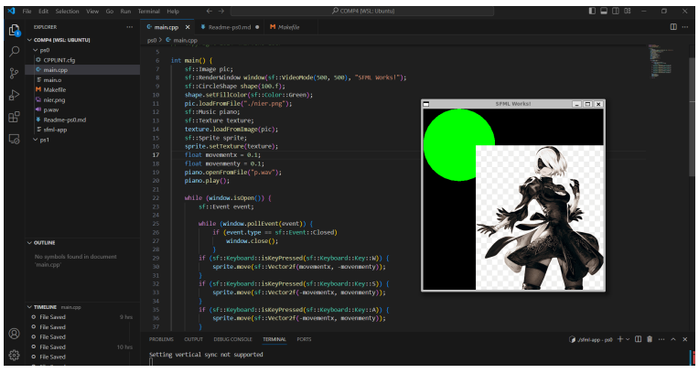

Figure 1: Ps0 output

1.3 What I accomplished

I made a makefile that complied and built my program. I made a main that draws a sprite. I made a sprite that loaded my image. The image can move left, right, up, and down based on keystrokes—I set up music that allows you to play music when the code complies.

1.4 What I already knew

I knew a little bit of SFML from taking computing three. I knew how to set up and compile the green circle. I knew how to modify the circle to grow or change colors.

1.5 What I learend

I learned how to make and compile a makefile. I learned how to load and display my own sprite. I learned how to add functionality like music and movement. My movements are based on the keystrokes up, down, right, and left.

1.6 Challenges

This project was easy and straightforward. I had some issues with the audio, my audio didn't work when I complied my code. I found out that's a sfml issue.

1.7 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++20 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework
# Your .hpp files
DEPS =
# Your compiled .o files
OBJECTS =
# The name of your program
PROGRAM = sfml-app

.PHONY: all clean lint


all: $(PROGRAM)

# Wildcard recipe to make .o files from corresponding .cpp file
%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<

$(PROGRAM): main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o $(PROGRAM)

lint:
    cpplint *.cpp *.hpp
```

main.cpp:

```
#include <iostream>
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
// "Copyright 2024 <Marvens Luc>"

int main() {
    sf::Image pic;
    sf::RenderWindow window(sf::VideoMode(500, 500), "SFML Works!");
    sf::CircleShape shape(100.f);
    shape.setFillColor(sf::Color::Green);
    pic.loadFromFile("./sprite.png");
    sf::Music piano;
    sf::Texture texture;
    texture.loadFromImage(pic);
    sf::Sprite sprite;
    sprite.setTexture(texture);
    float movementx = 0.1;
    float movenmenty = 0.1;
    piano.openFromFile("p.wav");
    piano.play();
    // this code runs the window
    while (window.isOpen()) {
        sf::Event event;

        while (window.pollEvent(event)) {
```

```
26              if (event.type == sf::Event::Closed)
27                  window.close();
28              }
29              // basic movement with the keys
30          if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::W)) {
31              sprite.move(sf::Vector2f(movementx, -movenmenty));
32          }
33          if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::S)) {
34              sprite.move(sf::Vector2f(-movementx, movenmenty));
35          }
36          if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::A)) {
37              sprite.move(sf::Vector2f(-movementx, movenmenty));
38          }
39          if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::D)) {
40              sprite.move(sf::Vector2f(movementx, movenmenty));
41          }
42          if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::F)) {
43              sprite.setRotation(90.f);  // absolute angle
44          }
45           // clears the window
46          window.clear();
47          // draws the shape and sprite display them
48          window.draw(shape);
49          window.draw(sprite);
50          window.display();
51      }
52          return 0;
53      }
```

2 ps1: Linear Feedback Shift Register & PhotoMagic

2.1 Overview

In ps1 we are supposed to make an image scrambler using LSFR(Linear Feedback Shift). An LSFR is a bit shift operation using the XOR operator. The LSFR result is based on three parameters, n (the number of bits), the initial seeds(The sequence of bits), and tap(the bit that you want to shift).

2.2 End Product

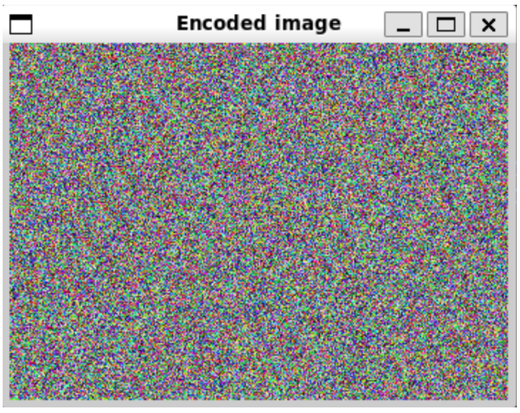

Figure 2: Encryption

Figure 3: Decryption

2.3 Design and Implementation

The assignment is separated into two parts, a and b. In part A, I overloaded the insertion operator to print out the seed in the current register. I set up the constructor to initialize our seed. I overloaded my extraction operator to read from the stream. In part b, I implemented the step function, which basically does a left-shift operation and returns a new character. Generate calls in steps several times and return an integer based on the result the step produces. The data structure I used is an array because strings are characters That are contiguous in memory; to access a bit, you need to specify an index. For the LSFR, we have to do three taps, which are positions 13, 12, and 11. The other data structure I used is a vector. I used a vector to iterate through the image pixel by pixel. I implemented a series of tests for ps1. I made three tests for step, which tested different combinations of the seeds. I tested the step to see if it returned the correct bit after the left shift operation. The last test I implemented was Generate; I just checked if Generate returned the right integer after calling step k for an amount of time.

2.4 What I learend

I learned how to use the boost test library. I learned how to actually make test for my code. I learned about LSFR and how to use it to scramble a image. I learned about how to use color in sfml.

2.5 Challenges

One challenge I faced was figuring out how to implement the transform function. I wasn't familiar with vectors, so I tried to do it with an array, which was inefficient.

2.6 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++20 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework
# Your .hpp files
DEPS = FibLFSR.hpp PhotoMagic.hpp
# Your compiled .o files
OBJECTS = FibLFSR.o PhotoMagic.o
# The name of your program
PROGRAM = PhotoMagic

TESTPROGRAM = test

STATIC_LIB = PhotoMagic.a

.PHONY: all clean lint


all: $(PROGRAM) $(STATIC_LIB) $(TESTPROGRAM)

# Wildcard recipe to make .o files from corresponding .cpp file
%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<

$(STATIC_LIB): $(OBJECTS)
    ar rcs $(STATIC_LIB) $@ $^

$(PROGRAM): main.o $(OBJECTS) $(STATIC_LIB)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

$(TESTPROGRAM): test.o $(OBJECTS) $(STATIC_LIB)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o $(PROGRAM) PhotoMagic.a test main

lint:
    cpplint *.cpp *.hpp
```

PhotoMagic.hpp:

```
// Copyright 2024 Marvens Luc
#pragma once
#include <SFML/Graphics.hpp>
#include "FibLFSR.hpp"
namespace PhotoMagic {
    void transform(sf::Image& img, FibLFSR* lfsr);
    void doubledisplay(sf::Image& input, sf::Image& output);
}
```

PhotoMagic.cpp:

```
// Copyright 2024 Marvens Luc
#include "PhotoMagic.hpp"
#include <iostream>
namespace PhotoMagic {
```

```
void transform(sf::Image& img, FibLFSR* lfsr) {
    sf::Vector2u size = img.getSize();  // 2 vector with length and width
    for (unsigned int y = 0; y < size.y; y++) {
        for (unsigned int x = 0; x < size.x; x++) {
            sf::Color pixel = img.getPixel(x, y);
            pixel.r = pixel.r ^ lfsr->generate(10);
            pixel.g = pixel.g ^ lfsr->generate(10);
            pixel.b = pixel.b ^ lfsr->generate(10);
            img.setPixel(x, y, pixel);
        }
    }
}

void doubledisplay(sf::Image& input, sf::Image& output) {
    sf::Vector2u size1 = input.getSize();
    sf::Vector2u size2 = input.getSize();
    sf::RenderWindow window1(sf::VideoMode(size1.x, size1.y), "Encoded image
    ");
    sf::RenderWindow window2(sf::VideoMode(size2.x, size2.y), "orginal image
    ");
    sf::Texture otexture;
    if (!otexture.loadFromImage(input)) {
        std::cout << "load from failed" << std::endl;
    }
    sf::Texture ntexture;
    if (!ntexture.loadFromImage(output)) {
        std::cout << "load from failed" << std::endl;
    }
    sf::Sprite sprite1;
    sf::Sprite sprite2;
    sprite1.setTexture(otexture);
    sprite2.setTexture(ntexture);
    while (window1.isOpen() && window2.isOpen()) {
        sf::Event event;
        while (window1.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window1.close();
        }
        while (window2.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window2.close();
        }
        window1.clear();
        window1.draw(sprite1);
        window1.display();
        window2.clear();
        window2.draw(sprite2);
        window2.display();
    }
}
}  // namespace PhotoMagic
```

FibLFSR.hpp:

```
#pragma once
#include <iostream>
#include <string>
// Copyright 2024 Marvens Luc

namespace PhotoMagic {
```

```
class FibLFSR {
 public:
    explicit FibLFSR(const std::string& seed);
    ~FibLFSR() {}
    int step();
    int generate(int k);
    friend std::ostream& operator<<(std::ostream& out, FibLFSR& lfsr);
 private:
    std::string sed;
};
  // std::ostream& operator<<(std::ostream&, const FibLFSR& lfsr);
}  // namespace PhotoMagic
```

FibLFSR.cpp:

```
// Copyright 2024 Marvens Luc
#include "FibLFSR.hpp"
#include <iostream>
namespace PhotoMagic {
int PhotoMagic::FibLFSR::generate(int k) {
    int temp = 0;
    for (int i = 0; i < k; i++) {
        temp = (temp * 2) + step();
    }
    return temp;
}

int PhotoMagic::FibLFSR::step() {  // return the bit the was newly generated
    int new_bit;
    std::string temp;
    new_bit = (sed[0] ^ sed[2] ^ sed[3] ^ sed[5]);
    temp = sed.substr(1, 15);
    sed = temp;
    sed += std::to_string(new_bit);
    return new_bit;
}

PhotoMagic::FibLFSR::FibLFSR(const std::string& seed) {  // check for a
   valid string
    sed = seed;
}

std::ostream& operator<<(std::ostream& out, FibLFSR& lfsr) {
    out << lfsr.sed;
    return out;
}


}  // namespace PhotoMagic
```

main.cpp:

```
// Copyright 2024 Marvens Luc
#include <ostream>
#include <string>
#include "FibLFSR.hpp"
#include "PhotoMagic.hpp"
using PhotoMagic::FibLFSR;

int main(int argc, char* argv[]) {
```

```
    std::string inputfile;
    std::string outputFile;
    std::string seeed;
    std::cin >> inputfile >> outputFile >> seeed;
    FibLFSR zed(seeed);
    sf::Image input;
    if (!input.loadFromFile(inputfile)) {
        std::cout << "Unable to open" << std::endl;
    }
    PhotoMagic::transform(input, &zed);
    std::cout << zed << std::endl;
    sf::Image output;
    if (!output.loadFromFile(inputfile)) {
        std::cout << "Unable to open" << std::endl;
    }
    if (!output.saveToFile(outputFile)) {
        return -1;
    }

    if (seeed.empty()) {
        std::cerr << "this is the string " << seeed << "Error: LFSR seed is missing or empty."
                  << std::endl;
        return 1;
    }
    PhotoMagic::doubledisplay(input, output);
}
```

test.cpp:

```
// Copyright 2022
// By Dr. Rykalova
// Editted by Dr. Daly
// test.cpp for PS1a
// updated 9/20/2024

#include <iostream>
#include <sstream>
#include <string>
#include <SFML/Graphics.hpp>
#include "FibLFSR.hpp"
#include "PhotoMagic.hpp"
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>


using PhotoMagic::FibLFSR;

BOOST_AUTO_TEST_CASE(testStepInstr) {
    FibLFSR l("1011011000110110");
    BOOST_REQUIRE_EQUAL(l.step(), 0);
    BOOST_REQUIRE_EQUAL(l.step(), 0);
    BOOST_REQUIRE_EQUAL(l.step(), 0);
    BOOST_REQUIRE_EQUAL(l.step(), 1);
    BOOST_REQUIRE_EQUAL(l.step(), 1);
    BOOST_REQUIRE_EQUAL(l.step(), 0);
    BOOST_REQUIRE_EQUAL(l.step(), 0);
    BOOST_REQUIRE_EQUAL(l.step(), 1);
}
```

```
31
32 BOOST_AUTO_TEST_CASE(testGenerateInstr) {
33     FibLFSR l("1011011000110110");
34     BOOST_REQUIRE_EQUAL(l.generate(9), 51);
35 }
36
37 BOOST_AUTO_TEST_CASE(testGenerateInst) {  // checking gen with a neg input
38     FibLFSR l("1011011000110110");
39     BOOST_REQUIRE_EQUAL(l.generate(-9), 0);
40 }
41
42
43 BOOST_AUTO_TEST_CASE(whatifstepis0) {  // checking step again
44     FibLFSR l("1011011000110110");
45     BOOST_REQUIRE_EQUAL(l.step(), 0);
46 }
47
48
49 BOOST_AUTO_TEST_CASE(Checkstepandgen) {  // checking step and generate
       together
50     FibLFSR l("1111111111111111");
51     BOOST_REQUIRE_EQUAL(l.step(), 0);
52     BOOST_REQUIRE_EQUAL(l.step(), 0);
53     BOOST_REQUIRE_EQUAL(l.step(), 0);
54     BOOST_REQUIRE_EQUAL(l.step(), 0);
55     BOOST_REQUIRE_EQUAL(l.step(), 0);
56
57     BOOST_REQUIRE_EQUAL(l.generate(5), 0);
58 }
59
60
61 BOOST_AUTO_TEST_CASE(Justtestinstep) {  // Check step is the string is all
       1's
62     FibLFSR l("1111111111111111");
63     BOOST_REQUIRE_EQUAL(l.step(), 0);
64     BOOST_REQUIRE_EQUAL(l.step(), 0);
65     BOOST_REQUIRE_EQUAL(l.step(), 0);
66     BOOST_REQUIRE_EQUAL(l.step(), 0);
67 }
68
69
70 BOOST_AUTO_TEST_CASE(TestingStringlenght) {  // checking if string is Not
       equal
71     FibLFSR l("10110110001101");
72     std::stringstream testing;
73     testing << l;
74     BOOST_REQUIRE_NE(testing.str(), "1011011000110110");
75 }
76
77 BOOST_AUTO_TEST_CASE(Checkinginsertion) {  // checking if string is equal
78     FibLFSR insert("1010101010101010");
79     std::stringstream newinsert;
80     newinsert << insert;
81     BOOST_REQUIRE_EQUAL(newinsert.str(), "1010101010101010");
82 }
83
84
85 BOOST_AUTO_TEST_CASE(checkingallzeros) {
86     FibLFSR l("0000000000000000");
```

```
87      BOOST_REQUIRE_EQUAL(l.step(), 0);
88      BOOST_REQUIRE_EQUAL(l.step(), 0);
89      BOOST_REQUIRE_EQUAL(l.step(), 0);
90      BOOST_REQUIRE_EQUAL(l.step(), 0);
91      BOOST_REQUIRE_EQUAL(l.step(), 0);
92  }
93
94  BOOST_AUTO_TEST_CASE(Alternating) {
95      FibLFSR l("1010101110101011");
96      BOOST_REQUIRE_EQUAL(l.step(), 0);
97      BOOST_REQUIRE_EQUAL(l.step(), 0);
98      BOOST_REQUIRE_EQUAL(l.generate(2), 2);
99  }
100
101 BOOST_AUTO_TEST_CASE(Alternatingl) {
102     FibLFSR l("1010101110101011");
103     BOOST_REQUIRE_EQUAL(l.step(), 0);
104     BOOST_REQUIRE_EQUAL(l.step(), 0);
105     BOOST_REQUIRE_EQUAL(l.generate(2), 2);
106     FibLFSR M("1110101110111111");
107     BOOST_REQUIRE_EQUAL(M.step(), 0);
108     BOOST_REQUIRE_EQUAL(M.generate(1), 1);
109 }
110
111
112 BOOST_AUTO_TEST_CASE(letstransformation) {
113     sf::Image original, genImage;
114
115     if (!original.loadFromFile("./cat.jpg"))
116         std::cerr << "Original image is empty or not loaded correctly!" <<
    std::endl;
117
118     if (!genImage.loadFromFile("./cat.jpg"))
119         std::cerr << "genimage image is empty or not loaded correctly!" <<
    std::endl;
120
121     BOOST_REQUIRE(original.getPixel(4, 5) == genImage.getPixel(4, 5));
122
123     FibLFSR encKey("1111111111111111");
124     transform(genImage, &encKey);
125     BOOST_REQUIRE(original.getPixel(4, 5) != genImage.getPixel(4, 5));
126
127     FibLFSR decKey("1111111111111111");
128     transform(genImage, &decKey);
129     BOOST_REQUIRE(original.getPixel(4, 5) == genImage.getPixel(4, 5));
130 }
```

3 ps2: pentaflake

3.1 Overview

In project 2, I was assigned to make a program that displays a pentaflake. The pentaflake is a series of small pentagon join together recursively making a snowflake but with five hexagon each side.

3.2 End Product

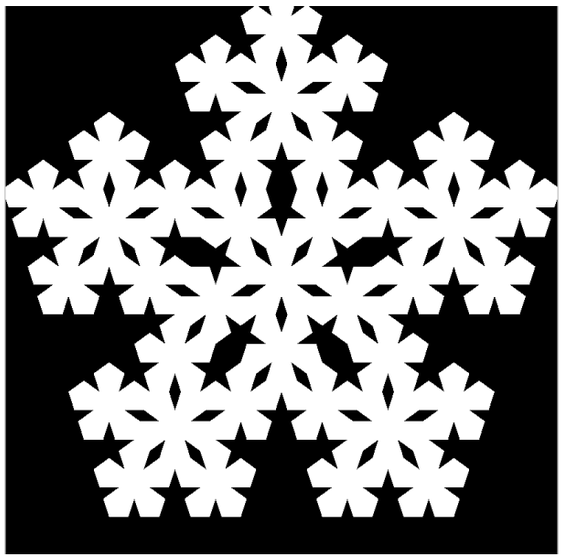

Figure 4: Ps2 output

3.3 Design and Implementation

I designed a function to draw the pentaflake based on its pos, depth, size, and initial angle. The draw pentaflake will be the most essential function in my main. For this particular project, the prerequisite is knowing basic trigonometry. The pentaflake is supposed to be drawn recursively based on the length(the side of the base pentagon) and depth, which is the number of recursions. We had to make an equation that took our base pentagon and drew one pentagon per side, and it would repeat the same process n times.

3.4 What I learend

I learned that Sfml can't draw concave shapes, so I used the convex shape class to define a hexagon. I also learned another application of recursion.

3.5 Challenges

The most challenging part was the math. It was difficult to make an equation that draws the hexagon recursively, starting with the base hexagon we drew.

3.6 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++20 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework
# Your .hpp files
DEPS = penta.hpp
# Your compiled .o files
OBJECTS = penta.cpp
# The name of your program
PROGRAM = Penta

.PHONY: all clean lint


all: $(PROGRAM)

# Wildcard recipe to make .o files from corresponding .cpp file
%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<

$(PROGRAM): main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm -f *.o $(PROGRAM)

lint:
    cpplint *.cpp *.hpp
```

pentaflake.hpp:

```
#pragma once
#include <SFML/Graphics.hpp>
// Copyright 2024 Marvens Luc
void drawpentagon(sf::RenderTarget &window, sf::Vector2f pos, int depth,
                  double size, double intial_angle);
```

pentaflake.cpp:

```
// Copyright 2024 Marvens Luc
#include "penta.hpp"
#include <cmath>
#include <iostream>
const float RadPerDeg = M_PI / 180;
const float golden_ration = (1 + sqrt(5)) / 2;
// Function to draw a pentagon with recursion
void drawpentagon(sf::RenderTarget &window, sf::Vector2f pos, int depth,
                  double size, double initial_angle) {
  // Calculate the circumradius of the pentagon
  if (depth > 0) {
    float _size = size / (1 + golden_ration);
    float inner_radius = _size / (2 * tan(36 * RadPerDeg));
    // other five pentagons around the center one
    for (size_t i = 0; i < 5; i++) {
      std::cout << "the outer angle at i: " << i << std::endl;
      float thetadegre = 72 * i + initial_angle;
```

```
      float thetaradian = thetadegre * RadPerDeg;
      float d = 2 * inner_radius;
      sf::Vector2f new_pos(
          pos.x + d * sin(thetaradian),   // X position 2*radius*cos0
          pos.y - d * cos(thetaradian));
      drawpentagon(window, new_pos, depth - 1, _size, initial_angle);
    }
    drawpentagon(window, pos, depth - 1, _size, initial_angle + 180);
  } else {
    float radius = size / (2 * sin(36 * RadPerDeg));   // Correct formula
    sf::CircleShape pentagon(radius, 5);
    pentagon.setOrigin(radius, radius);    // Origin in the center of the
   pentagon
    pentagon.setPosition(pos);             // Center of pentagon at pos
    pentagon.setRotation(initial_angle);   // Apply initial rotation
    pentagon.setFillColor(sf::Color::White);
    window.draw(pentagon);
  }
}
```

main.cpp:

```
// Copyright 2024 Marvens Luc
#include<iostream>
#include "penta.hpp"
int main(int argc, const char* argv[]) {
int n = 3;
if (argc > 1) {
    n = std::stoi(argv[1]);
}
sf::RenderWindow window(sf::VideoMode(800, 800), "pentaflake");
while (window.isOpen()) {
    sf::Event event;
    while (window.pollEvent(event)) {
        if (event.type == sf::Event::Closed)
            window.close();
    }
        window.clear();
        drawpentagon(window, sf::Vector2f(400, 400), n, 500, 0);
        window.display();
}
}
```

4 ps3: NBody

4.1 Overview

For Project 3, I was assigned to make a simulation of the solar system. The Implementation is broken into two parts, A and B. Part, A of the project is reading from the file the mass,velocity, and radius of the celetial body and displaying a static image of the universe. In part b I will implement the planets movement using Newtons law of motion.

4.2 End Product

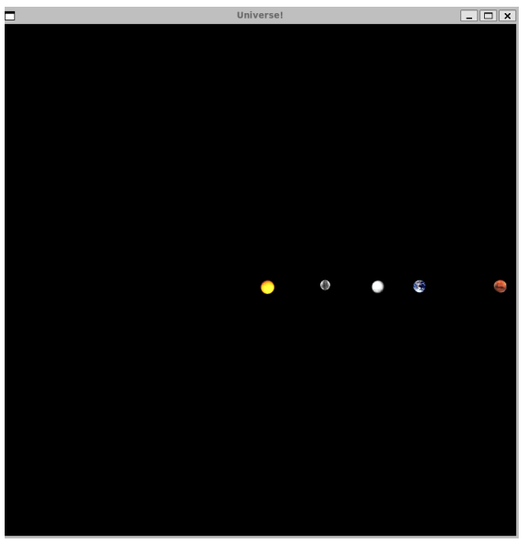

Figure 5: Part A

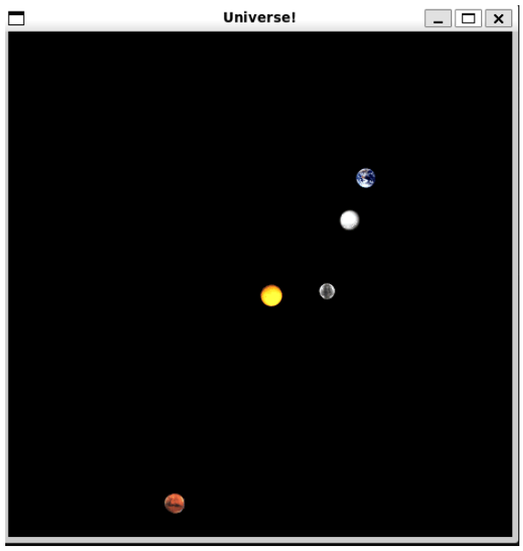

Figure 6: Part B

4.3 Design and Implementation

In the Implementation of PS3, I had to write functions for two classes. The classes are CelestialBody and Universe. The CelestialBody class is basically responsible for managing the planets and sun. The class needs to know the planet's mass, position, and velocity for movement. The universe needs to access the already CelestialBody and display it in whatever state it's in. To do that, we need to create a vector of CelestialBody inside the universe and use our draw function specifically made for the universe to display the predefined CelestialBody. The draw function for the CelestialBody is responsible for displaying the sprite using the information from the files. The step function is responsible for stimulating the planets rotating around the sun. To do that, we need to know how Newton's three laws of motion work. I need to iterate through each CelestialBody and calculate the total force(net force), velocity, and acceleration. The data structure that was used is vector because we are working with multiple Celestial bodies. It's efficient to put all the CelestialBody in a vector and initialize them by going through the vector. I made a series of tests for Project 3. I tested my insertion and extraction by reading from the files. I tested for position, velocity, step, and size by comparing the right value given with the value our code produced

4.4 What I already knew

I was already familiar with how sprites work from previous projects. I knew how to overload my extraction and insertion operator. I know how to set up main to accept command line arguments. If I did it again, I would use a smart pointer to make the CelestialBody. Smart pointers dynamically allocate space and deallocate when it is done being used.

4.5 What I learend

I learned about protected keywords. Function under the protect can be access by member or anything from the deprived class. I learned how I can make a static sprite Dynamic. I learned how to center a sprite.

4.6 Challenges

I didn't know how to center the sprite. My planet's position would be too big or too small. The position wouldn't be in the right order. I ran into many troubles. The first was how I could give celestial body access to the radius of the planets to do my calculation in the celestial body draw function. To do this, I initialized a celestial body variable inside my extraction. That takes the universe's radius. The math was a bit challenging because I calculated the net force total wrong. So, one planet would have more gravitation pull than the sun. This causes some bizarre interactions, like my planets flying across the window or hitting each.

4.7 Codebase

Makefile:

```
1
2  CC = g++
3  CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
4  LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
       lboost_unit_test_framework
5  # Your .hpp files
6  DEPS = CelestialBody.hpp  Universe.hpp
7  # Your compiled .o files
8  OBJECTS = CelestialBody.o  Universe.o
9  # The name of your program
10 PROGRAM = NBody
11
12 STATIC_LIB = NBody.a
13
14 .PHONY: all clean lint
15
16 all: $(PROGRAM) test
17
18
19
20 %.o: %.cpp $(DEPS)
21     $(CC) $(CFLAGS) -c $<
22
23 $(PROGRAM): main.o NBody.a
24     $(CC) $(CFLAGS) -o $@ $^ $(LIB)
25
26 test: test.o $(OBJECTS)
27     $(CC) $(CFLAGS) -o $@ $^ $(LIB)
28
29 main: main.o $(OBJECTS)
30     $(CC) $(CFLAGS) -o $@ $^ $(LIB)
31
32 $(STATIC_LIB): $(OBJECTS)
33     ar rcs $(STATIC_LIB) $@ $^ $(OBJECTS)
34
35 clean:
36     rm *.o $(PROGRAM) test NBody.a
37
38 lint:
39     cpplint *.cpp *.hpp
```

Universe.hpp:

```
1  // Copyright 2024 Marvens Luc
2  #pragma once
3  #include <vector>
4  #include <SFML/Graphics.hpp>
5  #include "CelestialBody.hpp"
6
7
8  namespace NB {
9  class Universe : public sf::Drawable {
10  public:
11     Universe();  // Required
12     explicit Universe(const std::string& filename);  // Optional
13     size_t size() const;  // Optional
```

```
    double radius() const;  // Optional
    const CelestialBody& operator[](size_t i) const;  // Optional
    friend std::istream& operator>>(std::istream& is, Universe& uni);
    friend std::ostream& operator<<(std::ostream& os, const Universe& uni);

    void step(double dt);  // Implemented in part b, behavior for part a is undefined
protected:
    void draw(
        sf::RenderTarget& window,
        sf::RenderStates states) const override;  // From sf::Drawable
private:
    double _radius;
    int _size;
    std::vector<CelestialBody> Cel_bodies;
    // Fields and helper functions go here
};

}  // namespace NB
```

Universe.cpp:

```
// Copyright 2024 Marvens Luc
#include "Universe.hpp"
#include <cmath>
#include <fstream>
#include <iostream>
const double gravity = 6.67e-11;


namespace NB {

std::istream& operator>>(std::istream& is, Universe& uni) {
    is >> uni._size;
    is >> uni._radius;
    uni.Cel_bodies.clear();
    CelestialBody body(uni.radius());
    for (int i = 0; i < uni._size; i++) {
        is >> body;
        uni.Cel_bodies.push_back(body);
    }
    return is;
}

std::ostream& operator<<(std::ostream& os, const Universe& uni) {
    os << uni._size << "\n";
    os << uni._radius << "\n";
    for (const auto& Cel_body : uni.Cel_bodies) {
        os << Cel_body << "\n";
    }
    return os;
}

Universe::Universe(const std::string& filename) {  // opens and read from file
    // open file
    std::ifstream file(filename);
    if (!file.is_open()) {
        std::cout << "File failed to open" << filename << std::endl;
        return;
```

```
38      }
39
40      file >> *this;  // extraction
41  }
42
43  size_t Universe::size() const { return _size; }
44  double Universe::radius() const { return _radius; }
45
46  void Universe::draw(sf::RenderTarget& window, sf::RenderStates states) const
        {
47      for (const auto& Cel_body : Cel_bodies) {
48          window.draw(Cel_body, states);
49      }
50  }
51
52  const CelestialBody& Universe::operator[](size_t i) const { return
       Cel_bodies.at(i); }
53
54  Universe::Universe() {
55      _radius = 1000.0;
56      _size = 0;
57      // Cel_bodies.reserve(_size);
58  }
59
60
61  void Universe::step(double dt) {
62      // iterate through celestial body
63      // std::unique_ptr<std::vector<CelestialBody>> bodies =
64      // std::make_unique<std::vector<CelestialBody>>(Cel_bodies);
65      for (int i = 0; i < _size;
66           i++) {  // iterate through the planets compute netforce and other
       position
67          sf::Vector2f Net_force(0.0f, 0.0f);
68          for (int j = 0; j < _size; j++) {
69              if (!(Cel_bodies[i].position().x == Cel_bodies[j].position().x
       &&
70                    Cel_bodies[i].position().y == Cel_bodies[j].position().y))
        {
71                  double delta_x =
72                      Cel_bodies[j].position().x -
73                      Cel_bodies[i].position().x;  // difference of the
       distance between 2 planets
74                  double delta_y = Cel_bodies[j].position().y - Cel_bodies[i].
       position().y;
75                  // std::cout << "delta y " << delta_y << std::endl;
76                  double delt_r =
77                      sqrt((delta_x * delta_x) + (delta_y * delta_y));  //
       distance between cel body
78                  // std::cout << "delta r " << delt_r << std::endl;
79                  double force_Mag =
80                      (gravity * Cel_bodies[i].mass() * Cel_bodies[j].mass())
       / (delt_r * delt_r);
81                  // std::cout << "force Mag " << force_Mag << std::endl;
82                  sf::Vector2f forces;
83                  forces.x = force_Mag * (delta_x / delt_r);
84                  forces.y = force_Mag * (delta_y / delt_r);
85                  // sum off all the force
86                  Net_force.x += forces.x;
87                  Net_force.y += forces.y;
```

```
            }
        }
        sf::Vector2f New_acceleration;
        New_acceleration.x = Net_force.x / Cel_bodies[i].mass();
        New_acceleration.y = Net_force.y / Cel_bodies[i].mass();
        // std::cout << " Net_force " << Net_force.x << " " << Net_force.y
   << std::endl;
        sf::Vector2f New_velocity;
        New_velocity.x = dt * New_acceleration.x;
        New_velocity.y = dt * New_acceleration.y;
        // updating new positon and velcity
        Cel_bodies[i].set_velocity(New_velocity);
        // std::cout << " pos " <<   Cel_bodies[2].position().x << " " <<
   Cel_bodies[2].position().y
        // << std::endl;
    }
    for (int i = 0; i < _size; i++) {
        sf::Vector2f New_pos;
        New_pos.x = dt * Cel_bodies[i].velocity().x;
        New_pos.y = dt * Cel_bodies[i].velocity().y;
        Cel_bodies[i].set_position(New_pos);
    }
}
}  // namespace NB
// namespace NB
```

CelestialBody.cpp:

```
#pragma once
#include <iostream>
#include <memory>
#include <SFML/Graphics.hpp>
// Copyright 2024 Marvens Luc

namespace NB {
// class Universe;
class CelestialBody : public sf::Drawable {
 public:
    CelestialBody();  // Required
    explicit CelestialBody(double radius);
    sf::Vector2f position() const { return _positon; }  // Optional
    sf::Vector2f velocity() const { return _velocity; }  // Optional
    float mass() const { return _mass; }  // Optional
    void set_position(sf::Vector2f pos);
    void set_velocity(sf::Vector2f vel);
    void setUniRadius(double radius) { _radius = radius; }
    friend std::istream& operator>>(std::istream& is, CelestialBody& uni);
    friend std::ostream& operator<<(std::ostream& os, const CelestialBody&
   uni);

 protected:
    void draw(
        sf::RenderTarget& window,
        sf::RenderStates states) const override;  // From sf::Drawable

 private:
    sf::Vector2f _positon;
    sf::Vector2f _velocity;
    float _mass;
    std::string _file_name;
```

```
32      double _radius;
33      // Universe* _universe; // pointer to the Universe objects
34
35      // Fields and helper methods go here
36  };
37
38  }  // namespace NB
```

CelestialBody.cpp:

```
1   // Copyright 2024 Marvens Luc
2   #include "CelestialBody.hpp"
3   #include <algorithm>
4   #include <fstream>
5   #include <iostream>
6   #include "Universe.hpp"
7
8   namespace NB {
9
10  std::istream& operator>>(std::istream& is, CelestialBody& uni) {
11      is >> uni._positon.x;
12      is >> uni._positon.y;
13      is >> uni._velocity.x;
14      is >> uni._velocity.y;
15      is >> uni._mass;
16      is >> uni._file_name;
17
18      return is;
19  }
20  std::ostream& operator<<(std::ostream& os, const CelestialBody& uni) {
21      return os << uni._positon.x << " " << uni._positon.y << " " << uni.
    _velocity.x << " "
22                << uni._velocity.y << " " << uni._mass << " " << uni.
    _file_name;
23  }
24
25  void CelestialBody::draw(sf::RenderTarget& window, sf::RenderStates states)
    const {
26      sf::Texture tex;
27      tex.loadFromFile(_file_name);
28      sf::Sprite sprit(tex);
29      // sprit.setTexture(tex);
30
31      // lets fail
32      // modify
33      // Calculate scaling factor
34      double r = _radius;
35      // Scale and apply positions
36      sf::Vector2f window_size;
37      sf::Vector2f half_window_size(window.getSize().x / 2, window.getSize().y
    / 2);
38      sf::Vector2f center =
39          sprit.getTransform().transformPoint(half_window_size);  // center
    midpoint of the screen
40      window_size.x =
41          center.x +
42          (_positon.x / r *
43           half_window_size.x);  // center the planets and adjust the position
     using x and y axis
44      window_size.y = center.y - (_positon.y / r * half_window_size.x);
```

```
    sprit.setPosition(window_size);
    window.draw(sprit, states);
}  // From sf::Drawable

CelestialBody::CelestialBody() {
    _positon.x = 0;
    _positon.y = 0;
    _velocity.x = 0;
    _velocity.y = 0;
    _file_name = "file";
}  // Required

CelestialBody::CelestialBody(double radius) { _radius = radius; }
void CelestialBody::set_position(sf::Vector2f pos) { _positon += pos; }
void CelestialBody::set_velocity(sf::Vector2f vel) { _velocity += vel; }
}  // namespace NB
```

main.cpp:

```
#include <iostream>
#include <limits>
#include <SFML/Audio.hpp>
#include <SFML/Graphics.hpp>
#include "CelestialBody.hpp"
#include "Universe.hpp"
// Copyright 2024 Marvens Luc

int main(int argc, char* argv[]) {
    sf::Music uni;
    double Time, deltaT;
    double elapsed_time = 0;
    uni.openFromFile("2001.wav");
    uni.play();
    sf::RenderWindow window(sf::VideoMode(500, 500), "Universe!");
    window.setFramerateLimit(60);
    NB::Universe body;
    std::cin >> body;
    Time = std::atoi(argv[1]);
    deltaT = std::atoi(argv[2]);
    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                window.close();
                break;
            }
        }

        if (elapsed_time < Time) {
            body.step(deltaT);
            elapsed_time += deltaT;
        }
        window.clear();
        window.draw(body);
        // std::cout << body << std::endl;
        window.display();
    }

    return 0;
}
```

test.cpp:

```
// Copyright 2024 Marvens Luc
#include <iostream>
#include <sstream>
#include <string>
#include <SFML/Graphics.hpp>
#include "CelestialBody.hpp"
#include "Universe.hpp"
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>

// Test insertion
// raduis
// numberofplanets
// velocity
// extraction
// position
// size

/*BOOST_AUTO_TEST_CASE(TestingExtraction) {
    //std::stringstream num_of_planets;
    NB::Universe planet("planets.txt");
    //num_of_planets >> planet;
    int number_of_planets = planet.size();
    int expected_number_of_plants = 5;
    BOOST_REQUIRE_EQUAL(number_of_planets, expected_number_of_plants);
}*/

BOOST_AUTO_TEST_CASE(TestingInsertion) {
    NB::Universe universe;
    std::stringstream input, output;
    std::string insertion = { "1\n"
                              "100\n"
                              "10 20 2 1 1e+20 earth.gif\n" };
    input << insertion;
    input >> universe;
    output << universe;
    BOOST_REQUIRE_EQUAL(input.str(), output.str());
}

BOOST_AUTO_TEST_CASE(PlanetMass) {
    NB::Universe body("1body.txt");
    float mass, expectedMass;
    mass = body[0].mass();
    expectedMass = 1e20;
    BOOST_REQUIRE_CLOSE(mass, expectedMass, 0.0001);
}

BOOST_AUTO_TEST_CASE(PlanetRadius) {
    double Radius, expectedRadius;
    NB::Universe body("1body.txt");
    Radius = body.radius();
    expectedRadius = 100.0;
    BOOST_REQUIRE_EQUAL(Radius, expectedRadius);
}

BOOST_AUTO_TEST_CASE(velocity) {
    NB::Universe body("1body.txt");
```

```
    sf::Vector2f velocity2(2.0f, 1.0f);
    sf::Vector2f velocity = body[0].velocity();
    BOOST_REQUIRE_CLOSE(velocity2.x, velocity.x, 0.0001);
    BOOST_REQUIRE_CLOSE(velocity2.y, velocity.y, 0.0001);
}

BOOST_AUTO_TEST_CASE(Position) {
    NB::Universe body("1body.txt");
    sf::Vector2f velocity2(10.0f, 20.0f);
    sf::Vector2f velocity = body[0].position();
    BOOST_REQUIRE_CLOSE(velocity2.x, velocity.x, 0.0001);
    BOOST_REQUIRE_CLOSE(velocity2.y, velocity.y, 0.0001);
}

BOOST_AUTO_TEST_CASE(Size) {
    NB::Universe body("planets.txt");
    BOOST_REQUIRE_EQUAL(body.size(), 5);
}

BOOST_AUTO_TEST_CASE(TestingStep1) {
    NB::Universe universe("planets.txt");
    universe.step(25000);
    NB::CelestialBody earth;
    earth = universe[0];
    // 1.4960e+11 7.4500e+08 -1.4820e+02 2.9800e+04 5.9740e+24 earth.gif
    BOOST_REQUIRE_CLOSE(earth.position().x, 1.4960e+11, 1);
    BOOST_REQUIRE_CLOSE(earth.position().y, 7.4500e+08, 1);
    BOOST_REQUIRE_CLOSE(earth.velocity().x, -1.4820e+02, 1);
    BOOST_REQUIRE_CLOSE(earth.velocity().y, 2.9800e+04, 1);
}

BOOST_AUTO_TEST_CASE(TestingStep2) {
    NB::Universe universe("planets.txt");
    universe.step(25000);
    universe.step(25000);
    NB::CelestialBody earth;
    earth = universe[0];
    // 1.4959e+11 1.4900e+09 -2.9640e+02 2.9799e+04 5.9740e+24 earth.gif
    BOOST_REQUIRE_CLOSE(earth.position().x, 1.4960e+11, 1);
    BOOST_REQUIRE_CLOSE(earth.position().y, 1.4900e+09, 1);
    BOOST_REQUIRE_CLOSE(earth.velocity().x, -2.9640e+02, 1);
    BOOST_REQUIRE_CLOSE(earth.velocity().y, 2.9799e+04, 1);
}

BOOST_AUTO_TEST_CASE(TestingStep4) {
    NB::Universe universe("planets.txt");
    universe.step(75000);
    universe.step(75000);
    NB::CelestialBody earth;
    earth = universe[0];
    // 1.4959e+11 1.4900e+09 -2.9640e+02 2.9799e+04 5.9740e+24 earth.gif
    BOOST_REQUIRE_CLOSE(earth.position().x, 1.4960e+11, 1);
    // BOOST_REQUIRE_CLOSE(earth.position().y, 1.4900e+09, 1);
    BOOST_REQUIRE_CLOSE(earth.position().y, 4.46950195e+09, 1);
    BOOST_REQUIRE_CLOSE(earth.velocity().x, -889.255066, 1);
    BOOST_REQUIRE_CLOSE(earth.velocity().y, 2.9799e+04, 1);
}


```

```
118 BOOST_AUTO_TEST_CASE(TestingStep3) {
119     NB::Universe universe("planets.txt");
120     NB::CelestialBody earth;
121     int delta = 25000;
122     // int num_step = 31557600 / 25000;  // 12,623
123     /*for (int i = 0; i < 31557600 ; i+=25000) {
124         universe.step(delta);
125     }*/
126     int i = 0;
127     while (i < 31557600) {
128 
129         universe.step(delta);
130         i += 250000;
131     }
132 
133 
134     // earth = universe[0];
135     //  universe.step(delta);
136 
137     // 1.4959e+11 -1.6531e+09 3.2949e+02 2.9798e+04 5.9740e+24 earth.gif
138     BOOST_REQUIRE_CLOSE(earth.position().x, 1.4960e+11, 1);
139     BOOST_REQUIRE_CLOSE(earth.position().y, -1.6531e+09, 1);
140     BOOST_REQUIRE_CLOSE(earth.velocity().x, 3.2949e+02, 1);
141     BOOST_REQUIRE_CLOSE(earth.velocity().y, 2.9798e+04, 1);
142 }
143 
144 
145 /*BOOST_AUTO_TEST_CASE(leapforgmethod) {
146     NB::Universe universe("planets.txt");
147     universe.step(25000);
148     NB::CelestialBody earth;
149     earth = universe[0];
150     // 1.4960e+11 7.4500e+08 -1.4820e+02 2.9800e+04 5.9740e+24 earth.gif
151     BOOST_REQUIRE_CLOSE(earth.position().x, 1.4960e+11, 1);
152     BOOST_REQUIRE_CLOSE(earth.position().y, 7.4500e+08, 1);
153     BOOST_REQUIRE_CLOSE(earth.velocity().x, -1.4820e+02, 1);
154     BOOST_REQUIRE_CLOSE(earth.velocity().y, 2.9800e+04, 1);
155 }*/
```

5 ps4: Sokoban

5.1 Overview

Sokoban is a title-based game in which a character pushes boxes into storage.

5.2 End Product

Figure 7: Sokoban UI

Figure 8: Sokoban

5.3 Design and Implementation

Project 4 is separated into two parts, a and b. Part A will be loading in the levels. To do that, I need to overload the insertion and extraction operator to read the levels from files. The extraction operator reads from the file and stores it into a 1D vector. The insertion reads in the width and height of the file. It turns my vector, all characters, into a 1D array. I did that because it helps make updating, moving the characters, and finding the location of an object easier. For the texture, I made a map that used a key, the symbol of each object in the map, and the value is a load from the file. I did it this way because I didn't want to load the file using my draw function. I can just use a loop and one sprite to go through the 1d vector and reference the map of texture based on the symbol. I made a function to find the player and box location. I made a movement function that moves the character and box based on direction. For extra credit, I made the player sprite update based on directional input. I made a reset function to reset the game if you press R. By storing the original copy of the map into a vector and calling that copy anytime you press R. I made multiple tests for project 4. I made a test for collision; meaning will the player or box go through the blocks, crate, or off the screen? I tested if my win function worked; I tested my reset function by checking the object's original location with the current. I made sure the player was only moving one tile at a time. If I were to redo it, I would use lambda expressions instead of making multiple loops to look for the play or box location.

5.4 What I already knew

Due to project 3 NBody, I knew how to load a texture and display a sprite from a file. I knew how to overload my extraction and insertion operator. I was familiar with how a map work and how to iterator through it.

5.5 What I learend

I learned a lot about game development. I learned, designed, and test my own collosion function. I learned on how to make gamestate meaning when the character win. The game goes to a state of pause, else the gamestate will be runing. I leaned how to use lamba expression

5.6 Challenges

The challenge I faced was that when I started, I made multiple sprites in my program. That causes a lot of weird behavior. When I update my game, the sprite will move, but the space accompanying the sprite will be in its original position. I mixed up the height and width, causing the game to load the objects in the wrong place. Sometimes, it causes a seg fault error. I tried to make an undo button by using a stack to store the previous position of the box and player. I couldn't get it to work because my map was not updating properly.

5.7 Codebase

Makefile:

```
1
2  CC = g++
3  CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
4  LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
       lboost_unit_test_framework
5  # Your .hpp files
6  DEPS = Sokoban.hpp
7  # Your compiled .o files
8  OBJECTS = Sokoban.o
9  # The name of your program
10 PROGRAM = Sokoban
11
12 STATIC_LIB = Sokoban.a
13
14 .PHONY: all clean lint
15
16 all: $(PROGRAM) test
17
18
19
20 %.o: %.cpp $(DEPS)
21     $(CC) $(CFLAGS) -c $<
22
23 $(PROGRAM): main.o Sokoban.a
24     $(CC) $(CFLAGS) -o $@ $^ $(LIB)
25
26 test: test.o $(OBJECTS)
27     $(CC) $(CFLAGS) -o $@ $^ $(LIB)
28
29 main: main.o $(OBJECTS)
30     $(CC) $(CFLAGS) -o $@ $^ $(LIB)
31
32 $(STATIC_LIB): $(OBJECTS)
33     ar rcs $(STATIC_LIB) $@ $^ $(OBJECTS)
34
35 clean:
36     rm *.o $(PROGRAM) test Sokoban.a
37
38 lint:
39     cpplint *.cpp *.hpp
```

Universe.hpp:

```
1  #pragma once
2  #include <fstream>
3  #include <iostream>
4  #include <stack>
5  #include <vector>
6  #include <SFML/Graphics.hpp>
7  // Copyright 2024 Marvens Luc
8
9
10 namespace SB {
11 enum  Direction { Up, Down, Left, Right };
12 class Sokoban : public sf::Drawable {
13  public:
```

```
    static const int TILE_SIZE = 64;
    enum class GamesState { Running, Pause };  // Helps with the
    Sokoban();
    void load_resouces();
    // Sokoban(const std::string& filname); const // Optional
    unsigned int pixelHeight() const;  // Optional
    unsigned int pixelWidth() const;  // Optional

    unsigned int height() const;
    unsigned int width() const;

    sf::Vector2u playerLoc() const;
    sf::Vector2u BoxLoc() const;
    bool isWon() const;
    bool checkValidity(sf::Vector2u pos, Direction dir);
    void movePlayer(Direction dir);
    void moveBox(Direction dir);
    friend std::ostream& operator<<(std::ostream& out, const Sokoban& s);
    friend std::istream& operator>>(std::istream& in, Sokoban& s);
    void undo();  // Optional XC
    void reset();  // Optional XC
    void FindBox();
    void Find_Storage();
    void setPlayerLoc(int x, int y);
    void defaultloc();
    void default_box_loc();
    bool check_box_valdity(sf::Vector2u pos, Direction dir);
    GamesState gamestate;


protected:
    void draw(sf::RenderTarget& target, sf::RenderStates states) const
   override;

private:
    size_t _height;
    size_t _width;
    std::vector<char> _all_characters;
    std::vector<sf::Vector2u> _Storage;
    sf::Texture _crate;
    std::map<Direction, sf::Texture> _player;
    Direction _direction = Direction::Down;
    int box_count;
    int storage_count;
    std::stack<std::vector<sf::Vector2u>> Game_history;  // store game
   history;
    std::vector<sf::Vector2u> _Box;
    std::string filename;
    sf::Vector2u _player_location;
    sf::Vector2u _box_location;
    std::map<char, sf::Texture> _texture;
    std::vector<char> _Map_reset;
    // Any fields you need go here.
};

}  // namespace SB
```

Universe.cpp:

```
#include "Sokoban.hpp"
```

```cpp
#include <algorithm>
#include <fstream>
#include <iostream>
#include <sstream>
// Copyright 2024 Marvens Luc
namespace SB {
Sokoban::Sokoban() {
    storage_count = 0;
    box_count = 0;
}
void Sokoban::defaultloc() {
    for (size_t y = 0; y < _height; y++) {
        for (size_t x = 0; x < _width; x++) {
            int i = x + y * _width;
            if (_Map_reset[i] == '@') {
                _player_location.x = x;
                _player_location.y = y;
                break;
            }
        }
    }
}
// Sokoban::Sokoban(const std::string &filname) // Optional
bool Sokoban::checkValidity(sf::Vector2u pos, Direction dir) {
    int i = pos.x + pos.y * _width;
    if (pos.x >= _width || pos.y >= _height) {
        return false;
    }
    // take pnew pos player wants to move
    auto pbox = std::find(_Box.begin(), _Box.end(), pos);
    if (_all_characters[i] != '#' && pbox == _Box.end())
        return true;
    return false;
}


// return pixelHeight used to render windpw
unsigned int Sokoban::pixelHeight() const { return _height * TILE_SIZE; }

// return pixelHeight used to render windpw
unsigned int Sokoban::pixelWidth() const { return _width * TILE_SIZE; }

void Sokoban::movePlayer(Direction dir) {
    moveBox(dir);
    sf::Vector2u newpos;
    // Calculate the new position based on the direction
    newpos.x = _player_location.x;
    newpos.y = _player_location.y;
    _direction = dir;
    if (dir == Direction::Up) {
        newpos.y -= 1;  // Move up by decreasing the y-coordinate
    } else if (dir == Direction::Left) {
        newpos.x -= 1;
    } else if (dir == Direction::Right) {
        newpos.x += 1;
    } else if (dir == Direction::Down) {
        newpos.y += 1;
    }

```

```
61      if (checkValidity(newpos, dir) == true) {
62          _player_location = newpos;  // Update the actual player position
63      }
64  }
65
66
67  // getter for height
68  unsigned int Sokoban::height() const { return _height; }
69
70  // getter for width
71  unsigned int Sokoban::width() const { return _width; }
72
73  // reads from the file height and width and read the
74  std::istream& operator>>(std::istream& in, Sokoban& s) {
75      // character put them in a vecto
76
77      in >> s._height;
78      in >> s._width;
79      // std::cout << s._height << " " << s._width << std::endl;
80      in.ignore();
81      char c;
82      s._all_characters.resize(s._width * s._height);
83      s._Map_reset.resize(s._width * s._height);
84      for (size_t y = 0; y < s.height(); y++) {
85          for (size_t x = 0; x < s.width(); x++) {
86              int i = x + y * s._width;
87              in >> c;
88              if (c == '@') {
89                  s._player_location.x = x;
90                  s._player_location.y = y;
91              }
92              if (c == 'a') {
93                  s.storage_count += 1;
94              }
95              if (c == 'A') {
96                  s.box_count += 1;
97              }
98              // std::cout << c;
99              s._all_characters[i] = c;
100             s._Map_reset[i] = c;
101             // std::cout << s._all_characters.at(i);
102         }
103
104         std::cout << "\n";
105         in.ignore();
106     }
107     s.load_resouces();
108     // s.defaultloc();
109     //  s._direction == Sokoban::Direction::Down;
110     s.FindBox();
111     s.Find_Storage();
112
113     return in;
114 }
115
116 // out the height, width, and whats in the vector of char exactly based on
        what in the level file
117 std::ostream& operator<<(std::ostream& out, const Sokoban& s) {
118     out << s._height;
```

```
    out << s._width;
    for (size_t y = 0; y < s.height(); y++) {
        for (size_t x = 0; x < s.width(); x++) {
            int i = x + y * s._width;
            out << s._all_characters.at(i);
        }
    }
    return out;
}


// finds the player
sf::Vector2u Sokoban::playerLoc() const {
    int x, y;
    x = _player_location.x;
    y = _player_location.y;
    return sf::Vector2u(x, y);
}


void Sokoban::FindBox() {
    for (size_t y = 0; y < _height; y++) {
        for (size_t x = 0; x < _width; x++) {
            int i = x + y * _width;
            char c = _all_characters[i];
            if (c == 'A') {
                sf::Vector2u boxPos(x, y);  // Create a new position for the box
                _Box.push_back(boxPos);
            }
        }
    }
}


// loosp through the vector and based on that positon it creates a sprite and setposition
void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const {
    // std::cout << " -1 " << std::endl;
    for (size_t y = 0; y < _height; y++) {
        for (size_t x = 0; x < _width; x++) {
            int i = x + y * _width;
            sf::Sprite sprite;
            sprite.setPosition(x * TILE_SIZE, y * TILE_SIZE);
            char c = _all_characters[i];
            if (c == '#') {
                sprite.setTexture(_texture.at('#'));
            } else if (c == 'a') {
                sprite.setTexture(_texture.at('a'));
            } else {
                sprite.setTexture(_texture.at('.'));
            }
            target.draw(sprite, states);
        }
    }

    sf::Sprite player;
    player.setTexture(_player.at(_direction));
```

```
175     player.setPosition(playerLoc().x * TILE_SIZE, playerLoc().y * TILE_SIZE)
        ;
176     target.draw(player, states);
177     for (unsigned int i = 0; i < _Box.size(); i++) {
178         sf::Sprite boxSprite;
179         boxSprite.setTexture(_crate);  // Assuming 'A' is the key for the
        box texture
180         boxSprite.setPosition(_Box[i].x * TILE_SIZE, _Box[i].y * TILE_SIZE);
181         target.draw(boxSprite, states);
182     }
183 }
184 // a map of texture that takes a charcter as key and load a file assciated
        with that charcter
185 void Sokoban::load_resouces() {
186     // map to store the tetxure associated with chracter
187     _texture['#'].loadFromFile("sokoban/block_06.png");
188     _texture['.'].loadFromFile("sokoban/ground_01.png");
189     _texture['A'].loadFromFile("sokoban/crate_03.png");
190     _texture['a'].loadFromFile("sokoban/ground_04.png");
191     _crate.loadFromFile("sokoban/crate_03.png");
192     _player[Direction::Up].loadFromFile("sokoban/player_08.png");
193     _player[Direction::Left].loadFromFile("sokoban/player_20.png");
194     _player[Direction::Right].loadFromFile("sokoban/player_17.png");
195     _player[Direction::Down].loadFromFile("sokoban/player_05.png");
196 }
197
198 void Sokoban::moveBox(Direction dir) {
199     // std::cout << "are you in " << std::endl;
200     sf::Vector2u player = _player_location;
201     // predicate the players positon
202     if (dir == Direction::Up) {
203         // std::cout << "are you in up " << std::endl;
204         player.y -= 1;
205     } else if (dir == Direction::Down) {
206         player.y += 1;
207     } else if (dir == Direction::Left) {
208         player.x -= 1;
209     } else if (dir == Direction::Right) {
210         player.x += 1;
211     }
212     // compare the box location to the predicated player location
213     auto pbox = std::find(_Box.begin(), _Box.end(), player);
214     if (pbox != _Box.end()) {
215         // std::cout << "we are in the if that finds a box " << std::endl;
216         sf::Vector2u temp = *pbox;
217         if (dir == Direction::Up) {
218             temp.y -= 1;
219         } else if (dir == Direction::Down) {
220             temp.y += 1;
221         } else if (dir == Direction::Left) {
222             temp.x -= 1;
223         } else if (dir == Direction::Right) {
224             temp.x += 1;
225         }
226         int newBoxIndex = temp.x + temp.y * _width;
227         if (temp.x >= _width || temp.y >= _height)
228             return;
229         if (_all_characters[newBoxIndex] == '#')
230             return;
```

```
231         if (std::find(_Box.begin(), _Box.end(), temp) != _Box.end())
232             return;
233 
234         *pbox = temp;
235     }
236     /*for (size_t y = 0; y < height(); y++) {
237         for (size_t x = 0; x < width(); x++) {
238             int i = x + y * _width;
239             std::cout << _all_characters.at(i);
240         }
241         std::cout << "\n";
242     }*/
243 }
244 
245 
246 void Sokoban::setPlayerLoc(int x, int y) {
247     _player_location.x = x;
248     _player_location.y = y;
249 }
250 
251 
252 bool Sokoban::check_box_valdity(sf::Vector2u pos, Direction dir) {
253     int i = pos.x + pos.y * _width;
254     if (pos.x >= _width || pos.y >= _height) {
255         return false;  // Outside bounds
256     }
257     if (dir == Direction::Up) {
258         if (_all_characters[i] != '#')
259             return true;
260     } else if (dir == Direction::Down) {
261         if (_all_characters[i] != '#')
262             return true;
263     } else if (dir == Direction::Left) {
264         if (_all_characters[i] != '#')
265             return true;
266 
267     } else if (dir == Direction::Right) {
268         if (_all_characters[i] != '#')
269             return true;
270     }
271     return false;
272 }
273 
274 
275 bool Sokoban::isWon() const {
276     int store = storage_count;
277     sf::Vector2u storre;
278     if (storage_count < box_count) {
279         // find the the box and compare it to storage location
280         std::cout << storage_count << " " << box_count;
281         for (unsigned int i = 0; i < _Storage.size(); i++) {
282             storre.x = _Storage[i].x;
283             storre.y = _Storage[i].y;
284             // if !-end return true
285         }
286 
287         for (unsigned int i = 0; i < _Box.size(); i++) {
288             if (_Box[i].x == storre.x && _Box[i].y == storre.y) {
289                 store -= 1;
```

```
            }

            if (store == 0) {
                return true;
            }
        }
    }

    return std::all_of(_Box.begin(), _Box.end(), [this](sf::Vector2u box) {
        int i = box.x + box.y * _width;
        return (_all_characters[i] == 'a');
    });
}


void Sokoban::Find_Storage() {
    for (size_t y = 0; y < _height; y++) {
        for (size_t x = 0; x < _width; x++) {
            int i = x + y * _width;
            char c = _all_characters[i];
            if (c == 'a') {
                sf::Vector2u boxPos(x, y);  // Create a new position for the box
                _Storage.push_back(boxPos);
            }
        }
    }
}


void Sokoban::reset() {
    for (size_t y = 0; y < height(); y++) {
        for (size_t x = 0; x < width(); x++) {
            int i = x + y * _width;
            // std::cout << c;
            _all_characters[i] = _Map_reset[i];  // map_reset has the map in it orginal state
        }
    }
    _Box.clear();
    defaultloc();
    Find_Storage();
    FindBox();
}
};  // namespace SB
```

main.cpp:

```
#include <iostream>
#include <sstream>
#include <string>
#include <SFML/Graphics.hpp>
#include "SFML/Audio.hpp"
#include "Sokoban.hpp"
// Copyright 2024 Marvens Luc
int main(int argc, char* argv[]) {
    // SB::Sokoban sokoban;
    std::ifstream level("sokoban/" + std::string(argv[1]));
    SB::Sokoban sokoban;
    level >> sokoban;
```

```
13      sf::RenderWindow window(sf::VideoMode(sokoban.pixelWidth(), sokoban.
    pixelHeight()), "Sokoban!");
14      window.setFramerateLimit(10);
15      sokoban.gamestate = SB::Sokoban::GamesState::Running;
16      while (window.isOpen()) {
17          sf::Event event;
18          while (window.pollEvent(event)) {
19              if (event.type == sf::Event::Closed) {
20                  window.close();
21                  break;
22              }
23          }
24
25          if (sokoban.gamestate == SB::Sokoban::GamesState::Running) {
26              if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::W)) {
27                  sokoban.movePlayer(SB::Direction::Up);
28              } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::A)) {
29                  sokoban.movePlayer(SB::Direction::Left);
30              } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::D)) {
31                  sokoban.movePlayer(SB::Direction::Right);
32              } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::S)) {
33                  sokoban.movePlayer(SB::Direction::Down);
34              } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::R)) {
35                  sokoban.reset();
36                  window.clear();
37                  window.draw(sokoban);
38                  window.display();
39              }
40          }
41
42          if (sokoban.isWon()) {
43              sokoban.gamestate = SB::Sokoban::GamesState::Pause;
44              sf::Texture texture;
45              texture.loadFromFile("sokoban/winner.png");
46              sf::Sprite sprite;
47              sprite.setTexture(texture);
48              sprite.setPosition(
49                  window.getSize().x / 2 - sprite.getGlobalBounds().width / 2,
50                  window.getSize().y / 2 - 300);
51              sf::Text text;
52              sf::Font font;
53              sf::Music piano;
54              piano.openFromFile("brass.wav");
55              piano.play();
56              font.loadFromFile("Roboto-Bold.ttf");
57              text.setFont(font);
58              text.setCharacterSize(100);
59              text.setFillColor(sf::Color::Yellow);
60              text.setString("win");
61              text.setPosition(
62                  window.getSize().x / 2 - text.getGlobalBounds().width / 2,
    window.getSize().y / 2);
63              window.draw(sprite);
64              window.draw(text);
65              window.display();
66          }
67          window.clear();
68          window.draw(sokoban);
69          window.display();
```

```
    }


    return 0;
}
```

test.cpp:

```
// Copyright 2022
// By Dr. Rykalova
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include "Sokoban.hpp"
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>

BOOST_AUTO_TEST_CASE(moveplayer) {
    std::ifstream level("sokoban/level1.lvl");
    SB::Sokoban sokoban;
    level >> sokoban;
    sokoban.movePlayer(SB::Direction::Up);
    BOOST_REQUIRE_EQUAL(sokoban.playerLoc().x, 3);
    BOOST_REQUIRE_EQUAL(sokoban.playerLoc().y, 5);
    sokoban.movePlayer(SB::Direction::Left);
    BOOST_REQUIRE_EQUAL(sokoban.playerLoc().x, 2);  // left = 3
    BOOST_REQUIRE_EQUAL(sokoban.playerLoc().x, 2);
    BOOST_REQUIRE_EQUAL(sokoban.playerLoc().y, 5);
    sokoban.movePlayer(SB::Direction::Down);
    BOOST_REQUIRE_EQUAL(sokoban.playerLoc().x, 2);
    BOOST_REQUIRE_EQUAL(sokoban.playerLoc().y, 6);
    sokoban.movePlayer(SB::Direction::Right);
    BOOST_REQUIRE_EQUAL(sokoban.playerLoc().x, 3);
    BOOST_REQUIRE_EQUAL(sokoban.playerLoc().y, 6);
    level.close();
}


BOOST_AUTO_TEST_CASE(Boxcollidewithwall) {  // box collide with wall
    std::ifstream level("sokoban/level1.lvl");
    SB::Sokoban sokoban;
    level >> sokoban;
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Right);
    BOOST_REQUIRE_EQUAL(sokoban.playerLoc().x, 7);
    BOOST_REQUIRE_EQUAL(sokoban.playerLoc().y, 6);
    level.close();
}

BOOST_AUTO_TEST_CASE(Win) {
    std::ifstream level("sokoban/autowin.lvl");
    SB::Sokoban sokoban;
    level >> sokoban;
```

```
53      BOOST_REQUIRE_EQUAL(sokoban.isWon(), true);
54      level.close();
55  }
56
57  BOOST_AUTO_TEST_CASE(reset) {
58      std::ifstream level("sokoban/level1.lvl");
59      SB::Sokoban sokoban;
60      level >> sokoban;
61      sokoban.movePlayer(SB::Direction::Up);
62      BOOST_REQUIRE_EQUAL(sokoban.playerLoc().x, 3);
63      BOOST_REQUIRE_EQUAL(sokoban.playerLoc().y, 5);
64      sokoban.reset();
65      BOOST_REQUIRE_EQUAL(sokoban.playerLoc().x, 3);
66      BOOST_REQUIRE_EQUAL(sokoban.playerLoc().y, 6);
67      level.close();
68  }
69
70  BOOST_AUTO_TEST_CASE(Playercollidewithwall) {  // player collide with wall
71      std::ifstream level("sokoban/level1.lvl");
72      SB::Sokoban sokoban;
73      level >> sokoban;
74      sokoban.movePlayer(SB::Direction::Left);
75      sokoban.movePlayer(SB::Direction::Left);
76      sokoban.movePlayer(SB::Direction::Left);
77      sokoban.movePlayer(SB::Direction::Left);
78      sokoban.movePlayer(SB::Direction::Left);
79      sokoban.movePlayer(SB::Direction::Left);
80      sokoban.movePlayer(SB::Direction::Left);
81      BOOST_REQUIRE_EQUAL(sokoban.playerLoc().x, 1);
82      BOOST_REQUIRE_EQUAL(sokoban.playerLoc().y, 6);
83      level.close();
84  }
85
86  BOOST_AUTO_TEST_CASE(boxcollidewithbox) {  // box on box violence
87      std::ifstream level("sokoban/level2.lvl");
88      SB::Sokoban sokoban;
89      level >> sokoban;
90      sokoban.movePlayer(SB::Direction::Up);
91      sokoban.movePlayer(SB::Direction::Up);
92      BOOST_REQUIRE_EQUAL(sokoban.playerLoc().x, 8);
93      BOOST_REQUIRE_EQUAL(sokoban.playerLoc().y, 5);
94      level.close();
95  }
96
97
98  BOOST_AUTO_TEST_CASE(playerwithbox) {  // player on box violence
99      std::ifstream level("sokoban/level2.lvl");
100     SB::Sokoban sokoban;
101     level >> sokoban;
102     sokoban.movePlayer(SB::Direction::Up);
103     sokoban.movePlayer(SB::Direction::Up);
104     BOOST_REQUIRE_EQUAL(sokoban.playerLoc().x, 8);
105     BOOST_REQUIRE_EQUAL(sokoban.playerLoc().y, 5);
106     level.close();
107 }
108
109 BOOST_AUTO_TEST_CASE(playeroffscreen) {
110     std::ifstream level("sokoban/pushdown.lvl");
111     SB::Sokoban sokoban;
```

```
    level >> sokoban;
    sokoban.movePlayer(SB::Direction::Left);
    sokoban.movePlayer(SB::Direction::Left);
    sokoban.movePlayer(SB::Direction::Left);
    BOOST_REQUIRE_EQUAL(sokoban.playerLoc().x, 0);
    BOOST_REQUIRE_EQUAL(sokoban.playerLoc().y, 2);
    level.close();
}

BOOST_AUTO_TEST_CASE(Testingmoreboxes) {
    std::ifstream level("sokoban/level5.lvl");
    SB::Sokoban sokoban;
    level >> sokoban;
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Up);
    sokoban.movePlayer(SB::Direction::Up);
    sokoban.movePlayer(SB::Direction::Up);
    sokoban.movePlayer(SB::Direction::Up);
    sokoban.movePlayer(SB::Direction::Left);
    sokoban.movePlayer(SB::Direction::Up);
    sokoban.movePlayer(SB::Direction::Right);
    BOOST_REQUIRE_EQUAL(sokoban.isWon(), true);
    level.close();
}


BOOST_AUTO_TEST_CASE(Testingmorestorage) {
    std::ifstream level("sokoban/level6.lvl");
    SB::Sokoban sokoban;
    level >> sokoban;
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Up);
    sokoban.movePlayer(SB::Direction::Up);
    sokoban.movePlayer(SB::Direction::Up);
    sokoban.movePlayer(SB::Direction::Up);
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Up);
    sokoban.movePlayer(SB::Direction::Left);
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Down);
    sokoban.movePlayer(SB::Direction::Down);
    sokoban.movePlayer(SB::Direction::Down);
    sokoban.movePlayer(SB::Direction::Down);
    sokoban.movePlayer(SB::Direction::Left);
    sokoban.movePlayer(SB::Direction::Down);
    sokoban.movePlayer(SB::Direction::Right);
    BOOST_REQUIRE_EQUAL(sokoban.isWon(), true);
    level.close();
}
```

6 ps5: DNA Sequence Alignment

6.1 Overview

For ps5 we are supposed to output the optimal DNA Alignment based on teh disatnce of the given DNA strand.

6.2 End Product

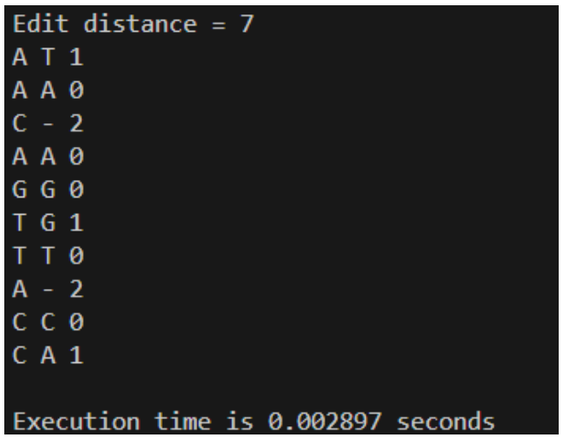

Figure 9: Output

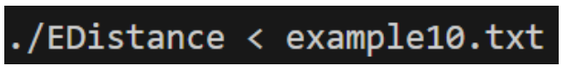

Figure 10: Input

6.3 Design and Implementation

We based the design on five functionsfive functions: penalty(), min3(), optDistance(), alignment(), and alignFormat(). The test are made to check if the functions are implemented properly. The EDistance is a class that take two string and intialize our private member variable. Strings are being compare with each other using our function penalty. That result is store in our data structure, which is a 2d array. The min3 function return the smallest list of the 3 elements. Aligment functions calculate the distance of the matrix based on the index inside our 2d vector.

6.4 What I learend

I learned how to use lamba expression to go through the DNA string.

6.5 Challenges

The main chalenge was aligment it was incorrectly inputing inaccurate values.

6.6 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++20 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework
# Your .hpp files
DEPS = EDistance.hpp
# Your compiled .o files
OBJECTS = EDistance.o
# The name of your program
PROGRAM = EDistance

STATIC_LIB = EDistance.a

.PHONY: all clean lint

all: $(PROGRAM) test




%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<

$(PROGRAM): main.o EDistance.a
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

test: test.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

main: main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

$(STATIC_LIB): $(OBJECTS)
    ar rcs $(STATIC_LIB) $@ $^ $(OBJECTS)

clean:
    rm *.o $(PROGRAM) EDistance.a test

lint:
    cpplint *.cpp *.hpp
```

EDistance.hpp:

```
// Copyright 2024 Christian Milord

#pragma once

#include <string>
#include <vector>

class EDistance {
 public:
    EDistance(const std::string& s1, const std::string& s2);

    static int penalty(char a, char b);
    static int min3(int a, int b, int c);
```

```
14
15      int optDistance();
16      std::string alignment();
17      void alignFormat(std::string& str, char a, char b);
18  private:
19      std::string str1, str2;
20      std::vector<std::vector<int>> matrix;
21  };
```

EDistance.cpp:

```
1   // Copyright 2024 Christian Milord
2
3   #include <string>
4   #include <vector>
5   #include <algorithm>
6   #include <iostream>
7   #include "EDistance.hpp"
8
9   EDistance::EDistance(const std::string& s1, const std::string& s2) {
10      str1 = s1;
11      str2 = s2;
12      std::vector<std::vector<int>> temp(str1.size() + 1, std::vector<int> (
        str2.size() + 1, 0));
13
14      for (int col = (int)str1.size(); col >= 0; col--) {
15          for (int row = (int)str2.size(); row >= 0; row--) {
16              if ((col < (int)str1.size()) && (row < (int)str2.size())) {
17                  temp[col][row] = min3(temp[col + 1][row + 1] + penalty(str1[
        col], str2[row]),
18                  temp[col + 1][row] + 2, temp[col][row + 1] + 2);
19              } else {
20                  temp[col][row] = (((int)str1.size() - col) * 2) + (((int)
        str2.size() - row) * 2);
21              }
22          }
23      }
24      std::swap(temp, matrix);
25  }
26
27  int EDistance::penalty(char a, char b) {
28      if (a == b) {
29          return 0;
30      }
31      if (a == '-' || b == '-') {
32          return 2;
33      }
34
35      return 1;
36  }
37  int EDistance::min3(int a, int b, int c) {
38      std::vector<int> list = {a, b, c};
39      return *std::min_element(list.begin(), list.end());
40  }
41
42  int EDistance::optDistance() {
43      return matrix[0][0];
44  }
45  std::string EDistance::alignment() {
46      std::string alignStr = "";
```

```
    int col = 0, row = 0;
    while ((col < (int)str1.size()) && (row < (int)str2.size())) {
        if ((matrix[col][row] == matrix[col + 1][row + 1]) && (str1[col] ==
   str2[row])){
            alignFormat(alignStr, str1[col], str2[row]);
            col++;
            row++;
        } else if ((matrix[col][row] == matrix[col + 1][row + 1] + 1) && (
   str1[col] != str2[row])) {
            alignFormat(alignStr, str1[col], str2[row]);
            col++;
            row++;
        } else if (matrix[col][row] == matrix[col + 1][row] + 2) {
            alignFormat(alignStr, str1[col], '-');
            col++;
        } else if (matrix[col][row] == matrix[col][row + 1] + 2) {
            alignFormat(alignStr, '-', str2[row]);
            row++;
        }
    }
    if ((col != (int)str1.size()) && (row == (int)str2.size())){
        for (; col < (int)str1.size(); col++){
            alignFormat(alignStr, str1[col], '-');
        }
    }
    if ((col == (int)str1.size()) && (row != (int)str2.size())){
        for (; row < (int)str2.size(); row++){
            alignFormat(alignStr, '-', str2[row]);
        }
    }
    return alignStr;
}

void EDistance::alignFormat(std::string& str, char a, char b) {
    str += a;
    str += " ";
    str += b;
    str += " ";
    str += std::to_string(penalty(a, b));
    str += "\n";
}
```

main.cpp:

```
// Copyright 2024 Christian Milord

#include <string>
#include <iostream>
#include <fstream>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include "EDistance.hpp"

int main(int argc, char* argv[]) {
    // argv[1];
    sf::Clock clock;
    std::ifstream myfile(argv[1]);
    std::string str1, str2;
    if (!myfile.is_open()) {
```

```
        std::cerr << "\nUnable to load DNA file:\nINVALID TEXT FILE\n";
    }
    myfile >> str1 >> str2;
    myfile.close();
    EDistance test(str1, str2);
    std::cout << "Edit distance = " << test.optDistance() << std::endl;
    std::cout << test.alignment() << std::endl;
    sf::Time t = clock.getElapsedTime();
    std::cout << "Execution time is " << t.asSeconds() << " seconds" << std
   ::endl;
    return 1;
}
```

test.cpp:

```
// Copyright 2022
// By Dr. Rykalova
// Editted by Dr. Daly and Christian Milord
// test.cpp for PS3
// updated 10/19/2024

#include <iostream>
#include <string>
#include <sstream>
#include "EDistance.hpp"
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>

BOOST_AUTO_TEST_CASE(testOptDistance) {
  EDistance test("AACAGTTACC", "TAAGGTCA");
  BOOST_REQUIRE_EQUAL(test.optDistance(), 7);
}


BOOST_AUTO_TEST_CASE(testAlignment) {
  EDistance testA("AACAGTTACC", "TAAGGTCA");
  BOOST_REQUIRE_EQUAL(testA.alignment(),
    "A T 1\nA A 0\nC - 2\nA A 0\nG G 0\nT G 1\nT T 0\nA - 2\nC C 0\nC A 1\n"
   );
}

BOOST_AUTO_TEST_CASE(testAlignmentSize) {
  EDistance testB("TA", "AACAGTTACC");
  BOOST_CHECK_EQUAL(testB.alignment().size(), 60);
}
```

7 ps6: Random Writer

7.1 Overview

For ps6 I was assigned to design make a RandWriter using the Markov model. The Markov model is a brute way created by Claude to generate random text according to oder of k. An order of K Markov model is K string to predict the next text. In this assigment I was assigned to implement a more efficient way to produce a random text.

7.2 End Product

7.3 Design and Implementation

In this particular project, we are making a map that has a string as a key, and the value of the map is another map. That map takes a character as a key and an integer as the value. I made the length of kgrams, our string, as a member variable. I design the constructor to go through the given string, and the amount of characters it extracts is based on size K. After that, I store the string as a key inside my map. Frequency is a function that sums up the number of times a character is followed by grams. To do that, I went through my second map and returned whatever value associated with my key. Krand produces a random character using a random number generator. Suppose the Frequency is smaller than the number randomly generated. My function will add the current Frequency with the next one and compare if the new Frequency is greater or equal to the randomly generated number. The last function, generate, takes the string and gram and adds the character that was randomly generated at the end. This makes a new string continuous inside the for loop l - k times. I developed a test to check if Frequency is working as intended by checking if Frequency returns a predetermined value of how many times a string is repeated inside our test. I test generate by comparing the size of the new string with a predetermined value. I test Krand with a simple input string "gag" and check if Kran returns a 'g' or 'a. If not, we throw an exception.

7.4 What I already knew

I aready know how to intialze and iterate through a map. I know how to overload the extraction operator to print out the newly generated word. I familiar with strings and the operation that can be preform on them.

7.5 What I learend

I learned about how a random number generator works in C++. In c, you had to design your random number generator and use the time to make sure you got a different number every time. In C++, there's a predefined generator like mt19937 that produces high-quality numbers. I learned about the Markov model of natural language. I learned about different methods of iterating through a nested map. I learned how to use the keyword throw to make an exception.

7.6 Challenges

The only challeneg I faced is figuring out on how to make text circular. I got a lot of seg fault because I'm going pass the size of the string. Sometime end of my string has some garbage value.

7.7 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework
# Your .hpp files
DEPS = RandWriter.hpp
# Your compiled .o files
OBJECTS = RandWriter.o
# The name of your program
PROGRAM = TextWriter

STATIC_LIB = TextWriter.a

.PHONY: all clean lint

all: $(PROGRAM) test




%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<

$(PROGRAM): TextWriter.o TextWriter.a
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

test: test.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

TextWriter: TextWriter.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

$(STATIC_LIB): $(OBJECTS)
    ar rcs $(STATIC_LIB) $@ $^ $(OBJECTS)

clean:
    rm -f *.o $(PROGRAM) test TextWriter.a

lint:
    cpplint *.cpp *.hpp
```

RandWriter.hpp:

```
#pragma once
#include <iostream>
#include <map>
#include <string>
// Copyright 2024 Marvens Luc

class RandWriter {
 public:
  // Create a Markov model of order k from given text
  // Assume that text has length at least k.
  RandWriter(const std::string &str, size_t k);

  size_t orderK() const;    // Order k of Markov model

```

```
  // Number of occurences of kgram in text
  // Throw an exception if kgram is not length k
  int freq(const std::string &kgram) const;
  // Number of times that character c follows kgram
  // if order=0, return num of times that char c appears
  // (throw an exception if kgram is not of length k)
  int freq(const std::string &kgram, char c) const;

  // Random character following given kgram
  // (throw an exception if kgram is not of length k)
  // (throw an exception if no such kgram)
  char kRand(const std::string &kgram);
  // Generate a string of length L characters by simulating a trajectory
  // through the corresponding Markov chain. The first k characters of
  // the newly generated string should be the argument kgram.
  // Throw an exception if kgram is not of length k.
  // Assume that L is at least k
  std::string generate(const std::string &kgram, size_t l);

 private:
  size_t _k;           // length of kgrams
  std::string _str;   // store the input
  std::map<std::string, std::map<char, int>> kfreq;
  // Private member variables go here
};
```

RandWriter.cpp:

```
#include "RandWriter.hpp"
#include <chrono>
#include <iostream>
#include <random>
#include <stdexcept>
// Copyright 2024 Marvens Luc

RandWriter::RandWriter(const std::string &str, size_t k) {
  // check the size
  if (k > str.length()) {
    throw std::invalid_argument("Not acceptable");
  }
  _k = k;
  _str = str;
  std::string circular =
      _str + _str.substr(0, k);    // does a circular % is  not needed
  for (unsigned int i = 0; i < _str.size(); i++) {
    std::string temp = circular.substr(i, k);    // get the first k charcter
    char nextchar = circular[i + k];    // gets the next character in the
   string
    // check if the string exist
    kfreq[temp][nextchar]++;     // aa a   aa a   int 1 ++
  }
}

size_t RandWriter::orderK() const { return _k; }    // Order k of Markov
   model

// Number of occurences of kgram in text
// Throw an exception if kgram is not length k
int RandWriter::freq(const std::string &kgram) const {
  int sum = 0;
```

```
  if (kgram.size() != _k) {
    throw std::invalid_argument("invalid");
  }

  auto it = kfreq.find(kgram);
  if (it != kfreq.end()) {
    for (auto const m2 : it->second) {
      sum += m2.second;
    }
    return sum;
  }
  return 0;
}
// Number of times that character c follows kgram
// if order=0, return num of times that char c appears
// (throw an exception if kgram is not of length k)
int RandWriter::freq(const std::string &kgram, char c) const {
  if (kgram.size() != _k) {
    throw std::invalid_argument("invalid");
  }
  auto it = kfreq.find(kgram);    // find kgram
  if (it != kfreq.end()) {   // kgram is found
    auto charit = it->second.find(c);   // look for c
    if (charit != it->second.end()) {    // if c is found
      return charit->second;    // return int num of frequency
    }
  }
  return 0;
}

// Random character following given kgram
// (throw an exception if kgram is not of length k)
// (throw an exception if no such kgram)
char RandWriter::kRand(const std::string &kgram) {
  if (kgram.size() < _k) {   // check kgram  < k
    throw std::invalid_argument("invalid size < k");
  }
  auto it = kfreq.find(kgram);   // find freq of k
  if (it == kfreq.end()) {        // if kgram doesn't exist invalid
    throw std::invalid_argument("invalid kgram doesn't exist");
  }

  int totalfeq = freq(kgram);   // store total frequency
  std::mt19937 gen(std::chrono::system_clock::now()
                       .time_since_epoch()
                       .count());    // member function
  std::uniform_int_distribution<int> dist(1, totalfeq);
  int cumlativefeq = 0;
  int ran = dist(gen);
  for (auto const &m2 :
       kfreq[kgram]) {   // go through the second map int based on kgram
    cumlativefeq += m2.second;    // adds the cumlative frequency
    if (cumlativefeq >= ran) {     // compare it to ran
      return m2.first;     // return the charcter
    }
  }
  throw std::runtime_error("Didn't find a letter");
}

```

```cpp
// Generate a string of length L characters by simulating a trajectory
// through the corresponding Markov chain. The first k characters of
// the newly generated string should be the argument kgram.
// Throw an exception if kgram is not of length k.
// Assume that L is at least k
std::string RandWriter::generate(const std::string &kgram, size_t l) {
  if (kgram.size() < _k) {
    throw std::invalid_argument("invalid size < k");
  }
  if (l < _k) {
    throw std::invalid_argument("invalid l < K");
  }
  std::string storedkgram = kgram;    //  string
  for (unsigned int i = 0; i < l - _k; i++) {
    char morechar = kRand(storedkgram.substr(i, _k));    // aa + k
    storedkgram += morechar;
  }
  return storedkgram;
}
```

TextWriter.cpp:

```cpp
#include "RandWriter.hpp"
#include <SFML/Graphics.hpp>
#include <fstream>
#include <iostream>
#include <sstream>
// Copyright 2024 Marvens Luc

int main(int argc, char *argv[]) {
  size_t k = std::atoi(argv[1]);
  size_t L = std::atoi(argv[2]);

  // Read from stdin and accumulate the content into a stringstream
  std::stringstream fileinput;
  std::string input;
  while (std::getline(std::cin, input)) {
    fileinput << input << '\n';   // Append each line with a newline
  }

  // Convert the accumulated content from stringstream to a string
  std::string fullInput = fileinput.str();

  // Initialize the RandWriter with the full input and k-gram order
  RandWriter writer(fullInput, k);

  // Extract the first k characters to form the k-gram
  std::string kgram = fullInput.substr(0, k);

  // Generate a string of length L using the k-gram
  std::string gentex = writer.generate(kgram, L);

  // Print the generated text
  std::cout << gentex << std::endl;

  return 0;
}
```

8 ps7: Kronos log

8.1 Overview

Project 7 is based on reading boot-up time and an end from a Kronos inTouch device. I need to use regular expressions to get this information. I also need the time that the associated server started, completion, and line number. Once we have this information, I need to output it in a file with the date, the amount of servers started, and completion. I need to output the difference between server started and completion.

8.2 End Product

```
=== Device boot ==
4(device5_intouch.log): 2013-05-04 05:28:13 Boot Start
**** Incomplete boot ****

=== Device boot ==
175(device5_intouch.log): 2013-10-26 07:45:12 Boot Start
**** Incomplete boot ****

=== Device boot ==
388(device5_intouch.log): 2013-11-01 12:28:32 Boot Start
**** Incomplete boot ****

=== Device boot ==
418(device5_intouch.log): 2013-11-01 12:31:02 Boot Start
**** Incomplete boot ****

=== Device boot ==
449(device5_intouch.log): 2013-11-01 13:01:47 Boot Start
```

Figure 11: First 18 lines of report

8.3 Design and Implementation

TThe design and Implementation were pretty straightforward. The first step I did was to make a regular expression to look for time, date, server started, and completion. Once the regular expression was completed, I used getline to read the lines of the files. I compare the content of the line to my regular expression. I made an enum call status; it organized the data by storing the time, line number, and status into one structure. I made a deque of type status to manage the data. Once we got our necessary data, I made a condition that checks if we have a server start and completion. If we do, that means we have a complete result. The code would basically be if data in the deque is greater than 1. If we have two servers start, that means the report is incomplete. I will just output the server start and pop it out of my deque.

8.4 What I already knew

I already know about regular expression through my foundation of computer science class. I am familiar with reading through a file because of previous projects.

8.5 What I learend

I learned about the regex director. I know what regular expressions are, but I didn't know the syntax of a regular expression. Most of the time, I use regex. It's more in a mathematical way. I learned how the regex library manages time using posix time.

8.6 Challenges

The first challenge I encounter is that my program was reading extra line in my file. The problem was my regex for server start wasn't right, It was reading other instance of server start the doesn't begin with 166. My date and time was outputing with a T in the middle. So I seperated the date and time into different variable using the posix time for time and gregorian for date.

8.7 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework
# Your .hpp files
DEPS =
# Your compiled .o files
OBJECTS =
# The name of your program
PROGRAM = ps7

.PHONY: all clean lint

all: $(PROGRAM)


$(PROGRAM): main.o
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

main: main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<

clean:
    rm *.o $(PROGRAM)

lint:
    cpplint *.cpp *.hpp
```

main.cpp:

```
// Copyright 2024 Marvens Luc
#include <algorithm>
#include <deque>
#include <fstream>
#include <iostream>
#include <regex>
#include <string>
#include <boost/date_time.hpp>

enum Status { START, COMPLETED };
struct KronoStatus {
    Status status;
    boost::posix_time::ptime timestamp;
    int line;
};

int main(int argc, char* argv[]) {
    KronoStatus temp;
    int line_number = 0;
    std::string line;

    std::ifstream log_file((std::string)(argv[1]));  // opens the file thats
    p
```

```
23      if (!log_file) {  // check if it's open
24          throw std::runtime_error("The log file couldn't open");
25      }
26
27      std::ofstream outfile((std::string)argv[1] + ".rpt");
28      if (!outfile) {
29          throw std::runtime_error("The log file couldn't open");
30      }
31
32      // Expressions we are searching for
33      std::regex start_up(".*166\\) server started.*");  // finds server start
34      std::regex reg_completion((".*AbstractConnector.*"));  // finds
    completion
35      std::regex time_stamp(
36          "(\\b(\\d{4}-\\d{2}-\\d{2} \\d{2}:\\d{2}:\\d{2}))");  // find the
    date and time
37      std::smatch match;
38      std::deque<KronoStatus> events;
39
40      // checking each line for regex
41      while (std::getline(log_file, line)) {
42          line_number++;
43          // look for start up or completion
44          if (std::regex_match(line, start_up)) {
45              if (std::regex_search(line, match, time_stamp)) {
46                  boost::posix_time::ptime timestamp =
47                      boost::posix_time::time_from_string(match[1].str());
48                  temp = { START, timestamp, line_number };
49                  events.push_back(temp);
50              }
51          } else if (std::regex_match(line, reg_completion)) {
52              if (std::regex_search(line, match, time_stamp)) {
53                  boost::posix_time::ptime timestamp =
54                      boost::posix_time::time_from_string(match[1].str());
55                  temp = { COMPLETED, timestamp, line_number };
56                  events.push_back(temp);
57              }
58          }
59
60          if (events.size() > 1) {
61              // start and completed together
62              if (events[0].status == START && events[1].status == COMPLETED)
    {
63                  boost::gregorian::date date;
64                  boost::posix_time::time_duration ptime;
65                  outfile << "=== Device boot == " << std::endl;
66                  boost::posix_time::time_duration duration =
67                      events[1].timestamp - events[0].timestamp;
68                  date = events[0].timestamp.date();
69                  ptime = events[0].timestamp.time_of_day();
70                  std::string string_date = boost::gregorian::
    to_iso_extended_string(date);
71                  std::string string_time = boost::posix_time::
    to_simple_string(ptime);
72                  outfile << events[0].line << "(" << (std::string)argv[1] <<
    "): " << string_date
73                          << " " << string_time << " Boot Start" << std::endl;
74                  date = events[1].timestamp.date();
75                  ptime = events[1].timestamp.time_of_day();
```

```
76                  string_date = boost::gregorian::to_iso_extended_string(date)
    ;
77                  string_time = boost::posix_time::to_simple_string(ptime);
78                  outfile << events[1].line << "(" << (std::string)argv[1] <<
    "): " << string_date
79                          << " " << string_time << " Boot Completed" << std::
    endl;
80                  outfile << "\tBoot Time"
81                          << ": " << duration.total_milliseconds() << "ms\n\n"
    ;
82                  events.clear();
83                  // start and start together
84              } else if (events[0].status == START && events[1].status ==
    START) {
85                  outfile << "=== Device boot == " << std::endl;
86                  boost::gregorian::date date;
87                  boost::posix_time::time_duration ptime;
88                  date = events[0].timestamp.date();
89                  ptime = events[0].timestamp.time_of_day();
90                  std::string string_date = boost::gregorian::
    to_iso_extended_string(date);
91                  std::string string_time = boost::posix_time::
    to_simple_string(ptime);
92                  outfile << events[0].line << "(" << (std::string)argv[1] <<
    "): " << string_date
93                          << " " << string_time << " Boot Start" << std::endl;
94                  outfile << "**** Incomplete boot ****\n\n";
95                  events.pop_front();  // pop the first start
96              } else {  // unhandled conditon
97                  std::cout << "start " << events[0].line << " " << events[0].
    status << " "
98                            << events[0].timestamp << std::endl;
99                  std::cout << "complete " << events[1].line << " " << events
    [1].status << " "
100                           << events[1].timestamp << std::endl;
101                 throw std::runtime_error("UNKOWN CONDITION");
102                 log_file.close();
103                 outfile.close();
104             }
105         }
106     }
107     // close files
108     log_file.close();
109     outfile.close();
110     return 0;
111 }
```