Computing IV Sec 202: Project Portfolio

Alan Van

Spring 2025

Contents

Time to Complete Portfolio: 18 hours

1 PS0: Hello SFML

1.1 Discussion

This project served as the introduction into the course, where we set up the build environment, which included getting familiar with the `Simple and Fast Multimedia Library (SFML)`, and the use of `Makefiles`. I set up a Linux environment and installed all the necessary packages, including `SFML` on my 2023 `MacBook Pro`.

For the actual project, we were tasked to create a simple interactive program that displayed 2 sprites, a static green circle, and an additional feature of our choice. I chose to create a blue rectangle that could be moved around the screen using input keys. The sprite could be moved around with the WASD keys, arrow keys, and reset with the spacebar, demonstrating keyboard input and sprite manipulation.

Additionally, for extra credit and to further enhance the functionality of the program, I added a sound effect using the SFML audio module, `sf:Sound`, which was activated whenever the sprite was moving. I also implemented basic boundaries to keep the sprite within the screen. This project in general helped me test and implement a variety of SFML capabilities, including rendering, audio playback, and real-time input which came to be beneficial for the rest of the course.

1.2 Core Components

The core component for completing this project was SFML, specially the event loop that continuously checked for keystrokes and were then processed accordingly:

```
1 sf::Event event;
2 while (window.pollEvent(event)) {
3     if (event.type == sf::Event::Closed)
4         window.close();
5 }
```

This event loop then updated the sprite's position based on the time elapsed between frames using `sf::Clock` and then rendered the new updated scene. To control the sprite's speed and ensure smooth movement of the sprite, the elapsed time was used to scale movement as seen here:

```
1 float time = clock.restart().asSeconds();
2 sf::Vector2f movement(0.f, 0.f);
3
4 if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left) || sf::Keyboard::
     isKeyPressed(sf::Keyboard::A)) {
5     movement.x -= speed * time;
6 }
7
8 rectangle.move(movement);
```

Additionally, boundary checks were implemented to prevent the sprite from moving off-screen, keeping the sprite within the window borders. This was done by checking whether the sprite's position exceeded the window size. The bounds for this project are unfortunately imperfect, having visual bugs in the top corners, and the bottom boundary does not fully work to keep the sprite within the window. Sound effects were also added and were triggered whenever movement occurred.

1.3 What I accomplished

- Set up build environment for course.
- Installed and configured SFML on macOS.
- Wrote a C++ program using SFML to render and move a sprite on screen.
- Added movement controls (WASD + arrow keys) and a reset feature with the spacebar.

- Added a sound effect that plays whenever the sprite moves.
- Implemented boundaries to limit sprite movement to the screen.

1.4 What I already knew

Prior to starting the project, I already had experience writing basic `C++` programs and using object-oriented programming (`OOP`) from previous computing courses. I was also comfortable with command-line compilation and using tools like `g++`.

1.5 What I learned

I learned how to use the SFML multimedia library for real-time graphics and sound, including how to load textures, control sprites, and play sound effects. I also learned how to use `sf::Clock` to control movement timing and how to handle user input using SFML's event polling system. This project was also my introduction into Makefiles, which I would have to use for every project going throughout the year.

1.6 Challenges

This project was fairly basic but, I still faced some challenges, specifically with the bounds being able to fully restrict the sprite's movement. Although most of the boundaries worked as expected, the bottom boundary does not work and there is a minor visual bug where when moving the sprite diagonally into the upper corners which causes flickering or some sort of slight visual distortion.
Despite these challenges, the program is fully functional.

1.7 Evidence

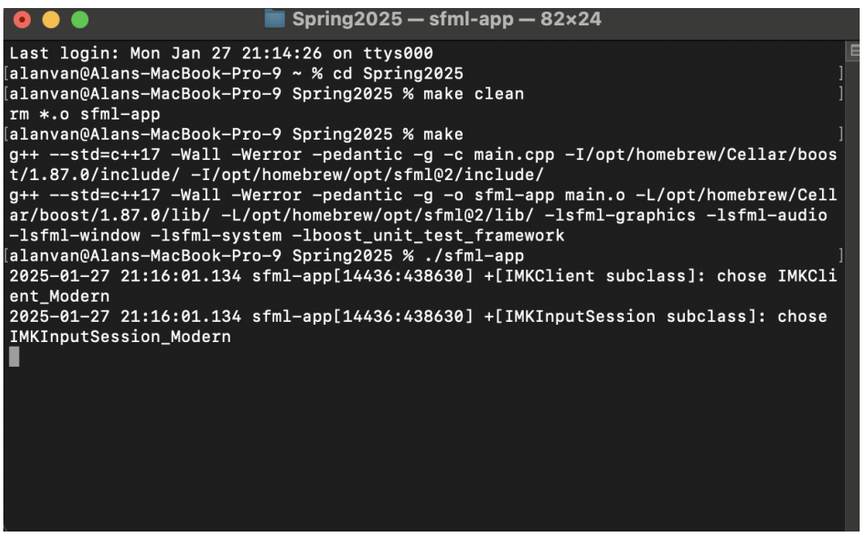


Figure 1: Screenshot showing code running

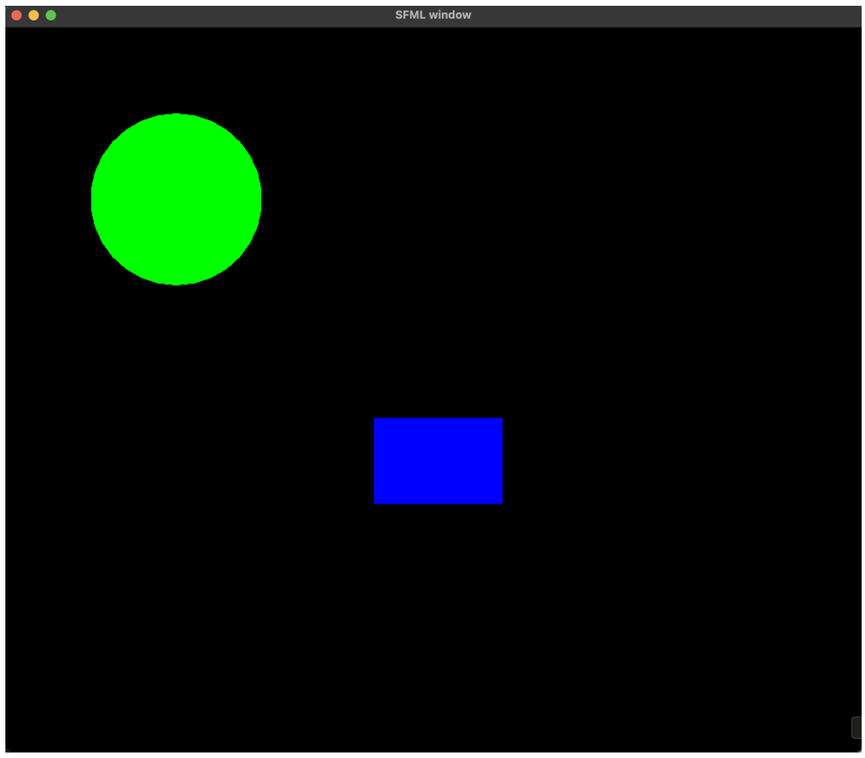


Figure 2: Output showing SFML sprites in window

1.8 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
INCLUDEDIR = -I/opt/homebrew/Cellar/boost/1.87.0/include/ -I/opt/homebrew/opt/sfml@2/include/
LIBDIR = -L/opt/homebrew/Cellar/boost/1.87.0/lib/ -L/opt/homebrew/opt/sfml@2/lib/

# 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 $< $(INCLUDEDIR)

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

clean:
    rm *.o $(PROGRAM)

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

main.cpp:

```cpp
// Copyright [2025] Alan Van

#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>

int main() {
    // Movement speed (250 pixels)
    const float speed = 250.f;
    // Create the main window (1000x1000 pixels)
    sf::RenderWindow window(sf::VideoMode(1000, 1000), "SFML window");
    // Green circle
    sf::CircleShape circle(100.f);
    circle.setFillColor(sf::Color::Green);
    circle.setPosition(100, 100);
    // Sprite (blue rectangle)
    sf::RectangleShape rectangle(sf::Vector2f(150.f, 100.f));
    rectangle.setFillColor(sf::Color::Blue);
    rectangle.setPosition(500, 500);
    sf::Clock clock;    // Clock for smooth movement
    // Toy whistle sound
    sf::SoundBuffer buffer;
    if (!buffer.loadFromFile("toy_whistle.wav")) {
        return -1;
    }
    sf::Sound sound;
    sound.setBuffer(buffer);
    // Loop for window & sprite
    while (window.isOpen()) {
        float time = clock.restart().asSeconds();
        bool moved = false;
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window.close();
        }
        sf::Vector2f movement(0.f, 0.f);
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)
            || sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {
            movement.x -= speed * time;
            moved = true;
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)
            || sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {
            movement.x += speed * time;
            moved = true;
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)
            || sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {
            movement.y -= speed * time;
            moved = true;
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)
            || sf::Keyboard::isKeyPressed(sf::Keyboard::S)) {
            movement.y += speed * time;
            moved = true;
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space)) {
```

```
            rectangle.setPosition(500, 500);
            moved = true;
        }
        if (moved && sound.getStatus() != sf::Sound::Playing) {
            sound.play();
        }
        rectangle.move(movement);
        // Boundaries for window (so sprite cant leave)
        sf::Vector2f position = rectangle.getPosition();
        sf::Vector2f size = rectangle.getSize();
        if (position.x < 0)
            rectangle.setPosition(0, position.y);
        if (position.y < 0)
            rectangle.setPosition(position.x, 0);
        if (position.x > 1000 - size.x)
            rectangle.setPosition(1000 - size.x, position.y);
        if (position.y > 1000 - size.y)     // Doesn't work, can still go
    down beyond window)
            rectangle.setPosition(position.x, 1000 - size.y);
        // Render
        window.clear();
        window.draw(circle);
        window.draw(rectangle);
        window.display();
    }
    return 0;
}
```

2 PS1: LFSR / PhotoMagic

2.1 Discussion

For this project, we were tasked with creating a program that produced pseudo-random bits by simulating a 16-bit `linear-feedback shift register (LFSR)` and using it to perform simple image encryption. The LFSR was initialized with given binary seed string that could encrypt and decrypt an image. The project was split into two parts which included:

(`Part A`) Creating and testing a `FibLFSR` class that could generate an 8-bit pseudo-random number using bitwise (`XOR`) operations. This involved implementing `Boost` unit tests, the `step()` function, the `generate()` function, and overloading the left shift (`<<`) operator for output.

(`Part B`) Using the `FibLFSR` class implemented from `Part A` to create the `PhotoMagic` class, which encrypted and decrypted an image pixel by pixel by modifying each pixel's color with Boolean exclusive-or (`XOR`) operations. This involved implementing the `transform()` function and main driver program. `SFML` was used for image loading and display and `Boost` was used for unit testing.

2.2 Core Components

The core component for completing this project was the simulation of a 16-bit Fibonacci linear-feedback shift register (`FibLFSR`). The LFSR updates its internal state by shifting all bits one position to the left and calculating the new rightmost bit as the XOR of bits at specific tap positions (10, 12, and 13). This core component was implemented in the `step()` function shown here:

```
int newBit = (registerBits[0] - '0') ^
             (registerBits[2] - '0') ^
             (registerBits[3] - '0') ^
             (registerBits[5] - '0');
registerBits = registerBits.substr(1) + std::to_string(newBit);
return newBit;
```

This allowed the LFSR to produce a reproducible yet random bit sequence from a given binary seed. Additionally, the `generate(k)` function added more to this by simulating `k` steps and building an integer from the resulting bits:

```
if (k <= 0) {
    throw std::invalid_argument("k must be positive.");
}
int result = 0;
for (int i = 0; i < k; i++) {
    result = (result << 1) | step();
}
return result;
```

In Part B, the LFSR was used to encrypt an image through the `transform()` function. For each pixel in the input image, the red, green, and blue components were each XORed with a newly generated 8-bit value, 'encrypting' the image using the LFSR's pseudo-random output:

```
sf::Color pixel = image.getPixel(x, y);
pixel.r ^= static_cast<uint8_t>(lfsr->generate(8));
pixel.g ^= static_cast<uint8_t>(lfsr->generate(8));
pixel.b ^= static_cast<uint8_t>(lfsr->generate(8));
image.setPixel(x, y, pixel);
```

Using the same binary seed a second time on the encrypted image 'decrypted' the image due to XOR's reversibility, which effectively created a basic encryption & decryption program.

2.3 What I accomplished

- Implemented a working `FibLFSR` class that simulated a 16-bit Fibonacci LFSR and tested it with various seed values.

- Implemented and tested the `generate()` function that returned 8-bit pseudo-random values.
- Implemented the `transform()` function that modified image pixels using FibLFSR.
- Used SFML to load, encrypt, decrypt and display images.
- Overloaded the left shift operator (`<<`) to print the current LFSR register states.
- Wrote unit tests to verify correctness of seed handling and LFSR behavior.
- Fixed the Makefile and added a `main.cpp` driver program to ensure proper compilation.

2.4 What I already knew

Prior to starting this project, I was already comfortable with writing C++ classes and basic file structure involving `.hpp` and `.cpp` files. I had a general idea of what LFSRs were and general cryptography knowledge from a previous course. I also had used SFML in PS0, which gave me a foundation for using it here as well.

2.5 What I learned

This project taught me how to integrate `unit test cases` using `Boost`, I also got more familiar with fixing file organization issues in C++, especially on macOS where certain compiler behaviors differ. This was also the first time I applied XOR in a practical context like image encryption which was very interesting.

2.6 Challenges

This was one of the assignments I resubmitted for additional credit. Initially, I failed the test portion of Part A, scoring 0/9, due to an incorrect submission. I later corrected the issues by adding a missing `main.cpp` file, which had caused build problems, and by updating the Makefile to properly include `$(PROGRAM)` under the `all` target. I also added a new test case to `test.cpp` to verify that `generate()` correctly handles invalid input.
Despite these challenges, the program is fully functional.

2.7 Evidence

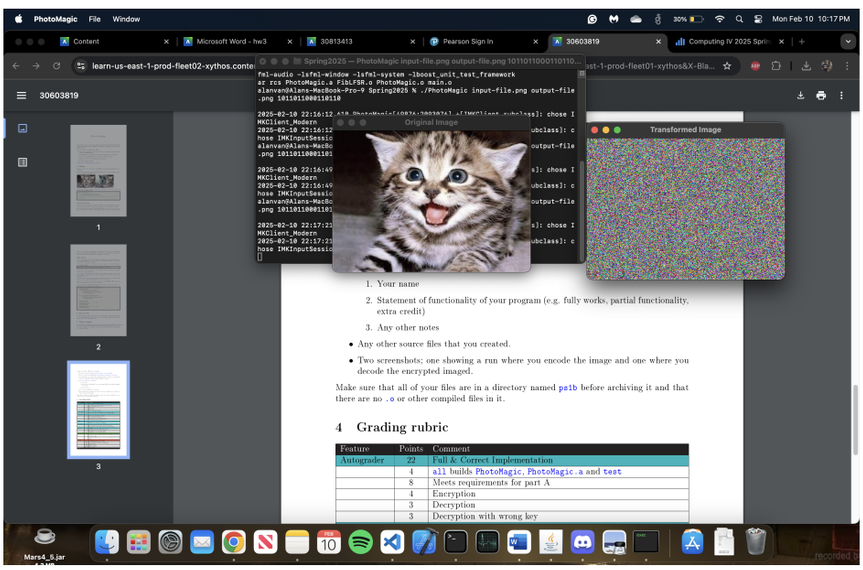


Figure 3: Original image being encoded

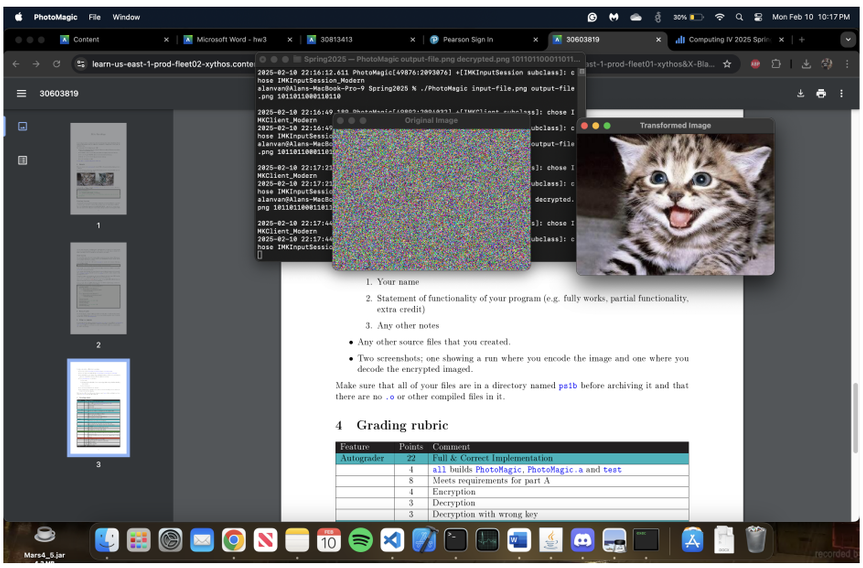


Figure 4: Encrypted image being decoded

2.8 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
INCLUDEDIR = -I/opt/homebrew/Cellar/boost/1.87.0/include/ -I/opt/homebrew/opt/sfml@2/include/
LIBDIR = -L/opt/homebrew/Cellar/boost/1.87.0/lib/ -L/opt/homebrew/opt/sfml@2/lib/

# Your .hpp files
DEPS = FibLFSR.hpp PhotoMagic.hpp
# Your compiled .o files
OBJECTS = FibLFSR.o PhotoMagic.o main.o
# The name of your program
PROGRAM = PhotoMagic
TEST_PROGRAM = test
STATIC_LIB = PhotoMagic.a

.PHONY: all clean lint

all: $(PROGRAM) $(TEST_PROGRAM) $(STATIC_LIB)

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

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

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

$(TEST_PROGRAM): test.o FibLFSR.o
    $(CC) $(CFLAGS) -o $@ $^ $(LIBDIR) $(LIB)

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

```

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

main.cpp:

```
// Copyright [2025] Alan Van

#include <iostream>
#include <SFML/Graphics.hpp>
#include "PhotoMagic.hpp"

int main(int argc, char* argv[]) {
    if (argc != 4) {
        std::cerr << "Usage: " << argv[0]
            << " <input-file.png> <output-file.png> <16-bit LFSR seed>\n";
        return 1;
    }
    std::string inputFile = argv[1];
    std::string outputFile = argv[2];
    std::string seed = argv[3];
    if (seed.length() != 16 || seed.find_first_not_of("01") != std::string::
   npos) {
        std::cerr << "Error: Seed must be a 16-bit binary string.\n";
        return 1;
    }
    sf::Image image;
    if (!image.loadFromFile(inputFile)) {
        std::cerr << "Error: Could not load input image.\n";
        return 1;
    }
    PhotoMagic::FibLFSR lfsr(seed);
    PhotoMagic::transform(image, &lfsr);
    if (!image.saveToFile(outputFile)) {
        std::cerr << "Error: Could not save output image.\n";
        return 1;
    }
    sf::Texture textureOriginal, textureTransformed;
    textureOriginal.loadFromFile(inputFile);
    textureTransformed.loadFromFile(outputFile);
    sf::Sprite spriteOriginal(textureOriginal);
    sf::Sprite spriteTransformed(textureTransformed);
    sf::RenderWindow windowOriginal(sf::VideoMode(textureOriginal.getSize().
   x,
        textureOriginal.getSize().y), "Original Image");
    sf::RenderWindow windowTransformed(sf::VideoMode(textureTransformed.
   getSize().x,
        textureTransformed.getSize().y), "Transformed Image");
    while (windowOriginal.isOpen() && windowTransformed.isOpen()) {
        sf::Event event;
        while (windowOriginal.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                windowOriginal.close();
            }
        while (windowTransformed.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                windowTransformed.close();
            }
    windowOriginal.clear();
    windowOriginal.draw(spriteOriginal);
    windowOriginal.display();
    windowTransformed.clear();
```

```
    windowTransformed.draw(spriteTransformed);
    windowTransformed.display();
    }

    return 0;
}
```

FibLFSR.hpp:

```
// Copyright [2025] Alan Van

#ifndef FIBLFSR_HPP
#define FIBLFSR_HPP

#include <iostream>
#include <string>

namespace PhotoMagic {
class FibLFSR {
 public:
    explicit FibLFSR(const std::string& seed);
    explicit FibLFSR(unsigned int seed);  // Optional

    static FibLFSR fromPassword(const std::string& password);  // Optional

    int step();
    int generate(int k);

    friend std::ostream& operator<<(std::ostream& os, const FibLFSR& lfsr);

 private:
    std::string registerBits;
};
}  // namespace PhotoMagic

#endif
```

FibLFSR.cpp:

```
// Copyright [2025] Alan Van

#include "FibLFSR.hpp"
#include <string>
#include <stdexcept>
#include <iostream>

namespace PhotoMagic {
// Constructor to create LFSR with the given initial seed
FibLFSR::FibLFSR(const std::string& seed) {
    if (seed.length() != 16) {
        throw std::invalid_argument("Seed must be 16 bits long");
    }
    if (seed.find_first_not_of("01") != std::string::npos) {
        throw std::invalid_argument("Seed must only contain '0's or '1's.");
    }
    registerBits = seed;
}
// Simulate one step and return the new bit as 0 or 1
int FibLFSR::step() {
    int newBit = (registerBits[0] - '0') ^
                 (registerBits[2] - '0') ^
                 (registerBits[3] - '0') ^
```

```
                (registerBits[5] - '0');
    registerBits = registerBits.substr(1) + std::to_string(newBit);
    return newBit;
}
// Simulate k steps and return a k-bit integer
int FibLFSR::generate(int k) {
    if (k <= 0) {
        throw std::invalid_argument("k must be positive.");
    }
    int result = 0;
    for (int i = 0; i < k; i++) {
        result = (result << 1) | step();
    }
    return result;
}
// Any fields that you need
std::ostream& operator<<(std::ostream& os, const PhotoMagic::FibLFSR& lfsr)
   {
    os << lfsr.registerBits;
    return os;
}
};  // namespace PhotoMagic
```

PhotoMagic.hpp:

```
// Copyright [2025] Alan Van

#ifndef PHOTOMAGIC_HPP
#define PHOTOMAGIC_HPP

#pragma once

#include <SFML/Graphics.hpp>
#include "FibLFSR.hpp"

namespace PhotoMagic {
void transform(sf::Image& img, FibLFSR* lfsr);
}   // namespace PhotoMagic

#endif
```

PhotoMagic.cpp:

```
// Copyright [2025] Alan Van

#include "PhotoMagic.hpp"

namespace PhotoMagic {
void transform(sf::Image &image, FibLFSR *lfsr) {
    sf::Vector2u size = image.getSize();
    for (unsigned int x = 0; x < size.x; x++) {
        for (unsigned int y = 0; y < size.y; y++) {
            sf::Color pixel = image.getPixel(x, y);
            pixel.r ^= static_cast<uint8_t>(lfsr->generate(8));
            pixel.g ^= static_cast<uint8_t>(lfsr->generate(8));
            pixel.b ^= static_cast<uint8_t>(lfsr->generate(8));
            image.setPixel(x, y, pixel);
        }
    }
}
}   // namespace PhotoMagic
```

pixels.cpp:

```cpp
// pixels.cpp:
// using SFML to load a file, manipulate its pixels, write it to disk


// g++ -o pixels pixels.cpp -lsfml-graphics -lsfml-window

#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

int main()
{
    sf::Image image;
    if (!image.loadFromFile("cat.jpg"))
        return -1;

    // p is a pixelimage.getPixel(x, y);
    sf::Color p;

    // create photographic negative image of upper-left 200 px square
    for (int x = 0; x<200; x++) {
        for (int y = 0; y< 200; y++) {
            p = image.getPixel(x, y);
            p.r = 255 - p.r;
            p.g = 255 - p.g;
            p.b = 255 - p.b;
            image.setPixel(x, y, p);
        }
    }

    sf::Vector2u size = image.getSize();
    sf::RenderWindow window(sf::VideoMode(size.x, size.y), "Meow");

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

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

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

        window.clear(sf::Color::White);
        window.draw(sprite);
        window.display();
    }

    // fredm: saving a PNG segfaults for me, though it does properly
    //   write the file
    if (!image.saveToFile("cat-out.bmp"))
        return -1;

```

```
    return 0;
}
```

test.cpp:

```
// Copyright 2022
// By Dr. Rykalova
// Editted by Dr. Daly
// test.cpp for PS1a
// updated 1/8/2024

#include <iostream>
#include <string>

#include "FibLFSR.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);
}

BOOST_AUTO_TEST_CASE(testGenerateInstr) {
  FibLFSR l("1011011000110110");
  BOOST_REQUIRE_EQUAL(l.generate(9), 51);
}

// Test << operator
BOOST_AUTO_TEST_CASE(testStreamInsertion) {
  FibLFSR l("1011011000110110");
  std::ostringstream os;
  os << l;
  BOOST_CHECK_EQUAL(os.str(), "1011011000110110");
}

// Test invalid seed input cases
BOOST_AUTO_TEST_CASE(testInvalidInputs) {
  BOOST_CHECK_THROW(FibLFSR("101"), std::invalid_argument);
  BOOST_CHECK_THROW(FibLFSR("10110110001101101010101010"), std::
    invalid_argument);
  BOOST_CHECK_THROW(FibLFSR("ABCDEFGHIJKLMNOP"), std::invalid_argument);
  BOOST_CHECK_THROW(FibLFSR("101101100011011O"), std::invalid_argument);
}

// Test step() to see if it returns to its inital seed after a cycle
BOOST_AUTO_TEST_CASE(testStepCycle) {
  std::string initial_seed = "1011011000110110";
  FibLFSR l(initial_seed);
  for (int i = 0; i < 16; i++) {
```

```
56        l.step();
57    }
58    std::ostringstream os;
59    os << l;
60    BOOST_REQUIRE_EQUAL(os.str(), initial_seed);
61  }
62
63  // Test generate() with a large k value
64  BOOST_AUTO_TEST_CASE(testGenerateLargeK) {
65    FibLFSR l("1011011000110110");
66    BOOST_REQUIRE_EQUAL(l.generate(20), 123);
67  }
68
69  // Test initializing with an empty seed
70  BOOST_AUTO_TEST_CASE(testEmptySeed) {
71    BOOST_CHECK_THROW(FibLFSR(""), std::invalid_argument);
72  }
```

3 PS2: Triangle Fractal

3.1 Discussion

This project involved implementing a Triangle Fractal, similar to a Sierpinski triangle. The goal of this project was to implement the fractal recursively using the `fractal()` function, with a main driver program to execute the program. At each level of recursion, the triangle is divided into smaller sub-triangles, producing a repeating visual pattern.
This project had the option for pair programming, where you can work with a partner but I opted out, doing it alone.

3.2 Core Components

The core component for completing this project was recursion, each recursive call had to calculate the center point of a smaller triangle and run `fractal()` on each of the three child positions. Each child triangle was calculated using geometric offsets from the parent. Each child was placed to the left, right, and bottom of the parent triangle and was half the size of the parent.
The height h of an equilateral triangle with side length L was calculated as:

$$h = \frac{\sqrt{3}}{2} \cdot L$$

At each level of recursion, the triangle's side length was halved like:

$$L_n = \frac{L}{2^n}$$

The vertical offset for positioning of the child triangles was based on:

$$\Delta y = \frac{h}{2} = \frac{\sqrt{3}}{4} \cdot L$$

Math translated into code here:

```
1 double leftX = center.x - (length / 2);
2 double leftY = center.y - diff - (length / 2) / sqrt(3);
3 double rightX = center.x + (length / 2) + (length / 4);
4 double rightY = center.y - diff + (height / 2) - (length / 2) / sqrt(3);
5 double bottomX = center.x - (length / 4);
6 double bottomY = center.y + radius + (height / 2) - (length / 2) / sqrt(3);
```

To render the triangle fractal, `sf::CircleShape` was use with three points to draw an equilateral triangle centered at the desired location. The triangle was rotated 180 degrees and was filled with a color determined by the recursion depth to create a pattern:

```
1 sf::CircleShape triangle(radius, 3);
2 triangle.setOrigin(radius, radius);
3 triangle.setPosition(center);
4
5 triangle.setRotation(180);
6 triangle.setFillColor(colorDepth);
7 window.draw(triangle);
```

Additionally, the color of each triangle was computed using a gradient formula based on the depth level. This added visual distinction and was included as an extra credit feature.

3.3 What I accomplished

- Implemented a recursive `fractal()` function to generate a consistent triangle fractal pattern.
- Rendered the fractal using SFML's basic drawing functions.
- Adjusted the recursion logic to dynamically scale the triangles and prevent clipping.

- Switched from using `ConvexShape` to direct window drawing for simpler and more efficient rendering.
- For extra credit, implemented different colors in a consistent manner to the fractal.
- Resubmitted for additional credit.

3.4 What I already knew

Prior to starting this project, I had a moderate amount of experience with recursion and basic C++ graphics using `SFML` from previous projects in this course. I also had some exposure to using vectors and coordinates in graphics from earlier coursework.

3.5 What I learned

This project taught me how to draw directly onto a render window using `SFML` and helped me understand the recursive subdivision visually. I also practiced writing basic unit test cases, even though testing was not the main focus of this problem set, I always had problems with the test cases.

3.6 Challenges

This was another one of the assignments I resubmitted for additional credit. I mainly focused on implementing the easier fixes that lost me points, such as a more descriptive `Readme` and fixing the recursion depth, which was off by one.

On the actual project, I initially tried using SFML's `ConvexShape` class to draw the triangles, but had difficulty managing its points and rendering and decided to switch to drawing directly on the window using vertex coordinates, which I figured to be easier. I also had issues wrapping my head around the actual math of the triangle fractal with its corner positions and overlapping triangles, which took some effort, but was doable.
Despite these challenges, the program is fully functional.

3.7 Evidence

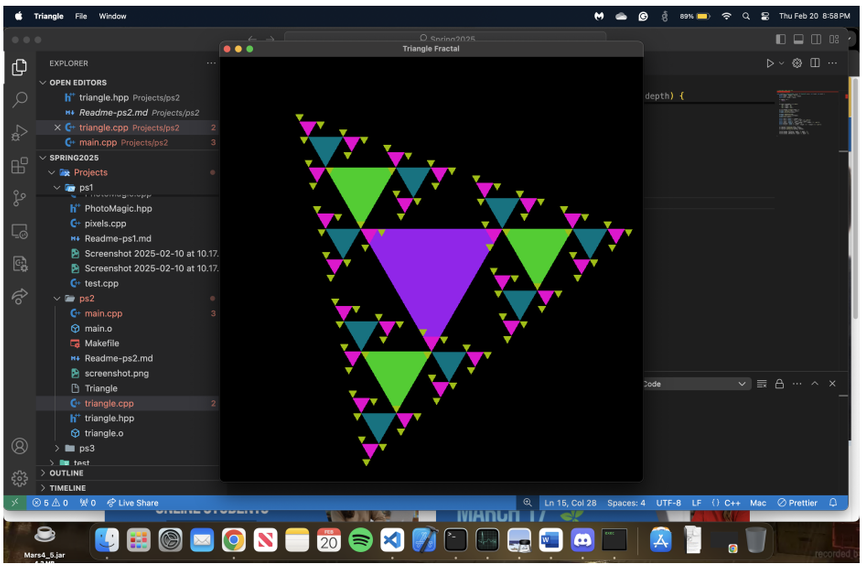


Figure 5: Fractal image displayed using SFML in a window

3.8 Codebase

Makefile:

```
1 CC = g++
2 CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
3 LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
    lboost_unit_test_framework
```

```
INCLUDEDIR = -I/opt/homebrew/Cellar/boost/1.87.0/include/ -I/opt/homebrew/opt/sfml@2/include/
LIBDIR = -L/opt/homebrew/Cellar/boost/1.87.0/lib/ -L/opt/homebrew/opt/sfml@2/lib/

# Your .hpp files
DEPS = triangle.hpp
# Your compiled .o files
OBJECTS = triangle.o main.o
# The name of your program
PROGRAM = Triangle

.PHONY: all clean lint

all: $(PROGRAM) $(TEST_PROGRAM) $(STATIC_LIB)

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

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

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

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

main.cpp:

```
// Copyright [2025] Alan Van
#include "triangle.hpp"
#include <SFML/Graphics.hpp>

int main(int argc, char* argv[]) {
    if (argc != 3) {
        std::cerr << "Usage: " << argv[0] << " L N" << std::endl;
        std::cerr << "L: Length of the base triangle side (double)" << std::endl;
        std::cerr << "N: Depth of recursion (int)" << std::endl;
        return 0;
    }

    double L = std::stod(argv[1]) * 0.5;
    int N = std::stoi(argv[2]);

    int windowSize = static_cast<int>(L * 3);
    sf::RenderWindow window(sf::VideoMode(windowSize, windowSize), "Triangle Fractal");

    sf::Vector2f center(windowSize / 2, windowSize / 2);

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

        window.clear();
        fractal(window, center, L, N);
```

```
        window.display();
    }

    return 1;
}
```

triangle.hpp:

```
// Copyright [2025] Alan Van
#ifndef TRIANGLE_HPP
#define TRIANGLE_HPP

#include <cmath>
#include <iostream>
#include <cstdlib>
#include <SFML/Graphics.hpp>

void fractal(sf::RenderTarget& window, sf::Vector2f center, int length, int
    depth);

#endif
```

triangle.cpp:

```
// Copyright [2025] Alan Van
#include "triangle.hpp"

void fractal(sf::RenderTarget& window, sf::Vector2f center, int length, int
    depth) {
    double radius = length / sqrt(3);
    double height = (sqrt(3) / 2) * length;
    double diff = height - radius;

    sf::Color colorDepth = sf::Color(
        100 + (depth * 60),
        100 + (depth * 90),
        100 + (depth * 180));

    sf::CircleShape triangle(radius, 3);
    triangle.setOrigin(radius, radius);
    triangle.setPosition(center);

    triangle.setRotation(180);
    triangle.setFillColor(colorDepth);
    window.draw(triangle);

    if (depth == 0) {
        return;
    }

    double leftX = center.x - (length / 2);
    double leftY = center.y - diff - (length / 2) / sqrt(3);
    double rightX = center.x + (length / 2) + (length / 4);
    double rightY = center.y - diff + (height / 2) - (length / 2) / sqrt(3);
    double bottomX = center.x - (length / 4);
    double bottomY = center.y + radius + (height / 2) - (length / 2) / sqrt
    (3);

    sf::Vector2f leftCorner(leftX, leftY);
    sf::Vector2f rightCorner(rightX, rightY);
    sf::Vector2f bottomCorner(bottomX, bottomY);

```

```
37      fractal(window, leftCorner, length / 2, depth - 1);
38      fractal(window, rightCorner, length / 2, depth - 1);
39      fractal(window, bottomCorner, length / 2, depth - 1);
40  }
```

4 PS3: N-Body Simulation

4.1 Discussion

This project involved implementing a N-body simulation to model gravitational interactions between celestial bodies in space. This project was split up into two parts which included:

(`Part A`) Setting up the UI for an orbital planetary simulator. Being able to read a universe file describing the number of bodies, their masses, positions, and velocities, then displaying each celestial body, which was represented by an image through `SFML`.

(`Part B`) Adding physics to the orbital planetary simulator, reading the universe file and updating their motion over time using physical equations.

4.2 Core Components

The core element for completing this project were smart pointers, which were helpful in managing the lifetime of my objects. The code was structured around two primary classes: `CelestialBody` and `Universe`. The `CelestialBody` class managed the position, velocity, mass, and texture of each body, and the `Universe` class maintained the collection of bodies, simulated their gravitational interactions, and rendering them onto the screen.

Smart pointers, specifically (`std::shared_ptr`) was used to manage each `sf::Texture` to ensure that the texture data remained the same whenever assigned to a `sf::Sprite`. Without this, planets may appear as white boxes due to storing a pointer instead of the data. Each body stores its texture like this:

```
1 _texture(std::make_shared<sf::Texture>()) {}
```

The universe was initialized by reading an input file that contained data of the number of celestial bodies, the universe's radius, and the initial conditions of each body. This data was parsed through using an overloaded extraction operator in `Universe.cpp`:

```
1  std::istream& operator>>(std::istream& is, Universe& uni) {
2      size_t n;
3      if (!(is >> n >> uni._radius)) {
4          is.setstate(std::ios::failbit);
5          return is;
6      }
7
8      uni._bodies.clear();
9
10     for (size_t i = 0; i < n; i++) {
11         CelestialBody body;
12         if (!(is >> body)) {
13             is.setstate(std::ios::failbit);
14             break;
15         }
16         uni._bodies.push_back(body);
17     }
18     return is;
19 }
```

The celestial bodies were rendered by transforming the coordinates from physical units to the window and centering its origin to the middle of the window. This allowed bodies with very large coordinates to be correctly visualized on screen:

```
1 sf::Transform transform;
2 transform.translate(windowSize.x / 2.0f, windowSize.y / 2.0f);
3 transform.scale(scale, -scale);
4 states.transform *= transform;
```

The core physics of the simulation was implemented in `Universe::step()`, using *Newton's law of universal gravitation*. For each of the celestial bodies, the gravitational force was calculated by:

$$F = G \cdot \frac{m_1 m_2}{r^2}$$

Where G is the gravitational constant, m_1 and m_2 are the body masses, and r is the distance between them.

Force is then broken into its x and y components:

$$F_x = F \cdot \frac{\Delta x}{r}, \quad F_y = F \cdot \frac{\Delta y}{r}$$

Then acceleration is calculated using Newton's second law:

$$a_x = \frac{F_x}{m}, \quad a_y = \frac{F_y}{m}$$

This logic was translated into code as follows:

```
1 double fx = (G * m1 * m2 / r^2) * (dx / r);
2 double ax = fx / m;
3 vx += dt * ax * 0.5;
4 x  += dt * (vx + dt * ax * 0.5);
5 vx += dt * ax * 0.5;
```

This implementation ensured that the orbit remained smooth and accurate, portraying a realistic gravitational movement of celestial bodies in an simulated universe.

4.3 What I accomplished

- Implemented the physics for gravitational interactions between multiple celestial bodies.
- Loaded textures and simulated movement using SFML.
- Used overloaded input/output operators to parse and print the universe file data.
- Used smart pointers to ensure textures persisted correctly across objects.
- Ensured scaling and transformations positioned planets were correct on screen.

4.4 What I already knew

Prior to starting this project, I had experience with `SFML` graphics from earlier projects, and a decent understanding of object-oriented programming in C++. I was also familiar with using vectors and working with basic physics equations like force and acceleration.

4.5 What I learned

I learned how to structure a larger simulation using multiple classes and how to use smart pointers like `std::shared_ptr` for managing textures. I also got more comfortable overloading operators to simplify input and output handling. Debugging rendering issues (like invisible planets due to small scale) helped me understand coordinate transformations and how to make the simulation visually accurate.

4.6 Challenges

This was the last of the assignments I resubmitted for additional credit. I fixed input handling in the `operator»` function for `CelestialBody` and corrected output formatting in `operator«`. I also fixed the universe scaling and flipping issue by adjusting the scale factor using `scaleFactor = windowSize.x / (2.0 * _radius)`, ensuring planets rendered correctly within the window. Additionally, I addressed several test failures to match expected output.

For the actual project, I had challenges with smart pointers, where in this project, I implemented a `std::shared_ptr` for textures to ensure proper resource management when copying objects. Initial issues also included planets being too small to render, incorrect coordinate transformations, failing test cases, and challenges with managing shared pointers. These fixes significantly improved the project's functionality and reliability.
Despite these challenges, the program is fully functional.

4.7 Evidence

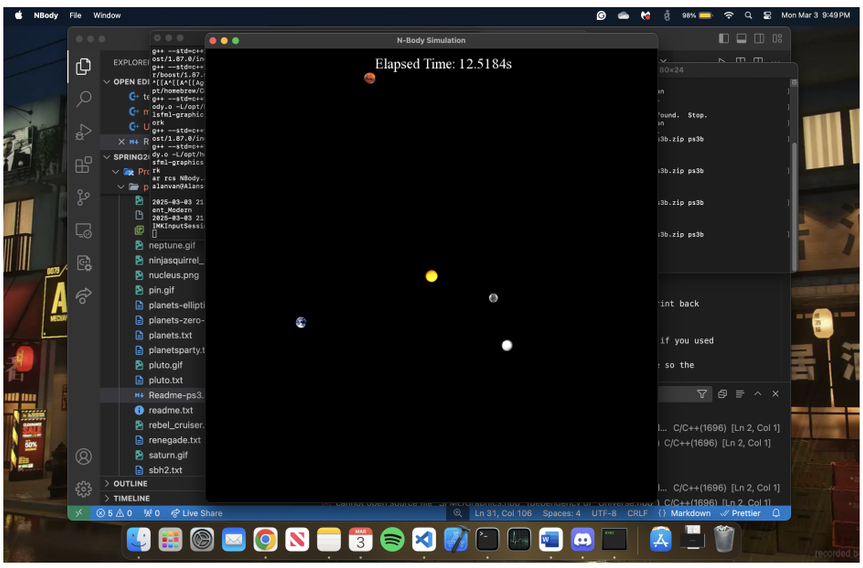


Figure 6: N-Body Simulation output showing celestial bodies rendered with SFML

4.8 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
INCLUDEDIR = -I/opt/homebrew/Cellar/boost/1.87.0/include/ -I/opt/homebrew/
    opt/sfml@2/include/
LIBDIR = -L/opt/homebrew/Cellar/boost/1.87.0/lib/ -L/opt/homebrew/opt/sfml@2
    /lib/

# Your .hpp files
DEPS = Universe.hpp CelestialBody.hpp
# Your compiled .o files
OBJECTS = Universe.o CelestialBody.o
# The name of your program
PROGRAM = NBody
TEST_PROGRAM = test
STATIC_LIB = NBody.a

.PHONY: all clean lint

all: $(PROGRAM) $(TEST_PROGRAM) $(STATIC_LIB)

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

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

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

$(TEST_PROGRAM): test.o Universe.o CelestialBody.o
    $(CC) $(CFLAGS) -o $@ $^ $(LIBDIR) $(LIB)

clean:
```

```
    rm -f *.o $(PROGRAM) $(TEST_PROGRAM) $(STATIC_LIB)

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

main.cpp:

```cpp
// Copyright [2025] Alan Van
#include "Universe.hpp"
#include <SFML/Graphics.hpp>

int main() {
    NB::Universe universe;
    if (!(std::cin >> universe)) {
        std::cerr << "Error reading universe data." << std::endl;
        return 1;
    }

    sf::RenderWindow window(sf::VideoMode(800, 800), "N-Body Simulation");
    window.setFramerateLimit(60);

    sf::Clock clock;

    sf::Font font;
    if (!font.loadFromFile("times.ttf")) {
        std::cerr << "Error loading font." << std::endl;
        return 1;
    }

    sf::Text elapsedTimeText;
    elapsedTimeText.setFont(font);
    elapsedTimeText.setCharacterSize(25);
    elapsedTimeText.setFillColor(sf::Color::White);
    elapsedTimeText.setPosition(300, 10);

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

    float elapsedTime = clock.getElapsedTime().asSeconds();
    universe.step(25000.0);

    std::ostringstream timeStream;
    timeStream << "Elapsed Time: " << elapsedTime << "s";
    elapsedTimeText.setString(timeStream.str());

    window.clear(sf::Color::Black);

    for (size_t i = 0; i < universe.size(); i++) {
        window.draw(universe[i]);
    }

    window.draw(elapsedTimeText);

    window.display();
    }
    return 0;
}
```

CelestialBody.hpp:

```
// Copyright [2025] Alan Van
#pragma once

#include <iostream>
#include <memory>
#include <iomanip>
#include <SFML/Graphics.hpp>

namespace NB {
class CelestialBody: public sf::Drawable {
 public:
    double _x, _y;
    double _vx, _vy;
    explicit CelestialBody();    // Required

    sf::Vector2f position() const;  // Optional
    sf::Vector2f velocity() const;  // Optional
    float mass() const;    // Optional

 protected:
    void draw(sf::RenderTarget& window,
      sf::RenderStates states) const override;    // From sf::Drawable
 private:
    // Fields and helper methods go here
    double _mass;

    std::shared_ptr<sf::Texture> _texture;
    std::vector<std::shared_ptr<CelestialBody>> _bodies;
    sf::Sprite _sprite;
    std::string _imageFileName;

    friend std::istream& operator>>(std::istream& is, CelestialBody& body);
    friend std::ostream& operator<<(std::ostream& os, const CelestialBody&
   body);
};
}  // namespace NB
```

CelestialBody.cpp:

```
// Copyright [2025] Alan Van
#include "CelestialBody.hpp"

namespace NB {

CelestialBody::CelestialBody()
:_x(0.0), _y(0.0),
_vx(0.0), _vy(0.0),
_mass(0.0),
_texture(std::make_shared<sf::Texture>()) {}

sf::Vector2f CelestialBody::position() const {
    return sf::Vector2f(static_cast<float>(_x), static_cast<float>(_y));
}

sf::Vector2f CelestialBody::velocity() const {
    return sf::Vector2f(static_cast<float>(_vx), static_cast<float>(_vy));
}

```

```cpp
float CelestialBody::mass() const {
    return static_cast<float>(_mass);
}

void CelestialBody::draw(sf::RenderTarget& target, sf::RenderStates states)
   const {
    sf::Sprite sprite = _sprite;

    float scaleFactor = 400.0f / 2.50e+11;
    sprite.setPosition(static_cast<float>(_x * scaleFactor) + 400.0f,
                       -static_cast<float>(_y * scaleFactor) + 400.0f);

    target.draw(sprite, states);
}

std::istream& operator>>(std::istream& is, CelestialBody& body) {
    std::string imageFileName;
    if (!(is >> body._x >> body._y >> body._vx >> body._vy >> body._mass >>
   imageFileName)) {
        is.setstate(std::ios::failbit);
        return is;
    }
    body._imageFileName = imageFileName;

    if (!body._texture->loadFromFile(imageFileName)) {
        is.setstate(std::ios::failbit);
        return is;
    }
    body._sprite.setTexture(*body._texture, true);
    sf::FloatRect bounds = body._sprite.getLocalBounds();
    body._sprite.setOrigin(bounds.width / 2, bounds.height / 2);
    return is;
}

std::ostream& operator<<(std::ostream& os, const CelestialBody& body) {
    os << std::scientific << std::setprecision(6)
       << body._x << " " << body._y << " "
       << body._vx << " " << body._vy << " "
       << body._mass << " " << body._imageFileName;
    return os;
}

}  // namespace NB
```

Universe.hpp:

```cpp
// Copyright [2025] Alan Van

#pragma once

#include <iostream>
#include <fstream>
#include <vector>
#include <stdexcept>
#include <sstream>
#include <cmath>
#include <SFML/Graphics.hpp>
#include "CelestialBody.hpp"

namespace NB {
class Universe: public sf::Drawable {
```

```
 public:
    Universe();   // Required
    explicit Universe(const std::string& filename);   // Optional

    size_t size() const;   // Optional
    double radius() const;    // Optional

    const CelestialBody& operator[](size_t i) const;  // Optional

    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:
    // Fields and helper functions go here
    double _radius;
    std::vector<CelestialBody> _bodies;

    friend std::istream& operator>>(std::istream& is, Universe& uni);
    friend std::ostream& operator<<(std::ostream& os, const Universe& uni);
};
}  // namespace NB
```

Universe.cpp:

```
// Copyright [2025] Alan Van
#include "Universe.hpp"
#include <SFML/Graphics.hpp>

namespace NB {

Universe::Universe() : _radius(0.0) {}

Universe::Universe(const std::string& filename) : _radius(0.0) {
    std::ifstream file(filename);
    if (!file.is_open()) {
        throw std::runtime_error("Failed to open file " + filename);
    }
    file >> *this;
}

size_t Universe::size() const {
    return _bodies.size();
}

double Universe::radius() const {
    return _radius;
}

const CelestialBody& Universe::operator[](size_t i) const {
    return _bodies[i];
}

void Universe::step(double dt) {
    const double G = 6.67e-11;
    size_t n = _bodies.size();

    std::vector<sf::Vector2<double>> forces(n, sf::Vector2<double>(0.0, 0.0)
   );

```

```
35      for (size_t i = 0; i < n; i++) {
36          for (size_t j = 0; j < n; j++) {
37              if (i == j)
38                  continue;
39              const CelestialBody& body1 = _bodies[i];
40              const CelestialBody& body2 = _bodies[j];
41              double dx = body2.position().x - body1.position().x;
42              double dy = body2.position().y - body1.position().y;
43              double r = std::sqrt(dx * dx + dy * dy);
44              if (r < 1e3)
45                  r = 1e3;
46
47              double magnitude = (G * body1.mass() * body2.mass()) / (r * r);
48              double fx = magnitude * (dx / r);
49              double fy = magnitude * (dy / r);
50              forces[i].x += fx;
51              forces[i].y += fy;
52          }
53      }
54
55      for (size_t i = 0; i < n; i++) {
56          CelestialBody& body = _bodies[i];
57          double ax = forces[i].x / body.mass();
58          double ay = forces[i].y / body.mass();
59
60          body._vx += dt * ax * 0.5;
61          body._vy += dt * ay * 0.5;
62
63          body._x += dt * (body._vx + dt * ax * 0.5);
64          body._y += dt * (body._vy + dt * ay * 0.5);
65
66          body._vx += dt * ax * 0.5;
67          body._vy += dt * ay * 0.5;
68      }
69  }
70
71  void Universe::draw(sf::RenderTarget& target, sf::RenderStates states) const
        {
72      sf::Vector2u windowSize = target.getSize();
73      float scale = static_cast<float>(windowSize.x / (2.0 * _radius));
74      scale *= 1e9;
75
76      sf::Transform transform;
77      transform.translate(windowSize.x / 2.0f, windowSize.y / 2.0f);
78      transform.scale(scale, -scale);
79      states.transform *= transform;
80
81      for (const auto& body : _bodies) {
82          target.draw(body, states);
83      }
84  }
85
86  std::istream& operator>>(std::istream& is, Universe& uni) {
87      size_t n;
88      if (!(is >> n >> uni._radius)) {
89          is.setstate(std::ios::failbit);
90          return is;
91      }
92      uni._bodies.clear();
```

```
    for (size_t i = 0; i < n; i++) {
        CelestialBody body;
        if (!(is >> body)) {
            is.setstate(std::ios::failbit);
            break;
        }
        uni._bodies.push_back(body);
    }
    return is;
}

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

}  // namespace NB
```

test.cpp:

```
// Copyright [2025] Alan Van
#include <fstream>
#include "Universe.hpp"
#include "CelestialBody.hpp"

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

BOOST_AUTO_TEST_CASE(testTxtFile) {
    std::ifstream file("1body.txt");
    BOOST_REQUIRE(file.is_open());
}

BOOST_AUTO_TEST_CASE(testCelestialBodyGetters) {
    std::string inputLine = "1.4960e+11 0.0000e+00 0.0000e+00 2.9800e+04
   5.9740e+24 earth.gif";
    std::istringstream iss(inputLine);
    NB::CelestialBody body;
    iss >> body;
    BOOST_CHECK_CLOSE(body.position().x, 1.4960e+11, 0.001);
    BOOST_CHECK_CLOSE(body.position().y, 0.0, 0.001);
    BOOST_CHECK_CLOSE(body.velocity().x, 0.0, 0.001);
    BOOST_CHECK_CLOSE(body.velocity().y, 2.9800e+04, 0.001);
    BOOST_CHECK_CLOSE(body.mass(), 5.9740e+24, 0.001);
}

BOOST_AUTO_TEST_CASE(testUniverseGetters) {
    std::string inputData =
        "2 2.0e+12\n1.4960e+11 0.0000e+00 0.0000e+00 2.9800e+04 5.9740e+24
   earth.gif\n"
        "2.2790e+11 0.0000e+00 0.0000e+00 2.4100e+04 6.4190e+23 mars.gif\n";
    std::istringstream iss(inputData);
    NB::Universe uni;
    iss >> uni;
    BOOST_CHECK_EQUAL(uni.size(), 2);
    BOOST_CHECK_CLOSE(uni.radius(), 2.0e+12, 0.001);
    const NB::CelestialBody& earth = uni[0];
```

```
37      BOOST_CHECK_CLOSE(earth.position().x, 1.4960e+11, 0.001);
38      BOOST_CHECK_CLOSE(earth.position().y, 0.0, 0.001);
39      const NB::CelestialBody& mars = uni[1];
40      BOOST_CHECK_CLOSE(mars.position().x, 2.2790e+11, 0.001);
41      BOOST_CHECK_CLOSE(mars.position().y, 0.0, 0.001);
42  }
43
44  BOOST_AUTO_TEST_CASE(testUniverseAltConstructor) {
45      try {
46          NB::Universe uni("1body.txt");
47          BOOST_CHECK(uni.size() > 0);
48      } catch (const std::exception& e) {
49          BOOST_FAIL(e.what());
50      }
51  }
52
53  BOOST_AUTO_TEST_CASE(testInsertionOperator) {
54      std::istringstream iss(
55          "1 2.50e+11\n"
56          "1.4960e+11 0.0000e+00 0.0000e+00 2.9800e+04 5.9740e+24 earth.gif\n"
        );
57      NB::Universe uni;
58
59      BOOST_REQUIRE_NO_THROW(iss >> uni);
60      BOOST_REQUIRE(!iss.fail());
61
62      std::ostringstream oss;
63      BOOST_REQUIRE_NO_THROW(oss << uni);
64
65      std::ostringstream expectedStream;
66      expectedStream << uni;
67
68      BOOST_TEST(oss.str() == expectedStream.str());
69  }
70
71  BOOST_AUTO_TEST_CASE(testStepFunction) {
72      const double G = 6.67e-11;
73      std::string inputData =
74          "2 2.0e+12\n"
75          "1.4960e+11 0.0000e+00 0.0000e+00 2.9800e+04 5.9740e+24 earth.gif\n"
76          "0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 1.9890e+30 sun.gif\n";
77
78      std::istringstream iss(inputData);
79      NB::Universe uni;
80      iss >> uni;
81
82      double timeStep = 25000.0;
83
84      const NB::CelestialBody& earthBefore = uni[0];
85      const NB::CelestialBody& sunBefore = uni[1];
86
87      double expectedAccelX = -(G * sunBefore.mass()) /
88          (earthBefore.position().x * earthBefore.position().x);
89      double expectedVelocityX = earthBefore.velocity().x + timeStep *
        expectedAccelX;
90
91      uni.step(timeStep);
92
93      const NB::CelestialBody& earthAfter = uni[0];
```

```
94
95      BOOST_CHECK_CLOSE(earthAfter.velocity().x, expectedVelocityX, 0.01);
96
97      double expectedNewPosX = earthBefore.position().x + timeStep *
        expectedVelocityX;
98      BOOST_CHECK_CLOSE(earthAfter.position().x, expectedNewPosX, 0.01);
99  }
100
101 BOOST_AUTO_TEST_CASE(testElapsedTime) {
102     sf::RenderWindow window(sf::VideoMode(800, 800), "N-Body Simulation");
103     sf::Font font;
104     font.loadFromFile("times.ttf");
105
106     sf::Text elapsedTimeText;
107     elapsedTimeText.setFont(font);
108     elapsedTimeText.setCharacterSize(20);
109     elapsedTimeText.setFillColor(sf::Color::White);
110     elapsedTimeText.setPosition(10, 10);
111
112     sf::Clock clock;
113
114     for (int i = 0; i < 5; i++) {
115         float elapsedSeconds = clock.getElapsedTime().asSeconds();
116         std::ostringstream timeStream;
117         timeStream << "Elapsed Time: " << elapsedSeconds << " s";
118         elapsedTimeText.setString(timeStream.str());
119
120         BOOST_TEST(elapsedTimeText.getString().toAnsiString().find("Elapsed
        Time: ")
121         != std::string::npos);
122     }
123 }
124
125 BOOST_AUTO_TEST_CASE(testFixedDeltas) {
126     double T = 157788000.0;
127     double dt = 25000.0;
128
129     std::istringstream iss(
130         "2 2.0e+12\n"
131         "1.4960e+11 0.0000e+00 0.0000e+00 2.9800e+04 5.9740e+24 earth.gif\n"
132         "0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 1.9890e+30 sun.gif\n");
133     NB::Universe uni;
134     BOOST_REQUIRE_NO_THROW(iss >> uni);
135
136     double elapsedTime = 0.0;
137
138     while (elapsedTime < T) {
139         double prevTime = elapsedTime;
140         uni.step(dt);
141         elapsedTime += dt;
142
143         BOOST_CHECK_CLOSE(elapsedTime - prevTime, dt, 0.001);
144     }
145 }
146
147 BOOST_AUTO_TEST_CASE(testInvertedAxis) {
148     std::istringstream iss(
149         "1 2.50e+11\n"
150         "1.4960e+11 1.0000e+11 0.0000e+00 0.0000e+00 5.9740e+24 earth.gif\n"
```

```
    );
    NB::Universe uni;
    BOOST_REQUIRE_NO_THROW(iss >> uni);

    const NB::CelestialBody& body = uni[0];

    double expectedX = 1.4960e+11;
    double expectedY = 1.0000e+11;

    BOOST_CHECK_CLOSE(body.position().x, expectedX, 0.001);
    BOOST_CHECK_CLOSE(body.position().y, expectedY, 0.001);

    sf::Sprite sprite;
    float scaleFactor = 400.0f / 2.50e+11;

    float expectedRenderX = static_cast<float>(expectedX * scaleFactor) +
    400.0f;
    float expectedRenderY = -static_cast<float>(expectedY * scaleFactor) +
    400.0f;

    sprite.setPosition(expectedRenderX, expectedRenderY);

    BOOST_CHECK_CLOSE(sprite.getPosition().x, expectedRenderX, 0.001);
    BOOST_CHECK_CLOSE(sprite.getPosition().y, expectedRenderY, 0.001);
}

BOOST_AUTO_TEST_CASE(testLeapfrog) {
    const double G = 6.67e-11;
    double dt = 25000.0;

    std::istringstream iss(
        "2 2.0e+12\n"
        "1.4960e+11 0.0000e+00 0.0000e+00 2.9800e+04 5.9740e+24 earth.gif\n"
        "0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 1.9890e+30 sun.gif\n");
    NB::Universe uni;
    BOOST_REQUIRE_NO_THROW(iss >> uni);

    const NB::CelestialBody& earthBefore = uni[0];
    const NB::CelestialBody& sunBefore = uni[1];

    double dx = earthBefore.position().x - sunBefore.position().x;
    double r = std::sqrt(dx * dx);
    double expectedAccelX = (G * sunBefore.mass()) / (r * r);

    if (dx > 0) expectedAccelX = -expectedAccelX;

    double expectedVelocityX = earthBefore.velocity().x + dt *
    expectedAccelX * 0.5;

    double newPosX = earthBefore.position().x + dt * expectedVelocityX;

    expectedVelocityX += dt * expectedAccelX * 0.5;

    BOOST_REQUIRE_NO_THROW(uni.step(dt));

    const NB::CelestialBody& earthAfter = uni[0];

    BOOST_CHECK_CLOSE(earthAfter.velocity().x, expectedVelocityX, 0.01);

```

```
206     BOOST_CHECK_CLOSE(earthAfter.position().x, newPosX, 0.01);
207 }
```

5 PS4: Sokoban

5.1 Discussion

This project was focused on implementing a Sokoban-style game using `SFML`. This project was split up into two parts which included:

(`Part A`) Setting up the UI for the block-pushing game by reading level data from a text file and displaying the game using textured mapped to characters such as wall, boxes, targets, and the player on a window.

(`Part B`) Adding game mechanics to the game, included player movement, box-pushing mechanics, boundaries, win conditions, and reset functionality.

5.2 Core Components

The core component for completing this project was the use of a 2D vector of characters to represent the game board and a parallel structure to track storage tiles. The game logic read level data into these vectors, then used `SFML` sprites to render each tile by stacking textures in the correct draw order. Each tile was rendered on top of a ground layer to ensure that textures were stacked in the correct order

Movement was handled by interpreting user input and updating the board accordingly. Boxes could only be pushed onto valid empty tiles, and the player could only move if no obstacles blocked the path. To move the player and push boxes, the program used offset logic with bounds and collision checks shown here:

```
if (isBox(dest)) {
    int nnx = nx + dx;
    int nny = ny + dy;
    if (nnx < 0 || nny < 0 || nnx >= static_cast<int>(_width) || nny >=
   static_cast<int>(_height))
        return;

    char& beyond = _board[nny * _width + nnx];
    if (!isEmpty(beyond)) return;

    beyond = (_storageMap[nny * _width + nnx]) ? '1' : 'A';
    dest = (_storageMap[ny * _width + nx]) ? 'a' : '.';
}

_playerPos = {static_cast<unsigned>(nx), static_cast<unsigned>(ny)};
```

Each tile on the board was rendered based on its type. The game always drew a ground tile first, and if a tile was part of the goal zone, a storage sprite was drawn on top. Then, a second layer handled rendering of boxes, walls, and the player:

```
sf::Sprite base;
base.setTexture(_textures[1]);
base.setPosition(x * TILE_SIZE, y * TILE_SIZE);
window.draw(base, states);

if (_storageMap[idx]) {
    sf::Sprite storage;
    storage.setTexture(_textures[4]);
    storage.setPosition(x * TILE_SIZE, y * TILE_SIZE);
    window.draw(storage, states);
}
```

Additionally, a reset feature was added by pressing the `'R'` key, and a move counter and win message were added as extra credit. The move counter stayed static on the screen, updating every time the player moved the character and once all storage tiles were filled with boxes, a victory message would be rendered in the center of the screen.

5.3 What I accomplished

- Successfully loaded Sokoban level files and rendered them using SFML.
- Implemented game mechanics including movement, box-pushing, collision detection, and level completion detection.
- Added a win message when the player completes the level.
- Included a move counter and reset functionality.
- Used operator overloading to support reading/writing level data to and from files.

5.4 What I already knew

The core element for completing this project was the level data, which was represented as a `2D vector` of strings, where each element corresponded to a specific tile type (wall, box, target, player, or ground). `SFML` sprites were used to draw each tile, ensuring visual consistency across the board. To maintain correct layering, every tile was first rendered with a ground texture, followed by any additional object textures (such as walls or boxes) drawn on top.

5.5 What I learned

I learned how to better manage tile-based rendering, especially layering textures. I also learned to apply lambda functions with the STL, such as in the `movePlayer()` logic to check tile states. Additionally, I learned to work with 2D vectors more effectively and structure game logic in a modular way.

5.6 Challenges

Some of the challenges I faced were linker errors when trying to output a file named "Sokoban," which conflicted with an existing file and caused build issues. I also found that implementing movement and collision logic was tricky but manageable with proper testing. Debugging the push mechanics and wall collisions required a lot of trial and error to ensure the player and boxes behaved correctly under all movement scenarios.
Despite these challenges, the program is fully functional.

5.7 Evidence

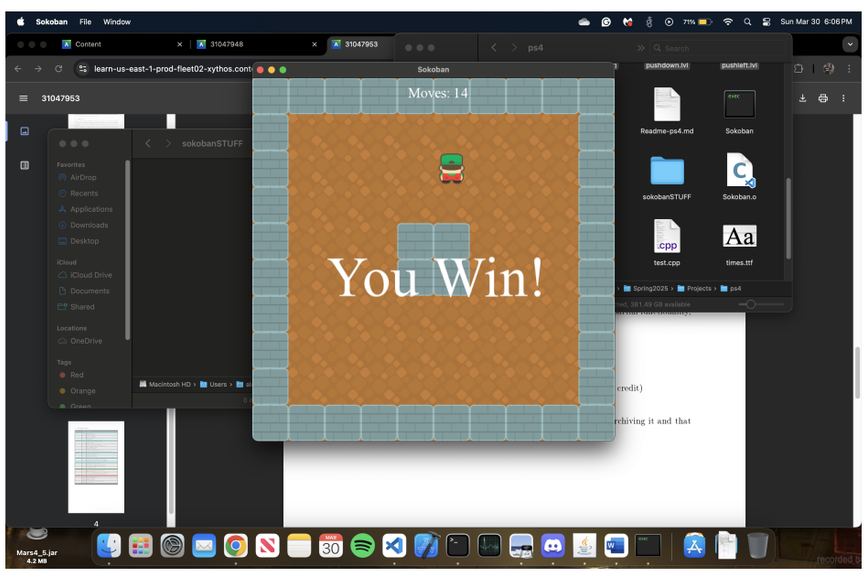


Figure 7: Sokoban UI displaying player, boxes, and goal tiles

5.8 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
INCLUDEDIR = -I/opt/homebrew/Cellar/boost/1.87.0/include/ -I/opt/homebrew/
    opt/sfml@2/include/
LIBDIR = -L/opt/homebrew/Cellar/boost/1.87.0/lib/ -L/opt/homebrew/opt/sfml@2
    /lib/

# Your .hpp files
DEPS = Sokoban.hpp
# Your compiled .o files
OBJECTS = Sokoban.o
# The name of your program
PROGRAM = Sokoban

STATIC_LIB = Sokoban.a

.PHONY: all clean lint

all: $(PROGRAM) $(TEST_PROGRAM) $(STATIC_LIB)

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

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

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

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

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

main.cpp:

```
// Copyright [2025] Alan Van
#include "Sokoban.hpp"

int main(int argc, char* argv[]) {
    std::ifstream levelFile(argv[1]);
    if (!levelFile) {
        std::cerr << "Error opening level file." << std::endl;
        return 1;
    }

    SB::Sokoban game;
    levelFile >> game;

    sf::RenderWindow window(sf::VideoMode(game.pixelWidth(), game.
    pixelHeight()), "Sokoban");

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

```
                window.close();

            if (event.type == sf::Event::KeyPressed) {
                switch (event.key.code) {
                    case sf::Keyboard::W: game.movePlayer(SB::Direction::Up); break;
                    case sf::Keyboard::A: game.movePlayer(SB::Direction::Left); break;
                    case sf::Keyboard::S: game.movePlayer(SB::Direction::Down); break;
                    case sf::Keyboard::D: game.movePlayer(SB::Direction::Right); break;
                    case sf::Keyboard::Up: game.movePlayer(SB::Direction::Up); break;
                    case sf::Keyboard::Left: game.movePlayer(SB::Direction::Left); break;
                    case sf::Keyboard::Down: game.movePlayer(SB::Direction::Down); break;
                    case sf::Keyboard::Right: game.movePlayer(SB::Direction::Right); break;
                    case sf::Keyboard::R: game.reset(); break;
                    default: break;
                }
            }
        }

        window.clear();
        window.draw(game);
        window.display();
    }

    return 0;
}
```

Sokoban.hpp:

```
// Copyright [2025] Alan Van
#pragma once

#include <iostream>
#include <fstream>
#include <sstream>
#include <algorithm>
#include <SFML/Graphics.hpp>

namespace SB {
enum class Direction {
    Up, Down, Left, Right
};

class Sokoban : public sf::Drawable {
 public:
    static const int TILE_SIZE = 64;

    Sokoban();
    explicit Sokoban(const std::string&);  // Optional

    unsigned int pixelHeight() const;  // Optional
    unsigned int pixelWidth() const;  // Optional

    unsigned int height() const;
```

```
    unsigned int width() const;

    sf::Vector2u playerLoc() const;

    bool isWon() const;

    void movePlayer(Direction dir);
    void reset();

    void undo();  // Optional XC
    void redo();  // Optional XC

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

 private:
    // Any fields you need go here.
    std::vector<char> _board;
    unsigned int _width;
    unsigned int _height;
    sf::Vector2u _playerPos;
    std::vector<sf::Texture> _textures;

    bool _hasWon = false;

    std::vector<char> _originalBoard;
    sf::Vector2u _originalPlayerPos;
    std::vector<bool> _storageMap;
    std::size_t _moveCount = 0;

    sf::Font _font;
    sf::Text _moveText;
    sf::Text _winText;

    Direction _lastMoveDir = Direction::Down;

    friend std::ostream& operator<<(std::ostream& out, const Sokoban& s);
    friend std::istream& operator>>(std::istream& in, Sokoban& s);
};

std::ostream& operator<<(std::ostream& out, const Sokoban& s);
std::istream& operator>>(std::istream& in, Sokoban& s);
}  // namespace SB
```

Sokoban.cpp:

```
// Copyright [2025] Alan Van
#include "Sokoban.hpp"

namespace SB {

Sokoban::Sokoban() : _width(0), _height(0), _playerPos(0, 0) {
    _textures.resize(8);

    _textures[0].loadFromFile("player_05.png");   // Player (down)
    _textures[1].loadFromFile("ground_01.png");   // Empty ground
    _textures[2].loadFromFile("block_06.png");    // Wall
    _textures[3].loadFromFile("crate_03.png");    // Box
    _textures[4].loadFromFile("ground_04.png");   // Storage
    _textures[5].loadFromFile("player_08.png");   // Player (up)
```

```
15      _textures[6].loadFromFile("player_20.png");   // Player (left)
16      _textures[7].loadFromFile("player_17.png");   // Player (right)
17
18      if (!_font.loadFromFile("times.ttf")) {
19          std::cerr << "Error loading font." << std::endl;
20      }
21
22      _moveText.setFont(_font);
23      _moveText.setCharacterSize(25);
24      _moveText.setFillColor(sf::Color::White);
25      _moveText.setPosition(10, 10);
26      _moveText.setString(sf::String("Moves: ") + std::to_string(_moveCount));
27
28      _winText.setFont(_font);
29      _winText.setCharacterSize(75);
30      _winText.setFillColor(sf::Color::White);
31      _winText.setString("You Win!");
32  }
33
34  unsigned int Sokoban::width() const {
35      return _width;
36  }
37
38  unsigned int Sokoban::height() const {
39      return _height;
40  }
41
42  unsigned int Sokoban::pixelWidth() const {
43      return _width * TILE_SIZE;
44  }
45
46  unsigned int Sokoban::pixelHeight() const {
47      return _height * TILE_SIZE;
48  }
49
50  sf::Vector2u Sokoban::playerLoc() const {
51      return _playerPos;
52  }
53
54  void Sokoban::movePlayer(Direction dir) {
55      if (_hasWon) return;
56
57      int dx = 0, dy = 0;
58      switch (dir) {
59          case Direction::Up: dy = -1; break;
60          case Direction::Down: dy = 1; break;
61          case Direction::Left: dx = -1; break;
62          case Direction::Right: dx = 1; break;
63      }
64
65      int x = static_cast<int>(_playerPos.x);
66      int y = static_cast<int>(_playerPos.y);
67      int nx = x + dx;
68      int ny = y + dy;
69
70      if (nx < 0 || ny < 0 || nx >= static_cast<int>(_width) || ny >=
        static_cast<int>(_height))
71          return;
72
```

```
73      char& dest = _board[ny * _width + nx];
74      char& current = _board[y * _width + x];
75
76      auto isBox = [](char c) { return c == 'A' || c == '1'; };
77      auto isEmpty = [](char c) { return c == '.' || c == 'a'; };
78
79      if (dest == '#') return;
80
81      if (isBox(dest)) {
82          int nnx = nx + dx;
83          int nny = ny + dy;
84          if (nnx < 0 || nny < 0 || nnx >= static_cast<int>(_width)
85              || nny >= static_cast<int>(_height))
86              return;
87
88          char& beyond = _board[nny * _width + nnx];
89          if (!isEmpty(beyond)) return;
90
91          beyond = (_storageMap[nny * _width + nnx]) ? '1' : 'A';
92          dest = (_storageMap[ny * _width + nx]) ? 'a' : '.';
93      }
94
95      _playerPos = {static_cast<unsigned>(nx), static_cast<unsigned>(ny)};
96      dest = '@';
97      current = (_storageMap[y * _width + x]) ? 'a' : '.';
98
99      _moveCount++;
100     _moveText.setString("Moves: " + std::to_string(_moveCount));
101
102     _hasWon = isWon();
103     _lastMoveDir = dir;
104 }
105
106 void Sokoban::reset() {
107     _board = _originalBoard;
108     _playerPos = _originalPlayerPos;
109     _hasWon = false;
110     _moveCount = 0;
111     _moveText.setString("Moves: 0");
112 }
113
114 bool Sokoban::isWon() const {
115     for (size_t i = 0; i < _storageMap.size(); ++i) {
116         if (_storageMap[i] && _board[i] != '1') {
117             return false;
118         }
119     }
120     return true;
121 }
122
123 void Sokoban::draw(sf::RenderTarget& window, sf::RenderStates states) const
    {
124     for (size_t y = 0; y < _height; ++y) {
125         for (size_t x = 0; x < _width; ++x) {
126             size_t idx = y * _width + x;
127             char tile = _board[idx];
128
129             sf::Sprite base;
130             base.setTexture(_textures[1]);
```

```
131             base.setPosition(x * TILE_SIZE, y * TILE_SIZE);
132             window.draw(base, states);
133 
134             if (_storageMap[idx]) {
135                 sf::Sprite storage;
136                 storage.setTexture(_textures[4]);
137                 storage.setPosition(x * TILE_SIZE, y * TILE_SIZE);
138                 window.draw(storage, states);
139             }
140 
141             sf::Sprite overlay;
142             overlay.setPosition(x * TILE_SIZE, y * TILE_SIZE);
143 
144             switch (tile) {
145                 case '#':
146                     overlay.setTexture(_textures[2]);
147                     window.draw(overlay, states);
148                     break;
149                 case 'A':
150                 case '1':
151                     overlay.setTexture(_textures[3]);
152                     window.draw(overlay, states);
153                     break;
154                 case '@': {
155                     switch (_lastMoveDir) {
156                         case Direction::Up:    overlay.setTexture(_textures
    [5]); break;
157                         case Direction::Down:  overlay.setTexture(_textures
    [0]); break;
158                         case Direction::Left:  overlay.setTexture(_textures
    [6]); break;
159                         case Direction::Right: overlay.setTexture(_textures
    [7]); break;
160                     }
161                     window.draw(overlay, states);
162                     break;
163                 }
164                 default:
165                     break;
166             }
167         }
168     }
169 
170     if (_hasWon) {
171         sf::Text winText = _winText;
172         sf::FloatRect bounds = winText.getLocalBounds();
173         winText.setOrigin(bounds.width / 2.f, bounds.height / 2.f);
174         winText.setPosition(_width * TILE_SIZE / 2.f, _height * TILE_SIZE /
    2.f);
175         window.draw(winText, states);
176     }
177 
178     window.draw(_moveText, states);
179 }
180 
181 std::istream& operator>>(std::istream& in, Sokoban& s) {
182     s._board.clear();
183     s._storageMap.clear();
184 
```

```
    in >> s._height >> s._width;
    in.ignore();

    size_t size = s._width * s._height;
    s._board.resize(size, '.');
    s._storageMap.resize(size, false);

    for (size_t y = 0; y < s._height; y++) {
        std::string line;
        std::getline(in, line);
        for (size_t x = 0; x < std::min(line.size(), static_cast<size_t>(s._width)); x++) {
            char ch = line[x];
            size_t index = y * s._width + x;
            s._board[index] = ch;

            if (ch == '@') {
                s._playerPos = {static_cast<unsigned>(x), static_cast<unsigned>(y)};
            }
            if (ch == 'a' || ch == '1') {
                s._storageMap[index] = true;
            }
        }
    }

    s._originalBoard = s._board;
    s._originalPlayerPos = s._playerPos;

    return in;
}

std::ostream& operator<<(std::ostream& out, const Sokoban& s) {
    out << s._height << " " << s._width << "\n";
    for (size_t y = 0; y < s._height; y++) {
        for (size_t x = 0; x < s._width; x++) {
            out << s._board[y * s._width + x];
            }
        out << "\n";
        }

    return out;
}
}   // namespace SB
```

test.cpp:

```
// Copyright [2025] Alan Van
#include "Sokoban.hpp"

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

const std::string level =
    "5 5\n"
    "#####\n"
    "#.a.#\n"
    "#.A.#\n"
    "#.@.#\n"
    "#####\n";
```

```
15
16 BOOST_AUTO_TEST_CASE(testLoadLevelAndParseSize) {
17     SB::Sokoban game;
18     std::istringstream in(level);
19     in >> game;
20
21     BOOST_CHECK_EQUAL(game.width(), 5);
22     BOOST_CHECK_EQUAL(game.height(), 5);
23 }
24
25 BOOST_AUTO_TEST_CASE(testPlayerMovesIntoEmptySpace) {
26     SB::Sokoban game;
27     std::istringstream in(level);
28     in >> game;
29
30     auto start = game.playerLoc();
31     game.movePlayer(SB::Direction::Left);
32     auto after = game.playerLoc();
33
34     BOOST_CHECK_EQUAL(after.x, start.x - 1);
35     BOOST_CHECK_EQUAL(after.y, start.y);
36 }
37
38 BOOST_AUTO_TEST_CASE(testPlayerBlockedByWall) {
39     SB::Sokoban game;
40     std::istringstream in(level);
41     in >> game;
42
43     auto start = game.playerLoc();
44     game.movePlayer(SB::Direction::Down);
45     auto after = game.playerLoc();
46
47     BOOST_CHECK_EQUAL(after.x, start.x);
48     BOOST_CHECK_EQUAL(after.y, start.y);
49 }
50
51 BOOST_AUTO_TEST_CASE(testPushBoxIntoStorageWinCondition) {
52     SB::Sokoban game;
53     std::istringstream in(level);
54     in >> game;
55
56     game.movePlayer(SB::Direction::Up);
57     game.movePlayer(SB::Direction::Up);
58
59     BOOST_CHECK(game.isWon());
60 }
61
62 BOOST_AUTO_TEST_CASE(testResetRestoresOriginalState) {
63     SB::Sokoban game;
64     std::istringstream in(level);
65     in >> game;
66
67     auto original = game.playerLoc();
68     game.movePlayer(SB::Direction::Left);
69     game.reset();
70     auto afterReset = game.playerLoc();
71
72     BOOST_CHECK_EQUAL(afterReset.x, original.x);
73     BOOST_CHECK_EQUAL(afterReset.y, original.y);
```

```
74      BOOST_CHECK(!game.isWon());
75  }
76
77  BOOST_AUTO_TEST_CASE(testCannotPushBoxIntoWall) {
78      const std::string level =
79          "5 5\n"
80          "#####\n"
81          "#.A##\n"
82          "#.@.#\n"
83          "#...#\n"
84          "#####\n";
85
86      SB::Sokoban game;
87      std::istringstream in(level);
88      in >> game;
89
90      game.movePlayer(SB::Direction::Up);
91
92      std::ostringstream out;
93      out << game;
94      std::string board = out.str();
95
96      BOOST_CHECK(board.find("A") != std::string::npos);
97      BOOST_CHECK(board.find("@") != std::string::npos);
98  }
99
100 BOOST_AUTO_TEST_CASE(testCannotPushBoxIntoAnotherBox) {
101     const std::string level =
102         "5 5\n"
103         "#####\n"
104         "#.A.#\n"
105         "#.A.#\n"
106         "#.@.#\n"
107         "#####\n";
108
109     SB::Sokoban game;
110     std::istringstream in(level);
111     in >> game;
112
113     game.movePlayer(SB::Direction::Up);
114
115     std::ostringstream out;
116     std::string board;
117     out << game;
118     board = out.str();
119
120     int boxCount = std::count(board.begin(), board.end(), 'A');
121     int playerCount = std::count(board.begin(), board.end(), '@');
122
123     BOOST_CHECK_EQUAL(boxCount, 2);
124     BOOST_CHECK_EQUAL(playerCount, 1);
125 }
126
127 BOOST_AUTO_TEST_CASE(testCannotMoveOrPushOffScreen) {
128     const std::string level =
129         "3 3\n"
130         "###\n"
131         "#@A\n"
132         "###\n";
```

```
133
134     SB::Sokoban game;
135     std::istringstream in(level);
136     in >> game;
137
138     std::ostringstream before;
139     before << game;
140
141     game.movePlayer(SB::Direction::Right);
142
143     std::ostringstream after;
144     after << game;
145
146     BOOST_CHECK_EQUAL(before.str(), after.str());
147 }
148
149 BOOST_AUTO_TEST_CASE(testCannotPushBoxOffScreen) {
150     const std::string level =
151         "3 3\n"
152         "###\n"
153         "#@A\n"
154         "###\n";
155
156     SB::Sokoban game;
157     std::istringstream in(level);
158     in >> game;
159
160     auto before = game.playerLoc();
161     game.movePlayer(SB::Direction::Right);
162     auto after = game.playerLoc();
163
164     BOOST_CHECK_EQUAL(before.x, after.x);
165     BOOST_CHECK_EQUAL(before.y, after.y);
166 }
167
168 BOOST_AUTO_TEST_CASE(testVictoryWithMultipleBoxes) {
169     const std::string level =
170         "5 5\n"
171         "#####\n"
172         "#.a.#\n"
173         "#.A.#\n"
174         "#.A@#\n"
175         "#####\n";
176
177     SB::Sokoban game;
178     std::istringstream in(level);
179     in >> game;
180
181     game.movePlayer(SB::Direction::Left);
182     game.movePlayer(SB::Direction::Up);
183     game.movePlayer(SB::Direction::Up);
184
185     game.movePlayer(SB::Direction::Down);
186     game.movePlayer(SB::Direction::Down);
187     game.movePlayer(SB::Direction::Left);
188     game.movePlayer(SB::Direction::Up);
189
190     BOOST_CHECK(game.isWon());
191 }
```

```
192
193 BOOST_AUTO_TEST_CASE(testVictoryWithExtraStorageTargets) {
194     const std::string level =
195         "5 5\n"
196         "#####\n"
197         "#.a.#\n"
198         "#.1.#\n"
199         "#.@.#\n"
200         "#####\n";
201
202     SB::Sokoban game;
203     std::istringstream in(level);
204     in >> game;
205
206     game.movePlayer(SB::Direction::Up);
207     game.movePlayer(SB::Direction::Up);
208
209     BOOST_CHECK(game.isWon());
210 }
211
212 BOOST_AUTO_TEST_CASE(testSymbolsAreRecognizedCorrectly) {
213     const std::string level =
214         "3 3\n"
215         "#a#\n"
216         "#@#\n"
217         "#A#\n";
218
219     SB::Sokoban game;
220     std::istringstream in(level);
221     in >> game;
222
223     std::ostringstream out;
224     out << game;
225     std::string board = out.str();
226
227     BOOST_CHECK(board.find('a') != std::string::npos);
228     BOOST_CHECK(board.find('@') != std::string::npos);
229     BOOST_CHECK(board.find('A') != std::string::npos);
230
231     BOOST_CHECK(!game.isWon());
232 }
```

6 PS5: DNA Alignment

6.1 Discussion

This project focused on implementing a dynamic programming solution to compute the optimal edit distance between two DNA sequences. The goal was to quantify how similar two sequences are by finding the least amount of operations (insertions, deletions, substitutions) required to transform one into the other. Additionally, to compute the edit distance, the program also outputs the alignment of the sequences showing which characters match, mismatch, or are inserted/deleted. The main logic was implemented in the `EDistance` class, and the functionality was tested through a driver and unit tests.
This project had the option for pair programming, where you work with a partner but I opted out, doing it alone.

6.2 Core Components

The core component for completing this project was the dynamic programming table which was represented as a 2D vector of integers, where each entry (`opt[i][j]`) stored the minimum edit distance between input sequences. To compute the table, the algorithm iteratively filled each cell based on the minimum cost of an insertion, deletion, or substitution operations:

```
1 int match = _opt[i + 1][j + 1] + penalty(_x[i], _y[j]);
2 int gap_x = _opt[i + 1][j] + 2;
3 int gap_y = _opt[i][j + 1] + 2;
4 _opt[i][j] = min3(match, gap_x, gap_y);
```

The `EDistance::penalty()` function assigned a cost of 0 for matches and 1 for mismatches, while insertions and deletions were assigned a gap cost of 2.

After constructing the table, the alignment was recovered by traversing the table from the top-left corner to the bottom-right corner, choosing the move that led to the cells optimal value. This backward traversal produced the DNA sequences with the least amount of edits and outputted the alignment correctly:

```
1  while (i < _m || j < _n) {
2      if (i < _m && j < _n && _opt[i][j] == _opt[i + 1][j + 1] + penalty(_x[i
       ], _y[j])) {
3          out << _x[i] << " " << _y[j] << " " << penalty(_x[i], _y[j]) << "\n"
       ;
4          i++;
5          j++;
6      } else if (i < _m && _opt[i][j] == _opt[i + 1][j] + 2) {
7          out << _x[i] << " " << "-" << " 2\n";
8          i++;
9      } else if (j < _n && _opt[i][j] == _opt[i][j + 1] + 2) {
10         out << "-" << " " << _y[j] << " 2\n";
11         j++;
12     }
13 }
```

6.3 What I accomplished

- Implemented a dynamic programming matrix to compute the minimum edit distance.
- Backtracks through the matrix to generate the alignment string.
- Provides functions to return the edit distance and aligned sequences.
- Wrote a test program to verified the correctness of the sequence

6.4 What I already knew

Prior to starting this project, I was already familiar with dynamic programming as a concept and had written basic DP problems in the past. I also understood the idea of edit distance from algorithms courses and had basic experience with string manipulation and 2D arrays in C++.

6.5 What I learned

I learned how to fully implement dynamic programming in C++ from scratch without relying on helper libraries. This included creating a 2D cost matrix, understanding the recurrence relations for computing minimum edits, and implementing an efficient backtracking algorithm to extract the aligned sequences. I also gained experience designing an object-oriented interface for an algorithmic solution, keeping the logic encapsulated in a class.

6.6 Challenges

Some of the challenges in this project was correctly handling the backtracking to reconstruct the aligned sequences. It was easy to get confused between insertions, deletions, and substitutions when navigating through the cost matrix. Another challenge was ensuring the base cases in the dynamic programming matrix were initialized correctly I overcame these by stepping through small example sequences and printing intermediate matrices.
Despite these challenges, the program is fully functional.

6.7 Evidence

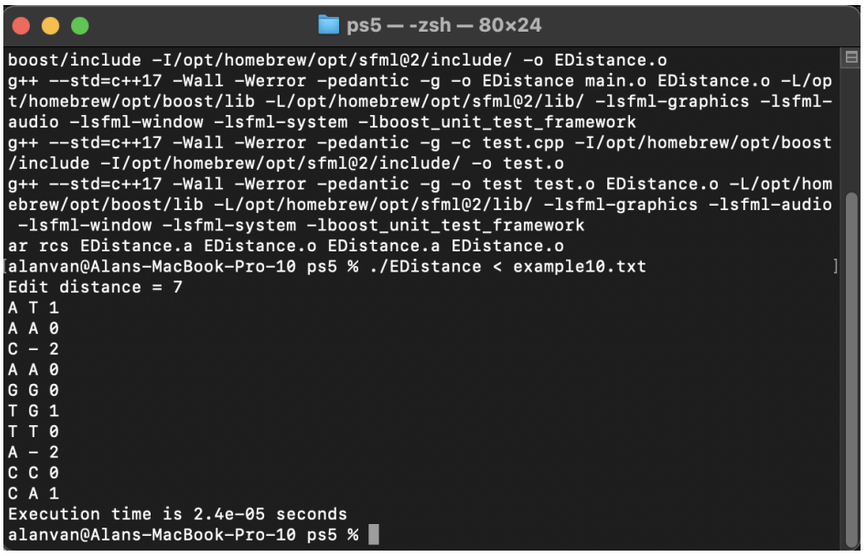
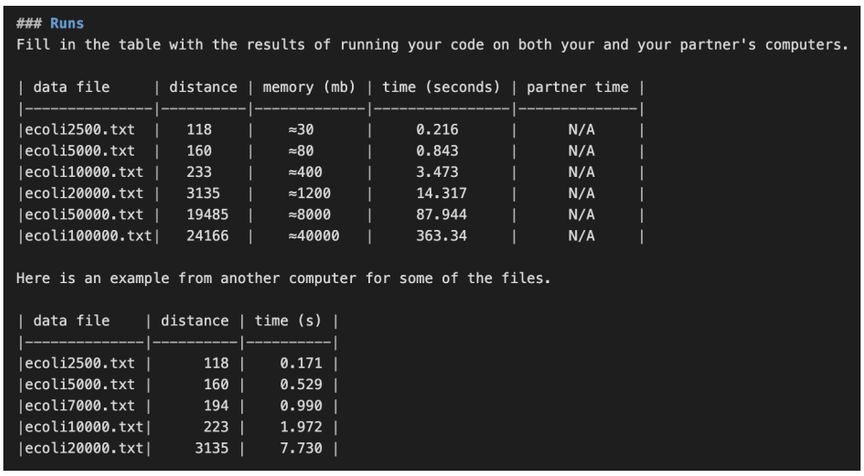


Figure 8: DNA sequence alignment computed with optimal edit distance algorithm

Additional evidence can be seen below, for memory usage based on runs executed on my local machine:

- `Computer:` 2023 MacBook Pro
- `Memory:` 16GB LPDDR5
- `Processor:` Apple M2 Pro



Runs

Fill in the table with the results of running your code on both your and your partner's computers.

data file	distance	memory (mb)	time (seconds)	partner time
ecoli2500.txt	118	≈30	0.216	N/A
ecoli5000.txt	160	≈80	0.843	N/A
ecoli10000.txt	233	≈400	3.473	N/A
ecoli20000.txt	3135	≈1200	14.317	N/A
ecoli50000.txt	19485	≈8000	87.944	N/A
ecoli100000.txt	24166	≈40000	363.34	N/A

Here is an example from another computer for some of the files.

data file	distance	time (s)
ecoli2500.txt	118	0.171
ecoli5000.txt	160	0.529
ecoli7000.txt	194	0.990
ecoli10000.txt	223	1.972
ecoli20000.txt	3135	7.730

Figure 9: Runtime and memory usage for multiple input sizes on my local machine

6.8 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
INCLUDEDIR = -I/opt/homebrew/opt/boost/include -I/opt/homebrew/opt/sfml@2/
    include/
LIBDIR = -L/opt/homebrew/opt/boost/lib -L/opt/homebrew/opt/sfml@2/lib/

# Your .hpp files
DEPS = EDistance.hpp
# Your compiled .o files
OBJECTS = EDistance.o
# The name of your program
PROGRAM = EDistance
TEST_PROGRAM = test

STATIC_LIB = EDistance.a

.PHONY: all clean lint

all: $(PROGRAM) $(TEST_PROGRAM) $(STATIC_LIB)

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

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

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

test.o: test.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $< $(INCLUDEDIR) -o $@

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

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

```

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

main.cpp:

```cpp
// Copyright [2025] Alan Van
#include "EDistance.hpp"
#include <SFML/System/Clock.hpp>
#include <SFML/System/Time.hpp>

int main() {
    std::string x, y;
    std::cin >> x >> y;

    sf::Clock clock;

    EDistance ed(x, y);
    int distance = ed.optDistance();
    std::string alignment = ed.alignment();

    sf::Time t = clock.getElapsedTime();

    std::cout << "Edit distance = " << distance << "\n";
    std::cout << alignment;
    std::cout << "Execution time is " << t.asSeconds() << " seconds" << std
   ::endl;

    return 0;
}
```

EDistance.hpp:

```cpp
// Copyright [2025] Alan Van
#pragma once
#include <iostream>
#include <string>
#include <sstream>
#include <iomanip>

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);

    int optDistance();
    std::string alignment();
 private:
    std::string _x;
    std::string _y;
    int** _opt;
    int _m;
    int _n;
};
```

EDistance.cpp:

```cpp
// Copyright [2025] Alan Van
#include "EDistance.hpp"

EDistance::EDistance(const std::string &x, const std::string &y)
```

```
5       : _x(x), _y(y), _m(x.length()), _n(y.length()) {
6       _opt = new int*[_m + 1];
7       for (int i = 0; i <= _m; ++i) {
8           _opt[i] = new int[_n + 1];
9       }
10  }
11
12  int EDistance::penalty(char a, char b) {
13      if (a == b) return 0;
14      return 1;
15  }
16
17  int EDistance::min3(int a, int b, int c) {
18      return std::min(std::min(a, b), c);
19  }
20
21  int EDistance::optDistance() {
22      for (int i = 0; i <= _m; i++) {
23          _opt[i][_n] = 2 * (_m - i);
24      }
25      for (int j = 0; j <= _n; j++) {
26          _opt[_m][j] = 2 * (_n - j);
27      }
28
29      for (int i = _m - 1; i >= 0; i--) {
30          for (int j = _n - 1; j >= 0; --j) {
31              int match = _opt[i + 1][j + 1] + penalty(_x[i], _y[j]);
32              int gap_x = _opt[i + 1][j] + 2;
33              int gap_y = _opt[i][j + 1] + 2;
34              _opt[i][j] = min3(match, gap_x, gap_y);
35          }
36      }
37
38      return _opt[0][0];
39  }
40
41  std::string EDistance::alignment() {
42      std::ostringstream out;
43      int i = 0, j = 0;
44
45      while (i < _m || j < _n) {
46          if (i < _m && j < _n && _opt[i][j] == _opt[i + 1][j + 1] + penalty(
    _x[i], _y[j])) {
47              out << _x[i] << " " << _y[j] << " " << penalty(_x[i], _y[j]) <<
    "\n";
48              ++i;
49              ++j;
50          } else if (i < _m && _opt[i][j] == _opt[i + 1][j] + 2) {
51              out << _x[i] << " " << "-" << " 2\n";
52              ++i;
53          } else if (j < _n && _opt[i][j] == _opt[i][j + 1] + 2) {
54              out << "-" << " " << _y[j] << " 2\n";
55              ++j;
56          }
57      }
58
59      return out.str();
60  }
```

test.cpp:

```cpp
// Copyright [2025] Alan Van
#include "EDistance.hpp"

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

BOOST_AUTO_TEST_CASE(testPenalty) {
    BOOST_CHECK_EQUAL(EDistance::penalty('A', 'A'), 0);
    BOOST_CHECK_EQUAL(EDistance::penalty('A', 'T'), 1);
}

BOOST_AUTO_TEST_CASE(testMin3) {
    BOOST_CHECK_EQUAL(EDistance::min3(3, 2, 4), 2);
    BOOST_CHECK_EQUAL(EDistance::min3(5, 5, 5), 5);
}

BOOST_AUTO_TEST_CASE(testOptDistance) {
    EDistance ed("AAC", "AGC");
    BOOST_CHECK_EQUAL(ed.optDistance(), 1);
}

BOOST_AUTO_TEST_CASE(testAlignmentExample10) {
    EDistance ed("AACAGTTACC", "TAAGGTCA");
    BOOST_CHECK_EQUAL(ed.optDistance(), 7);
    std::string align = ed.alignment();
    BOOST_CHECK(align.find("A T 1") != std::string::npos);
    BOOST_CHECK(align.find("C - 2") != std::string::npos);
}
```

7 PS6: RandWriter

7.1 Discussion

This project focused on implemented a random text generator based on a Markov model of order k. The model builds a frequency table of k-grams and the characters that follow them in a given source text. The generated output mimics the statistical structure of the input by randomly selecting next characters based on observed frequencies. The main class, `RandWriter`, handles building the frequency table and generating characters and text, while `TextWriter` is the driver program that handles input/output and command-line arguments.

7.2 Core Components

The core component for completing this project was a *Markov model* that stored k-gram frequencies and the following character distributions using a `std::unordered_map`, where each key was a string representing a k-gram and the value was another map storing the count of each following character.

Given an input string S of length n and a model order k, the Markov model defines a probability of characters that follow each k-gram. For each k-gram 'g', the probability of a character 'c' appearing after g is computed as:

$$P(c \mid g) = \frac{\texttt{count}(g \to c)}{\sum_{c'} \texttt{count}(g \to c')}$$

This probability formula was later used to in `RandWriter::generate` to generate random text by sampling observed frequencies.

The model was constructed by sliding a window of size k across the input text and tallying character transitions. To handle wraparound, the text was treated as circular by appending the first k characters to the end:

```
std::string circular = text + text.substr(0, k);

for (size_t i = 0; i < text.length(); i++) {
    std::string kgram = circular.substr(i, k);
    char nextChar = circular[i + k];
    _model[kgram][nextChar]++;
}
```

To generate random text, the program repeatedly chose the next character based on the observed frequencies using `std::discrete_distribution`, which sampled from a weighted probability distribution:

```
std::vector<char> chars;
std::vector<int> weights;

for (const auto& pair : freqMap) {
    chars.push_back(pair.first);
    weights.push_back(pair.second);
}

std::discrete_distribution<> dist(weights.begin(), weights.end());
return chars[dist(_gen)];
```

The generated output was built one character at a time, beginning with the initial seed k-gram and repeatedly extracting the most recent k character from the current output and uses that as a basis for selecting the next. The chosen character is then appended to the output and repeated until the desired length is reach:

```
std::string output = kgram;
while (output.length() < L) {
    std::string lastKgram = output.substr(output.length() - _k, _k);
    output += kRand(lastKgram);
}
```

7.3 What I accomplished

- A constructor that builds a map of k-grams to the frequencies of following characters.
- Functions to return the frequency of a given k-gram and of a k-gram/character pair.
- A method to generate a random character following a given k-gram using weighted randomness.
- A method to generate a pseudo-random text string of length N.
- Wrote a test program to verify the correctness the random character generation and text simulation.

7.4 What I already knew

Prior to starting this project, I already had experience with STL data structures like `std::map` and `std::string`, and I was comfortable working with iterators, file I/O, and string manipulation in C++. I was also familiar with concepts behind probability distributions and random number generation.

7.5 What I learned

I learned how to implement a probabilistic model that mimics real text structure using Markov chains. Specifically, I improved my understanding of how to compute frequencies from raw data, how to simulate weighted randomness using a cumulative distribution, and how to manage edge cases like wraparound in k-gram strings. I improved my skill in writing unit tests to verify specific behaviors, such as correct frequency counts and consistent output with a given random seed for this project.

7.6 Challenges

Some of the biggest challenges for this project was implementing `kgramFreq` and `charFreq` accurately. Off-by-one errors were common when scanning sub-strings or wrapping around the end of the text. Another challenge was generating characters based on weighted probabilities, which was to make sure the sum of frequencies matched and the logic respected the expected distribution.
Despite these challenges, the program is fully functional.

7.7 Evidence

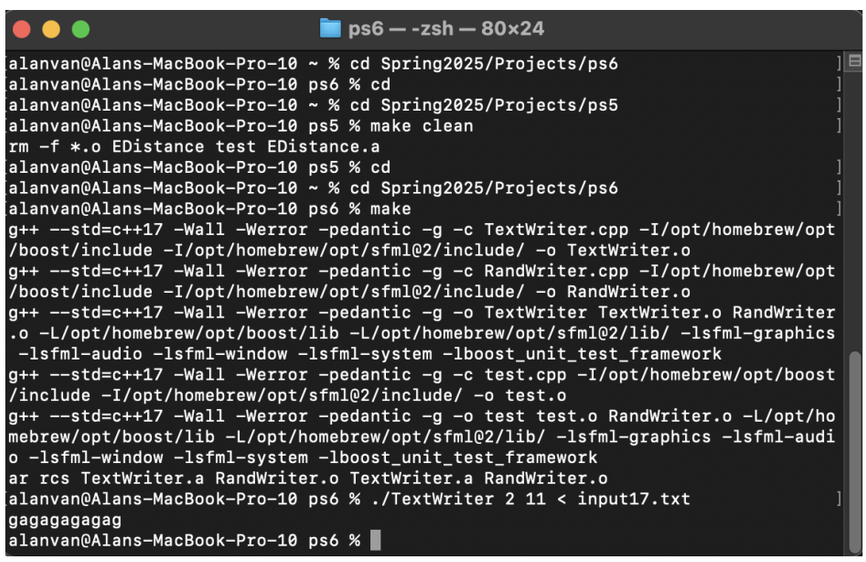


Figure 10: Random text generated using a character-based Markov model of order 2

7.8 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
INCLUDEDIR = -I/opt/homebrew/opt/boost/include -I/opt/homebrew/opt/sfml@2/
    include/
LIBDIR = -L/opt/homebrew/opt/boost/lib -L/opt/homebrew/opt/sfml@2/lib/

# Your .hpp files
DEPS = RandWriter.hpp
# Your compiled .o files
OBJECTS = RandWriter.o
# The name of your program
PROGRAM = TextWriter
TEST_PROGRAM = test

STATIC_LIB = TextWriter.a

.PHONY: all clean lint

all: $(PROGRAM) $(TEST_PROGRAM) $(STATIC_LIB)

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

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

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

test.o: test.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $< $(INCLUDEDIR) -o $@

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

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

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

TextWriter.cpp:

```
// Copyright [2025] Alan Van
#include "RandWriter.hpp"

int main(int argc, char* argv[]) {
    if (argc != 3) {
        std::cerr << "Usage: ./TextWriter <order k> <length L>\n";
        return 1;
    }

    size_t k = std::stoi(argv[1]);
    size_t L = std::stoi(argv[2]);

    std::stringstream buffer;
    buffer << std::cin.rdbuf();
```

```
    std::string input = buffer.str();

    RandWriter rw(input, k);
    std::string seed = input.substr(0, k);

    std::string generated = rw.generate(seed, L);
    std::cout << generated << std::endl;

    return 0;
}
```

RandWriter.hpp:

```
// Copyright [2025] Alan Van
#include <string>
#include <random>
#include <stdexcept>
#include <sstream>
#include <algorithm>
#include <set>
#include <map>
#include <unordered_map>
#include <vector>
#include <iostream>

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);

    friend std::ostream& operator<<(std::ostream& out, const RandWriter& w);

 private:
    // Private member variables go here
    size_t _k;
    std::string _text;
    std::string _alphabet;
    std::unordered_map<std::string, std::map<char, int>> _model;
    std::mt19937 _gen;
```

```
};
```

RandWriter.cpp:

```
// Copyright [2025] Alan Van
#include "RandWriter.hpp"

RandWriter::RandWriter(const std::string& text, size_t k) : _k(k), _text(text), _gen(1337) {
    if (text.length() < k) throw std::invalid_argument("Text must be at least length k");

    std::string circular = text + text.substr(0, k);

    for (size_t i = 0; i < text.length(); i++) {
        std::string kgram = circular.substr(i, k);
        char nextChar = circular[i + k];
        _model[kgram][nextChar]++;
    }

    std::set<char> alphabet;
    for (char c : text) alphabet.insert(c);
    _alphabet.assign(alphabet.begin(), alphabet.end());
}

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 {
    if (kgram.length() != _k) throw std::invalid_argument("kgram must be of length k");
    auto it = _model.find(kgram);
    if (it == _model.end()) return 0;
    int total = 0;
    for (const auto& pair : it->second) total += pair.second;
    return total;
}

// 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 (_k != 0 && kgram.length() != _k) throw std::invalid_argument("kgram must be of length k");
    if (_k == 0) return std::count(_text.begin(), _text.end(), c);
    auto it = _model.find(kgram);
    if (it == _model.end()) return 0;
    auto inner = it->second.find(c);
    return (inner != it->second.end()) ? inner->second : 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.length() != _k) throw std::invalid_argument("kgram must be of length k");
    if (_model.find(kgram) == _model.end()) throw std::invalid_argument("kgram not found");
```

```
51
52      const auto& freqMap = _model[kgram];
53      std::vector<char> chars;
54      std::vector<int> weights;
55
56      for (const auto& pair : freqMap) {
57          chars.push_back(pair.first);
58          weights.push_back(pair.second);
59      }
60
61      std::discrete_distribution<> dist(weights.begin(), weights.end());
62      return chars[dist(_gen)];
63  }
64
65  // Generate a string of length L characters by simulating a trajectory
66  // through the corresponding Markov chain. The first k characters of
67  // the newly generated string should be the argument kgram.
68  // Throw an exception if kgram is not of length k.
69  // Assume that L is at least k
70  std::string RandWriter::generate(const std::string& kgram, size_t L) {
71      if (kgram.length() != _k) throw std::invalid_argument("kgram must be of
        length k");
72      if (L < _k) throw std::invalid_argument("length L must be at least k");
73
74      std::string output = kgram;
75      while (output.length() < L) {
76          std::string lastKgram = output.substr(output.length() - _k, _k);
77          output += kRand(lastKgram);
78      }
79      return output;
80  }
81
82  // Overload the stream insertion operator << and display the internal state
83  // of the Markov model. Print out the order, alphabet, and the frequencies
84  // of the k-grams and k+1-grams
85  std::ostream& operator<<(std::ostream& out, const RandWriter& w) {
86      out << "Order: " << w._k << "\n";
87      out << "Alphabet: " << w._alphabet << "\n";
88
89      for (const auto& pair : w._model) {
90          out << "Kgram: " << pair.first << "\n";
91          for (const auto& c : pair.second) {
92              out << "    '" << c.first << "' : " << c.second << "\n";
93          }
94          out << "\n";
95      }
96      return out;
97  }
```

test.cpp:

```
1   // Copyright [2025] Alan Van
2   #include "RandWriter.hpp"
3
4   #define BOOST_TEST_DYN_LINK
5   #define BOOST_TEST_MODULE RandWriter_Tests
6   #include <boost/test/included/unit_test.hpp>
7
8   BOOST_AUTO_TEST_CASE(testOrderK) {
9       RandWriter rw("gagggagaggcgagaaa", 2);
10      BOOST_REQUIRE_EQUAL(rw.orderK(), 2);
```

```
}

BOOST_AUTO_TEST_CASE(testFreq) {
    RandWriter rw("gagggagaggcgagaaa", 2);

    BOOST_REQUIRE_NO_THROW(rw.freq("ga"));
    BOOST_REQUIRE_EQUAL(rw.freq("ga"), 5);

    BOOST_REQUIRE_NO_THROW(rw.freq("gc"));
    BOOST_REQUIRE_EQUAL(rw.freq("gc"), 1);

    BOOST_REQUIRE_NO_THROW(rw.freq("zz"));
    BOOST_REQUIRE_EQUAL(rw.freq("zz"), 0);

    BOOST_REQUIRE_THROW(rw.freq("g"), std::invalid_argument);
}

BOOST_AUTO_TEST_CASE(testFreqCharacters) {
    RandWriter rw("gagggagaggcgagaaa", 2);

    BOOST_REQUIRE_NO_THROW(rw.freq("ga", 'g'));
    BOOST_REQUIRE_EQUAL(rw.freq("ga", 'g'), 4);

    BOOST_REQUIRE_NO_THROW(rw.freq("ga", 'a'));
    BOOST_REQUIRE_EQUAL(rw.freq("ga", 'a'), 1);

    BOOST_REQUIRE_NO_THROW(rw.freq("ga", 'x'));
    BOOST_REQUIRE_EQUAL(rw.freq("ga", 'x'), 0);

    BOOST_REQUIRE_THROW(rw.freq("g", 'a'), std::invalid_argument);
}

BOOST_AUTO_TEST_CASE(testKRand) {
    RandWriter rw("gagggagaggcgagaaa", 2);

    BOOST_REQUIRE_NO_THROW(rw.kRand("ga"));

    char c = rw.kRand("ga");

    BOOST_REQUIRE(c == 'g' || c == 'a');

    BOOST_REQUIRE_THROW(rw.kRand("g"), std::invalid_argument);
    BOOST_REQUIRE_THROW(rw.kRand("zz"), std::invalid_argument);
}

BOOST_AUTO_TEST_CASE(testGenerate) {
    RandWriter rw("gagggagaggcgagaaa", 2);

    std::string output = rw.generate("ga", 10);

    BOOST_REQUIRE_EQUAL(output.length(), 10);
    BOOST_REQUIRE(output.substr(0, 2) == "ga");

    BOOST_REQUIRE_THROW(rw.generate("g", 10), std::invalid_argument);

    BOOST_REQUIRE_THROW(rw.generate("zz", 10), std::invalid_argument);
}
```

8 PS7: Kronos Log Parsing

8.1 Discussion

This project involved writing a C++ program that parses through boot logs from six Kronos devices and reports the start and end times of each boot event, as well as the duration. The goal was to determine which boots completed successfully and which did not. To do this, I used regular expressions to identify boot start markers and success indicators, and Boost's `DateTime` library to parse through and compute time differences between timestamps.

8.2 Core Components

The core component for completing this project was parsing through boot logs from the Kronos InTouch system to extract timestamps and calculate boot durations. Regular expressions (`regex`) were used to identify specific lines that marked the start and completion messages from the log files. These were matched using patterns like:

```
1 std::regex bootStartRegex(R"(log\.c\.166\) server started)");
2 std::regex bootCompleteRegex(R"(oejs\.AbstractConnector:Started
    SelectChannelConnector@.*)");
```

Boost's `date_time` library was also used to parse through and manipulate timestamps extracted from matched lines. This allowed accurate computation of system uptime by subtracting boot start times from completion times. The program was structured with modular helper functions for timestamp extraction, regex matching, and formatting. To handle edge cases, such as incomplete boots or malformed logs, it gracefully skipped or flagged problematic entries with clear output labels.

Data was stored using `std::pair` and `std::vector` containers, allowing flexible accumulation of log entries and report formatting. The final report was written to a `.rpt` file, summarizing each boot event with line numbers, timestamps, and calculated durations.

8.3 What I accomplished

I implemented a program that successfully reads through the log files, extracts relevant timestamps, determines the completion status of boots, and reports both the duration and outcome. My code outputs a readable report for each device and handles edge cases such as logs with missing completion markers. I structured the solution across multiple files: one for time utilities, one for regex parsing, and a main driver that coordinates file reading and output.

8.4 What I already knew

Prior to starting this project, I was already familiar with regular expressions in general and had experience parsing through structured text files. I also had some familiarity with working across multiple files in C++ and using headers for modularity. I had used 'std::chrono' before, but not the Boost DateTime library.

8.5 What I learned

I learned how to use Boost's `ptime` and `time_duration` objects for time calculations. I also became more comfortable with C++ regex capturing groups and file stream handling through this project. Additionally, I gained insight into how log-based systems track boot sequences and how to write flexible parsers that can gracefully handle incomplete or corrupted logs.

8.6 Challenges

Some of the biggest challenges in this project was regex, getting the regular expressions to match the correct lines in the logs was difficult as it was new. At first, I was checking for the wrong completion pattern (`Boot Completed`), when the correct pattern was actually something like `oejs.AbstractConnector:Started SelectChannelConnector`. This caused all logs to be incorrectly marked as incomplete. Once I fixed the pattern, everything

else started falling into place. Another minor challenge was handling incorrect timestamps gracefully without crashing the parser.
Despite these challenges, the program is fully functional.

8.7 Evidence

```
Device Boot Report

InTouch log file: ./device1_intouch.log
Lines Scanned: 443838

Device boot count: initiated = 6, completed: 6


=== Device boot ===
435369(./device1_intouch.log): 2014-03-25 19:11:59 Boot Start
435759(./device1_intouch.log): 2014-03-25 19:15:02 Boot Completed
    Boot Time: 183.000s

=== Device boot ===
436500(./device1_intouch.log): 2014-03-25 19:29:59 Boot Start
436859(./device1_intouch.log): 2014-03-25 19:32:44 Boot Completed
    Boot Time: 165.000s

=== Device boot ===
440719(./device1_intouch.log): 2014-03-25 22:01:46 Boot Start
440791(./device1_intouch.log): 2014-03-25 22:04:27 Boot Completed
    Boot Time: 161.000s

=== Device boot ===
440866(./device1_intouch.log): 2014-03-26 12:47:42 Boot Start
441216(./device1_intouch.log): 2014-03-26 12:50:29 Boot Completed
    Boot Time: 167.000s

=== Device boot ===
442094(./device1_intouch.log): 2014-03-26 20:41:34 Boot Start
442432(./device1_intouch.log): 2014-03-26 20:44:13 Boot Completed
    Boot Time: 159.000s

=== Device boot ===
443073(./device1_intouch.log): 2014-03-27 14:09:01 Boot Start
443411(./device1_intouch.log): 2014-03-27 14:11:42 Boot Completed
    Boot Time: 161.000s
```

Figure 11: Boot log output from device1_intouch.log.rpt

8.8 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lboost_date_time
INCLUDEDIR = -I/opt/homebrew/opt/boost/include -I/opt/homebrew/opt/sfml@2/
   include/
LIBDIR = -L/opt/homebrew/opt/boost/lib -L/opt/homebrew/opt/sfml@2/lib/

SRC = ps7.cpp
# Your compiled .o files
```

```
OBJECTS = ps7.o
# The name of your program
PROGRAM = ps7

.PHONY: all clean lint

all: $(PROGRAM)

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

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

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

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

ps7.hpp:

```
// Copyright [2025] Alan Van
#pragma once

#include <iostream>
#include <regex>
#include <string>
#include <fstream>
#include <vector>
#include <boost/date_time/posix_time/posix_time.hpp>

struct BootEvent {
    int lineNumber;
    std::string timestampStr;
    boost::posix_time::ptime timestamp;
};

boost::posix_time::ptime parseTimestamp(const std::string& line);

std::string extractTimestampStr(const std::string& line);
```

ps7.cpp:

```
// Copyright [2025] Alan Van
#include "ps7.hpp"

boost::posix_time::ptime parseTimestamp(const std::string& line) {
    std::regex tsRegex(R"((\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}(?:\.\d+)?))")
    ;
    std::smatch match;
    if (std::regex_search(line, match, tsRegex)) {
        std::string timestampStr = match.str(1);

        std::istringstream iss(timestampStr);
        std::locale loc(std::locale::classic(), new
            boost::posix_time::time_input_facet("%Y-%m-%d %H:%M:%S"));
        iss.imbue(loc);
        boost::posix_time::ptime pt;
        iss >> pt;

```

```
17          if (pt.is_not_a_date_time() && timestampStr.find('.') != std::string
    ::npos) {
18              std::string truncated = timestampStr.substr(0, timestampStr.find
    ('.'));
19              std::istringstream iss2(truncated);
20              iss2.imbue(loc);
21              iss2 >> pt;
22          }
23          return pt;
24      }
25      return boost::posix_time::ptime(boost::posix_time::not_a_date_time);
26  }
27
28  std::string extractTimestampStr(const std::string& line) {
29      std::regex tsRegex(R"((\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2})(?:\.\d+)?)")
    ;
30      std::smatch match;
31      if (std::regex_search(line, match, tsRegex)) {
32          return match.str(1);
33      }
34      return "Unknown";
35  }
36
37  int main(int argc, char* argv[]) {
38      if (argc != 2) {
39          std::cerr << "Usage: ./ps7 <logfile>" << std::endl;
40          return 1;
41      }
42
43      std::string inputFilename = argv[1];
44      std::ifstream infile(inputFilename);
45      if (!infile) {
46          std::cerr << "Error opening input file." << std::endl;
47          return 1;
48      }
49
50      std::string outputFilename = inputFilename + ".rpt";
51      std::ofstream outfile(outputFilename);
52      std::stringstream bootLog;
53      if (!outfile) {
54          std::cerr << "Error creating output file." << std::endl;
55          return 1;
56      }
57
58      std::regex bootStartRegex(R"(log\.c\.166\) server started)");
59      std::regex bootCompleteRegex(R"(oejs\.AbstractConnector:Started
    SelectChannelConnector@.*)");
60
61      int lineNumber = 0;
62      std::string line;
63      BootEvent currentBoot;
64      bool inBoot = false;
65      int bootStarted = 0;
66      int bootCompleted = 0;
67
68      while (std::getline(infile, line)) {
69          lineNumber++;
70
71          if (std::regex_search(line, bootStartRegex)) {
```

```
72              if (inBoot) {
73                  bootLog << "\n=== Device boot ===\n";
74                  bootLog << currentBoot.lineNumber << "(" << inputFilename <<
    "): "
75                          << currentBoot.timestampStr << " Boot Start\n";
76                  bootLog << "**** Incomplete boot ****\n";
77              }
78
79              currentBoot.lineNumber = lineNumber;
80              currentBoot.timestampStr = extractTimestampStr(line);
81              currentBoot.timestamp = parseTimestamp(line);
82              inBoot = true;
83              bootStarted++;
84              continue;
85          }
86
87          if (inBoot && std::regex_search(line, bootCompleteRegex)) {
88              boost::posix_time::ptime completeTime = parseTimestamp(line);
89              if (completeTime.is_not_a_date_time() || currentBoot.timestamp.
    is_not_a_date_time()) {
90                  bootLog << "\n=== Device boot ===\n";
91                  bootLog << currentBoot.lineNumber << "(" << inputFilename <<
    "): "
92                          << currentBoot.timestampStr << " Boot Start\n";
93                  bootLog << "**** Incomplete boot ****\n";
94                  inBoot = false;
95                  continue;
96              }
97
98              bootCompleted++;
99              boost::posix_time::time_duration elapsed = completeTime -
    currentBoot.timestamp;
100
101             bootLog << "\n=== Device boot ===\n";
102             bootLog << currentBoot.lineNumber << "(" << inputFilename << "):
    "
103                     << currentBoot.timestampStr << " Boot Start\n";
104             bootLog << lineNumber << "(" << inputFilename << "): "
105                     << extractTimestampStr(line) << " Boot Completed\n";
106             bootLog << "\tBoot Time: " << std::fixed << std::setprecision(3)
107                     << elapsed.total_milliseconds() / 1000.0 << "s\n";
108
109             inBoot = false;
110         }
111     }
112
113     if (inBoot) {
114         bootLog << "\n=== Device boot ===\n";
115         bootLog << currentBoot.lineNumber << "(" << inputFilename << "): "
116                 << currentBoot.timestampStr << " Boot Start\n";
117         bootLog << "**** Incomplete boot ****\n";
118     }
119
120     outfile << "Device Boot Report\n\n";
121     outfile << "InTouch log file: " << inputFilename << "\n";
122     outfile << "Lines Scanned: " << lineNumber << "\n\n";
123     outfile << "Device boot count: initiated = " << bootStarted
124             << ", completed: " << bootCompleted << "\n\n";
125     outfile << bootLog.str();
```

```
126
127     infile.close();
128     outfile.close();
129
130     return 0;
131 }
```

datetime.cpp:

```
1  // date and time sample code
2  // Copyright (C) 2015 Fred Martin
3  // Tue Apr 21 17:37:46 2015
4
5  // compile with
6  // g++ datetime.cpp -lboost_date_time
7  // Y. Rykalova  4/12/2021
8
9  // http://www.boost.org/doc/libs/1_58_0/doc/html/date_time/gregorian.html
10 // http://www.boost.org/doc/libs/1_58_0/doc/html/date_time/posix_time.html
11
12 #include <iostream>
13 #include <string>
14 #include <boost/date_time/gregorian/gregorian.hpp>
15 #include <boost/date_time/posix_time/posix_time.hpp>
16
17 using std::cout;
18 using std::cin;
19 using std::endl;
20 using std::string;
21
22 using boost::gregorian::date;
23 using boost::gregorian::from_simple_string;
24 using boost::gregorian::date_period;
25 using boost::gregorian::date_duration;
26
27 using boost::posix_time::ptime;
28 using boost::posix_time::time_duration;
29
30 int main() {
31   // Gregorian date stuff
32   string s("2015-01-01");
33   date d1(from_simple_string(s));
34   date d2(2015, boost::gregorian::Apr, 21);
35
36   date_period dp(d1, d2);  // d2 minus d1
37
38   date_duration dd = dp.length();
39
40   cout << "duration in days " << dd.days() << endl;
41
42   // Posix date stuff
43   ptime t1(d1, time_duration(0, 0, 0, 0));  // hours, min, secs, nanosecs
44   ptime t2(d2, time_duration(0, 0, 0, 0));
45
46   time_duration td = t2 - t1;
47
48   cout << "duration in hours " << td.hours() << endl;
49   cout << "duration in ms " << td.total_milliseconds() << endl;
50
51 }
```

stdin_regex.cpp:

```
// compile with
// g++ stdin_regex.cpp

// regex_match example
#include <iostream>
#include <string>
#include <regex>

using namespace std;

int main ()
{


  string s, rs;
  regex e;

  // see https://en.cppreference.com/w/cpp/regex/error_type
  cout << "Here are some helpful error codes you may encounter\n";
  cout << "while constructing your regex\n";
  cout << "error_collate " << regex_constants::error_collate << endl;
  cout << "error_ctype " << regex_constants::error_ctype << endl;
  cout << "error_escape " << regex_constants::error_escape << endl;
  cout << "error_backref " << regex_constants::error_backref << endl;
  cout << "error_brack " << regex_constants::error_brack << endl;
  cout << "error_paren " << regex_constants::error_paren << endl;
  cout << "error_brace " << regex_constants::error_brace << endl;
  cout << "error_badbrace " << regex_constants::error_badbrace << endl;

  cout << endl;

  cout << "Enter regex > ";
  getline (cin, rs);

  try {
    e = regex (rs);
  } catch (regex_error& exc) {
    cout << "Regex constructor failed with code " << exc.code() << endl;
    exit(1);
  }

  cout << "Enter line > ";

  while (getline(cin, s)) {

    cout << endl;

    if (regex_match (s,e))
      cout << "string object \"" << s << "\" matched\n\n";

    if ( regex_match ( s.begin(), s.end(), e ) )
      cout << "range on \"" << s << "\" matched\n\n";

    smatch sm;    // same as match_results<string::const_iterator> sm;
    regex_match (s,sm,e);
    cout << "string object \"" << s << "\" with " << sm.size() << " matches\
   n\n";
    // uses constant iterators so requires -std=gnu++0x
```

```
58      //    regex_match ( s.cbegin(), s.cend(), sm, e);
59      //    cout << "range on \"" << s << "\" with " << sm.size() << " matches
        \n";
60
61      if (sm.size() > 0) {
62        cout << "the matches were: ";
63        for (unsigned i=0; i<sm.size(); ++i) {
64      cout << "[" << sm[i] << "] " << endl;
65        }
66      }
67
68      cout << endl << endl;
69
70      cout << "Enter line > ";
71    }
72
73
74    return 0;
75  }
```