Computing IV Sec 201: Project Portfolio

James Walsh

Spring 2023

Contents

Time to Complete Portfolio: 10 hours

0.1 Discussion

What I accomplished

In this project, I created a Sprite Object that is a class from the Simple and Fast Multimedia Library (SFML). The Sprite has a number of features. One feature is allowing the Sprite to move across the display window in response to keystrokes. Also, it can increase or decrease in size. Lastly, the sprite has been given an image to represent the object on the display window.

Design and Features

For my project, I decided to use the arrow keys to move the sprite on the screen. To do this I used a `switch` statement that responded to `event.type.code` which is a keystroke. If any arrow (up, down, right, left) was pressed, the Sprite moved 10 units in the corresponding direction. For this assignment, I had to come up with a feature to implement on my own. The feature I decided to add was increasing and decreasing the size of the Sprite. I approached the implementation the same way as before, with a switch statement responding to keystrokes. If the 'I' key is pressed the image will increase by a scale of 2 units by using the Sprite's `setScale()` function. Similarly, if the 'D' key is pressed the image will decrease by a scale of .2 units by using the same function as before. The reason I used switch statements to implement the features of the Sprite was for simplicity. Instead of having multiple `if` statements to check the key pressed, having one `switch` statement with multiple cases makes the codes' functionality and appearance much simpler.

What I already knew and what I learned

Going into this project I had a good understanding of what classes are and how to use them. However, I have never used or heard of the SFML library, so getting comfortable with its classes was a learning curve.

When completing this assignment I learned how to use the simple objects/classes in the SFML library. Specifically Shapes, Colors, Sprites, Textures, Events, and RenderWindows. Each of these objects have a number of member functions that come along with them. I was able to get a good understanding of how each work by reading the documentation found at `sfml-dev.org`.

1 PS1: Linear Feedback Shift Register and Image Encoding

1.1 Discussion

What I accomplished

For this project, I created a Linear Feedback Shift Register (LFSR) class. My implementation of the LFSR, given a given bit register, shifts each bit one space to the left and replaces the vacated bit (the far right bit) with a bit computed by the Boolean exclusive-or function (XOR). The function computes the replacement bit by XOR'ing 3 tap bits and the bit cut-off at the far left of the register after the shift. In this project, the tap bits used were 10, 12, and 13.

In this project, I also created an algorithm that encrypts and decrypts images by using a linear feedback shift register. The Algorithm is described below in the **D**esign and Features section.

Design and Features

When implementing the LFSR I used a `std::vector` of characters to represent the register of bits. I found implementing the register as a vector was a good representation because adding to vector was simple by using `push back()`. The `std::vector` class also has random access memory, which makes implementation easy. Within the class, I created two member functions, `step()` and `generate()`. The `step()` function is a void function that takes no parameters, and simulates one LFSR shift, and updates the bit register accordingly. The `generate()` function simulates takes an integer `k` as a parameter, and simulates `k` steps of the register and returns a `k-bit` integer that the bit-register represents.

I used the LFSR to encrypt and decrypt images by extracting each pixel in the image's red, green, and blue components. For each pixel, the integer value that represents the red/blue/green components are XOR'ed with a newly-generated 8-bit integer that is generated by the LFSR's `generate()` function. Lastly, the resulting integer from each component (r/b/g) is used to create a new color which is saved to the current pixel. This same process can be used for encrypting and decrypting an image. However, to decrypt an image the original bit-register must be identical to the original bit-register used to encrypt the image. This algorithm is implemented in the `transform()` function which can be found in the `PhotoMagic.cpp` file.

What I already knew and what I learned

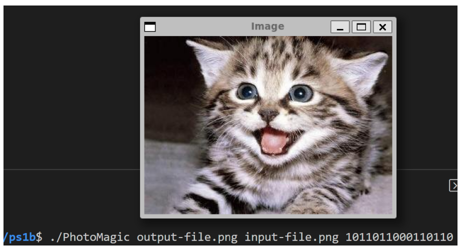
Before completing this assignment I knew what I knew what I bit-register was and I was also familiar with the Boolean exclusive-or function. However, I have never seen a programming implementation of an LFSR.

When writing the program I learned how to use what I already knew and apply it to the LFSR object. Once I understood all aspects of the assignment, the implementation was not difficult. When using my newly created LFSR object to encypt and decrypt an image there was definitely a learning curve. I was not familiar with image encryption, but I quickly realized how I could use the `generate()` function to XOR the result with each rbg integer to create a new color to use in the encrypted image.

Unit Testing

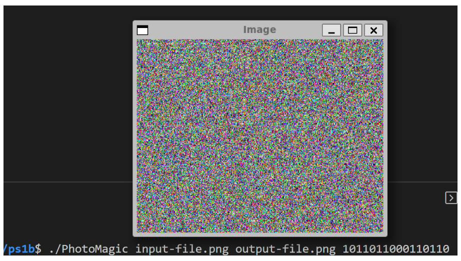
For this program, I wrote six different unit tests using the Boost Unit Test Framework. All test created a FibLFSR with "1011011000110110" as the given register. I created two tests using `BOOST_REQUIRE_EQUAL()` testing if the `step()` and `generate()` function returns the propper integers. Then I used `BOOST_REQUIRE_NO_THROW()` to test is any for loops go out of bounds. Lastly, I created a test `BOOST_CHECK_GE` and `BOOST_CHECK_LE` to test if `generate(8)` returns a integer between 0 and 255 inclusively. The reason I created this test is to ensure the return value would be a valid rbg color. My program passed all tests.

Below is a screenshot of the Original Image. See Figure 1.
Along with a screenshot of the Encoded Image. See Figure 2.



```
/ps1b$ ./PhotoMagic output-file.png input-file.png 1011011000110110
```

Figure 1: Original/Decoded Image.



```
/ps1b$ ./PhotoMagic input-file.png output-file.png 1011011000110110
```

Figure 2: Encoded Image.

1.2 Codebase

```
CC = g++
CFLAGS = -Wall -Werror -pedantic -std=c++17 -g
LIBS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
    lboost_unit_test_framework
DEPS = FibLFSR.hpp
OBJS = FibLFSR.o

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

.PHONY: all clean

all: PhotoMagic test

PhotoMagic: PhotoMagic.o $(OBJS)
  $(CC) $(CFLAGS) -o $@ $^ $(LIBS)

test: $(OBJS) test.o
  $(CC) $(FLAGS) -o test $^ $(LIBS)

clean:
  rm *.o PhotoMagic test
```

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

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

int main(int argc, char* argv[])
{
  char* inputFile = argv[1];
  char* outputFile = argv[2];
  char* LFSRseed = argv[3];

  sf::Image image;
  if (!image.loadFromFile(inputFile))
    return -1;
  FibLFSR myLFSR(LFSRseed);

  transform(image, &myLFSR);

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

  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(outputFile))
    return -1;

  return 0;
}

// Transforms image using FibLFSR
void transform(sf::Image& aImage, FibLFSR* aFibLFSR) {
  sf::Vector2u size = aImage.getSize();
  sf::Color p;
  unsigned int x;
  unsigned int y;
  int newInt;

  for (x = 0; x < size.x; x++) {
    for (y = 0; y < size.y; y++) {
      p = aImage.getPixel(x, y);
      newInt = aFibLFSR->generate(8);
      p.r = p.r ^ newInt;
      newInt = aFibLFSR->generate(8);
      p.g = p.g ^ newInt;
      newInt = aFibLFSR->generate(8);
      p.b = p.b ^ newInt;
      aImage.setPixel(x, y, p);
    }
    }

}
// Display an encrypted copy of the picture, using LFSR to do the encryption
```

```
#pragma once

#include <string>
#include <vector>

class FibLFSR{
public:
    // Constructor to create LFSR with the given initial seed
    FibLFSR(std::string seed);

    // Simulate one step and return the new bit 0 or 1
    int step();
    // Simulate k steps and return a k-bit integer
    int generate(int k);

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

private:
    // Representation of the seed as an integer
    std::vector<char> seedVector;
    // Taps
    int t1, t2, t3;
};
std::ostream& operator<<(std::ostream& out, const FibLFSR& lfsr);
```

```
// Copyright 2023 James Walsh
#include "FibLFSR.hpp"
#include <iostream>
#include <vector>
#include <string>
#include <cmath>

int const LENGTH_OF_SEED = 16;

FibLFSR::FibLFSR(std::string seed) {
    // convert the seed to a vector of chars
    int i;

    for (i = 0; i < LENGTH_OF_SEED; i++) {
        seedVector.push_back(seed[i]);
    }

    // init taps indexies
    t1 = 2;
    t2 = 3;
    t3 = 5;
}

int FibLFSR::step() {
    int newBit;
    int i;

    // XOR the last bit and tap bits
    newBit = seedVector[0] ^ seedVector[t3];
    newBit = newBit ^ seedVector[t2];
    newBit = newBit ^ seedVector[t1];

    for (i = 0; i < LENGTH_OF_SEED - 1; i++) {
        seedVector[i] = seedVector[i + 1];
    }
    if (newBit == 1) {
        seedVector[LENGTH_OF_SEED - 1] = ’1’;
    } else {
        seedVector[LENGTH_OF_SEED - 1] = ’0’;
    }

    return newBit;
}

int FibLFSR::generate(int k) {
    int currentStep = 0;
    int kBit = 0;
    int i;

    while (currentStep < k) {
        this->step();
        currentStep++;
    }
    for (i = LENGTH_OF_SEED - 1; i > (LENGTH_OF_SEED - k - 1); i--) {
        if (seedVector[i] == ’1’) {
            kBit += pow(2, ((LENGTH_OF_SEED - 1) - i));
        }
    }
    return kBit;
}

std::ostream& operator<<(std::ostream& out, const FibLFSR& lfsr) {
    int i;

    for (i = 0; i < LENGTH_OF_SEED; i++) {
        out << lfsr.seedVector[i];
    }

    return out;
}
```

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

#include "FibLFSR.hpp"

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

BOOST_AUTO_TEST_CASE(testStepInstr1) {
  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);
```

```cpp
}

BOOST_AUTO_TEST_CASE(testStepInstr2) {
  FibLFSR l2("1011011000110110");
  BOOST_REQUIRE_EQUAL(l2.generate(9), 51);
}

BOOST_AUTO_TEST_CASE(testConstructorOutOfBounds) {
  FibLFSR l3("1011011000110110");
  BOOST_REQUIRE_NO_THROW(l3.step());
}

BOOST_AUTO_TEST_CASE(testStepOutOfBounds) {
  FibLFSR l4("1011011000110110");
  BOOST_REQUIRE_NO_THROW(l4.step());
}

BOOST_AUTO_TEST_CASE(testGenerateOutOfBounds) {
  FibLFSR l5("1011011000110110");
  BOOST_REQUIRE_NO_THROW(l5.step());
}

BOOST_AUTO_TEST_CASE(testTransformAlgorithm) {
   FibLFSR l6("1011011000110110");
  int eightBitNum = l6.generate(8);
  BOOST_CHECK_GE(eightBitNum, 0);
  BOOST_CHECK_LE(eightBitNum, 255);
}
```

2 PS2: Sokoban

2.1 Discussion

What I accomplished

This program creates a fully functional Sokoban Game. Sokoban is a tile-based video game where the player controls a warehouse worker. The goal of the game is to push boxes into designated storage locations. The program uses a level file, which is a text file consisting of a width and height as well as a series of symbols ('.' - empty space, '#' - wall, 'a' - empty storage space, 'A' - box, '@' -player) that represent a level for the Sokoban game. The program then creates a Sokoban object and sets up the game. It establishes a width and height for the display window and stores the series of symbols in a `std::vector` of characters which represents a Matrix. Each matrix element (x, y) corresponds to a coordinate on the display window where an image block will be drawn. After the Sokoban object is initialized, the program draws the level to the display window, it does this by iterating through the matrix. The image corresponding to the current element is drawn to the screen for the entire matrix. After this is complete, the Sokoban level is properly displayed using a SFML render window.

Once the UI is set up, the user can use the WASD keys to move the player up, left, down, and up to play the game. The player is allowed to move in any direction as long as the player is not moving into a wall, the end of the window, or a moveable box. A box is unmovable if the space where the player is trying to move it has another box, if there is a wall, or if the box is already in a storage location. Once the Player has moved all the boxes into storage, or if all the storages contain boxes, you win! Also, the user can use the 'R' key to reset the game at any time.

Design and Features

To program the game, I created a `Sokoban` class that is inherited from SFML's `Drawable` class. The class consists of two constructors, a default constructor that prompts the user to enter a level and then uses the extraction to initialize the object, a value constructor that takes a character array that represents a level file, and a `movePlayer()` function. The class also has accessor and mutator functions for each of its private member variables, besides a mutator function for the `GRID_SCALE` variable because it is const. The class has a private draw function which is a virtual function inherited from the sf::Drawable class. The draw function iterates through the matrix (vector of characters) and draws the corresponding image to the display window. Along with these member functions, the class contains a list of private member variables. A virtual draw function which is overridden from the `Drawable` class, height and width variables that store the height/width of the game, an isStorage boolean that is used to keep track if the user has entered a storage location, a const `GRID_SCALE` that is set to 64 because each tile image is 64 pixels, a SFML `Texture player` that is used for the current player image being displayed, two counter variables `moveable_boxes` and `empty_boxes`, and lastly the `gameGrid`. I chose to implement the game grid as an `std::vector` of characters because using I found using std iterators is useful if, for some reason, I changed the type of the game grid to a `list` the iterators would carry over somewhat easily. The `Sokoban` class also overloads the insertion and extraction operators. The extraction operator initializes the Sokoban object using the level file, and the insertion operator outputs the level to the terminal in the format given in the level file.

The movePlayer function takes an Enumerated type `Movement` (W, A, S, D), and the function checks if the move is possible. It does this by calling a series of helper functions that check if the movement is in range, if there is a box in the way, and if so can the box be moved. If the movement is possible, the movePlayer function updates the `gameGrid` vector. Once the player has moved all the boxes into storage, or if all the storages contain boxes, you win, and a message is displayed. It does this by calling a `isWon` function which checks if the player has won. If so, it returns true. Main uses this to display the message and pause the game until it is reset.

What I already knew and what I learned

Going into this project, I was not familiar with making any sort of video game, and I have not had much experience with visual display windows either. However, I was used to working with vectors and how to manipulate the elements such that they represent a 2D-Array, so that is why I decided to implement the game grid as a vector of characters.

When completing this project, I learned a lot about how implementing a low-level video game works. Going into this project, I did not realize how intricate a single move can be. I learned how to being aware of every how every decision you make in your code is crutial. This project improved my advanced programming design skills drastically. As the project went on, I got better and better at implementing conditionals and often found myself going back to previous functions and reworking some of the conditionals because I had found a better/simpler algorithm. Lastly, I learned a lot about how delegating work to other functions can make code a lot cleaner and more understandable.

Below is a screenshot of a Completed Level: Figure 3.

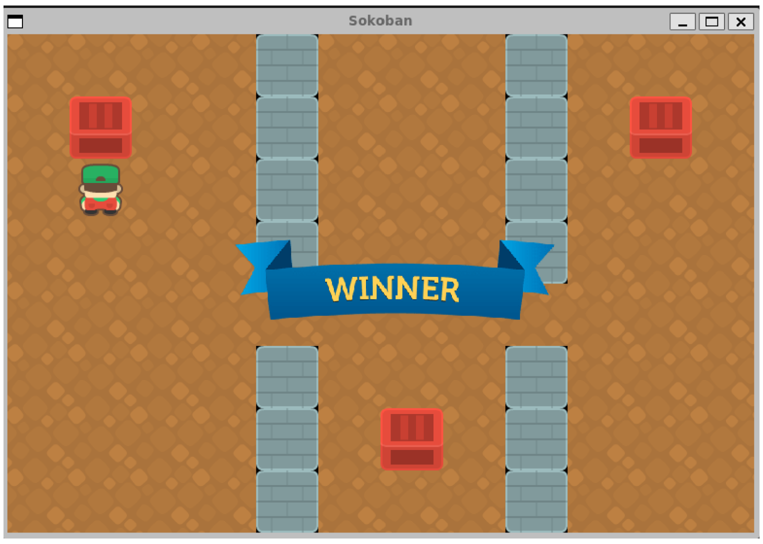


Figure 3: Game Display.

2.2 Codebase

```
CC = g++
CFLAGS = -Wall -Werror -pedantic -std=c++17 -g
LIBS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
DEPS = Sokoban.hpp
OBJS = Sokoban.o

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

.PHONY: all clean lint

all: Sokoban

Sokoban: main.o $(OBJS)
  $(CC) $(CFLAGS) -o $@ $^ $(LIBS)

lint:
  cpplint *.cpp *.hpp

clean:
  rm *.o Sokoban
```

```
// Copyright 2023 James Walsh
#include <iostream>
```

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

int main(int argc, char* argv[]) {
    char* level = argv[1];
    Sokoban newGame(level);

    sf::RenderWindow window(sf::VideoMode(
            newGame.get_height() * newGame.get_scale(),
            newGame.get_width() * newGame.get_scale()), "Sokoban");

    sf::Texture winnerImage;
    winnerImage.loadFromFile("winner.png");
    sf::Sprite won(winnerImage);
    won.scale(.7, .7);
    won.setPosition((newGame.get_height() * newGame.get_scale()) / 3.5,
         (newGame.get_width() * newGame.get_scale())/ 7);

    bool end = false;
    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                window.close();
            }
            if (event.type == sf::Event::KeyPressed) {
                if (!end) {
                    switch (event.key.code) {
                        case sf::Keyboard::W:   // move player up
                            newGame.movePlayer(UP); break;
                        case sf::Keyboard::A:   // move player left
                            newGame.movePlayer(LEFT); break;
                        case sf::Keyboard::S:   // move player down
                            newGame.movePlayer(DOWN); break;
                        case sf::Keyboard::D:   // move player right
                            newGame.movePlayer(RIGHT); break;
                        default: break;
                    }
                }
                if (event.key.code == sf::Keyboard::R) {
                        newGame.reset_level(level);
                        end = false;
                        break;
                }
            }
            window.clear();
            window.draw(newGame);
            if (newGame.isWon()) {
                end = true;
                window.draw(won);
                window.display();
            }
            window.display();
        }
    }

    return 0;
}
```

```cpp
// Copyright 2023 James Walsh
#pragma once
#include <iostream>
#include <vector>
#include <map>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

enum movement { UP, LEFT, DOWN, RIGHT };
typedef enum movement Movement;

class Sokoban : public sf::Drawable {
 public:
  Sokoban();
  explicit Sokoban(char* iLevelFile);

  void movePlayer(const Movement m);
  bool isWon() const;
  bool check_and_set_box(size_t x, size_t y, Movement m);
  void reset_level(char* iLevelFile);
  bool within_bounds(size_t x, size_t y) const;

  void set_width(size_t w) { width = w; }
  void set_height(size_t h) { height = h; }
  void set_gameGrid(size_t size) { gameGrid = std::vector<char>(size, ' '); }
  size_t get_width() const { return width; }
  size_t get_height() const { return height; }
  std::vector<char> get_gameGrid() const { return gameGrid; }
  int get_scale() const { return GRID_SCALE; }
  void set_player_pos(size_t x, size_t y, Movement m);
  void set_movable_boxes(size_t increment) { movable_boxes += increment; }
  void set_empty_locations(size_t increment) { empty_locations += increment; }

 private:
  void draw(sf::RenderTarget& target, sf::RenderStates states) const override;
  size_t width;
  size_t height;
  std::vector<char> gameGrid;
  bool isStorage = false;
  const int GRID_SCALE = 64;
  sf::Texture player;
  size_t movable_boxes = 0;
  size_t empty_locations = 0;
};

std::istream& operator>>(std::istream& inStream, Sokoban& object);
std::ostream& operator<<(std::ostream& outStream, const Sokoban& object);
```

```cpp
// Copyright 2023 James Walsh
#include "Sokoban.hpp"
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
```

```

Sokoban::Sokoban() {
    std::cout << "Enter a level to play: ";
    std::cin >> *this;
    std::cout << *this << std::endl;
    player.loadFromFile("player_05.png");
}

Sokoban::Sokoban(char* iLevelFile) {
    std::ifstream levelFile;
    levelFile.open(iLevelFile);

    levelFile >> width;
    levelFile.get();
    levelFile >> height;
    gameGrid = std::vector<char>((width * height), ’ ’);

    char content;
    size_t x;
    size_t y;
    for (x = 0; x < width; x++) {
        for (y = 0; y < height; y++) {
            levelFile >> content;
            gameGrid[x + y * width] = content;
        }
    }

    empty_locations = std::count(gameGrid.begin(), gameGrid.end(), ’a’);
    movable_boxes = std::count(gameGrid.begin(), gameGrid.end(), ’A’);

    // DISPLAY THE LEVEL TO THE TERMINAL
    std::cout << *this << std::endl;
    std::cout << movable_boxes << " " << empty_locations << std::endl;
    player.loadFromFile("player_05.png");
}

void Sokoban::reset_level(char* iLevelFile) {
    empty_locations = 0;
    movable_boxes = 0;
    std::ifstream levelFile;
    levelFile.open(iLevelFile);

    levelFile >> width;
    levelFile.get();
    levelFile >> height;
    gameGrid = std::vector<char>((width * height), ’ ’);

    char content;
    size_t x;
    size_t y;
    for (x = 0; x < width; x++) {
        for (y = 0; y < height; y++) {
            levelFile >> content;
            gameGrid[x + y * width] = content;
        }
    }

    empty_locations = std::count(gameGrid.begin(), gameGrid.end(), ’a’);
    movable_boxes = std::count(gameGrid.begin(), gameGrid.end(), ’A’);

    // DISPLAY THE LEVEL TO THE TERMINAL
    std::cout << *this << std::endl;
    player.loadFromFile("player_05.png");
}

void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const {
    size_t x;
    size_t y;
    sf::Texture wall, empty, box, storage;
    char current_block;

    wall.loadFromFile("block_06.png");
    empty.loadFromFile("ground_01.png");
    box.loadFromFile("crate_03.png");
    storage.loadFromFile("ground_04.png");
    for (x = 0; x < width; x++) {
        for (y = 0; y < height; y++) {
            current_block = gameGrid[x + y * width];
            if (current_block == ’#’) {
                sf::Sprite block(wall);
                block.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                target.draw(block);
            }
            if (current_block == ’.’) {
                sf::Sprite block(empty);
                block.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                target.draw(block);
            }
            if (current_block == ’a’) {
                sf::Sprite back(empty);
                back.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                target.draw(back);
                sf::Sprite block(storage);
                block.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                target.draw(block);
            }
            if (current_block == ’A’) {
                sf::Sprite back(empty);
                back.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                target.draw(back);
                sf::Sprite block(box);
                block.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                target.draw(block);
            }
            if (current_block == ’1’) {
                sf::Sprite back(empty);
                back.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                target.draw(back);
                sf::Sprite block(box);
                block.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                target.draw(block);
            }
            if (current_block == ’@’) {
                if (isStorage) {
                    sf::Sprite back(storage);
                    back.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    target.draw(back);
```

```cpp
                } else {
                    sf::Sprite back(empty);
                    back.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    target.draw(back);
                }
                sf::Sprite block(player);
                block.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                target.draw(block);
            }
        }
    }
}

bool Sokoban::isWon() const {
    if (empty_locations == 0 || movable_boxes == 0) {
        return true;
    }
    return false;
}

void Sokoban::movePlayer(const Movement m) {
    size_t x = 0;
    size_t y = 0;

    auto get_player_pos = [this, &x, &y] () -> void {
        char currentBlock;

        for (x = 0; x < width; x++) {
            for (y = 0; y < height; y++) {
                currentBlock = gameGrid[x + y * width];
                if (currentBlock == '@') {
                    return;
                }
            }
        }
    };
    get_player_pos();

    switch (m) {
        case UP:
            if (within_bounds(x - 1, y) && (gameGrid[(x - 1) + y * width]
                 != '#' && gameGrid[(x - 1) + y * width] != '1')) {
                set_player_pos(x, y, UP);
                player.loadFromFile("player_08.png");
            }
            break;
        case LEFT:
            if (within_bounds(x, y - 1) && (gameGrid[x + (y - 1) * width]
                 != '#' && gameGrid[x + (y - 1) * width] != '1')) {
                set_player_pos(x, y, LEFT);
                player.loadFromFile("player_20.png");
            }
            break;
        case DOWN:
            if (within_bounds(x + 1, y) && (gameGrid[(x + 1)+ y * width]
                 != '#' && gameGrid[(x + 1) + y * width] != '1')) {
                set_player_pos(x, y, DOWN);
                player.loadFromFile("player_05.png");
            }
            break;
        case RIGHT:
            if (within_bounds(x, y + 1) && (gameGrid[x + (y + 1) * width]
                 != '#' && gameGrid[x + (y + 1) * width] != '1')) {
                set_player_pos(x, y, RIGHT);
                player.loadFromFile("player_17.png");
            }
            break;
        default: break;
    }
}

void Sokoban::set_player_pos(size_t x, size_t y, Movement m) {
    switch (m) {
        case UP:
            if (!check_and_set_box(x, y, UP)) break;
            if (isStorage) {
                gameGrid[(x - 1) + y * width] = '@';
                gameGrid[x + y * width] = 'a';
                isStorage = false;
            } else {
                if (gameGrid[(x - 1) + y * width] == 'a') {
                    isStorage = true;
                }
                gameGrid[(x - 1) + y * width] = '@';
                gameGrid[x + y * width] = '.';
            }
            break;
        case LEFT:
            if (!check_and_set_box(x, y, LEFT)) break;
            if (isStorage) {
                gameGrid[x + (y - 1) * width] = '@';
                gameGrid[x + y * width] = 'a';
                isStorage = false;
            } else {
                if (gameGrid[x + (y - 1) * width] == 'a') {
                    isStorage = true;
                }
                gameGrid[x + (y - 1) * width] = '@';
                gameGrid[x + y * width] = '.';
            }
            break;
        case DOWN:
            if (!check_and_set_box(x, y, DOWN)) break;
            if (isStorage) {
                gameGrid[(x + 1) + y * width] = '@';
                gameGrid[x + y * width] = 'a';
                isStorage = false;
            } else {
                if (gameGrid[(x + 1) + y * width] == 'a') {
                    isStorage = true;
                }
                gameGrid[(x + 1) + y * width] = '@';
                gameGrid[x + y * width] = '.';
            }
            break;
        case RIGHT:
            if (!check_and_set_box(x, y, RIGHT)) break;
            if (isStorage) {
                gameGrid[x + (y + 1) * width] = ’@’;
                gameGrid[x + y * width] = ’a’;
                isStorage = false;
            } else {
                if (gameGrid[x + (y + 1) * width] == ’a’) {
                    isStorage = true;
                }
                gameGrid[x + (y + 1) * width] = ’@’;
                gameGrid[x + y * width] = ’.’;
            }
            break;
        default: break;
    }
}

bool Sokoban::check_and_set_box(size_t x, size_t y, Movement m) {
    switch (m) {
    case UP:
        if (gameGrid[(x - 1) + y * width] ==’A’) {
            if (within_bounds(x - 2, y) && gameGrid[(x - 2) + y * width] !=
   ’.’) {
                if (gameGrid[(x - 2) + y * width] == ’a’) {
                    gameGrid[(x - 2) + y * width] = ’1’;
                    gameGrid[(x - 1) + y * width] = ’@’;
                    gameGrid[x + y * width] = ’.’;
                    empty_locations--;
                    movable_boxes--;
                    return true;
                    break;
                }
            }
            if (within_bounds(x - 2, y) && gameGrid[(x - 2) + y * width]
                 != ’#’ && gameGrid[(x - 2) + y * width] != ’A’) {
                gameGrid[(x - 2) + y * width] = ’A’;
                return true;
            } else {
                return false;
            }
        } else {
            return true;
        }
        break;
    case LEFT:
        if (gameGrid[x + (y - 1) * width] ==’A’) {
            if (within_bounds(x, y - 2) && gameGrid[x + (y - 2) * width] !=
   ’.’) {
                if (gameGrid[x + (y - 2) * width] == ’a’) {
                    gameGrid[x + (y - 2) * width] = ’1’;
                    gameGrid[x + (y - 1) * width] = ’@’;
                    gameGrid[x + y * width] = ’.’;
                    empty_locations--;
                    movable_boxes--;
                    return true;
                    break;
                }
            }
            if (within_bounds(x, y - 2) && gameGrid[x + (y - 2) * width]
                 != ’#’ && gameGrid[x + (y - 2) * width] != ’A’) {
                gameGrid[x + (y - 2) * width] = ’A’;
                return true;
            } else {
                return false;
            }
        } else {
            return true;
        }
        break;
    case DOWN:
        if (gameGrid[(x + 1) + y * width] ==’A’) {
            if (within_bounds(x + 2, y) && gameGrid[(x + 2) + y * width] !=
   ’.’) {
                if (gameGrid[(x + 2) + y * width] == ’a’) {
                    gameGrid[(x + 2) + y * width] = ’1’;
                    gameGrid[(x + 1) + y * width] = ’@’;
                    gameGrid[x + y * width] = ’.’;
                    empty_locations--;
                    movable_boxes--;
                    return true;
                    break;
                }
            }
            if (within_bounds(x + 2, y) && gameGrid[(x + 2) + y * width]
                 != ’#’ && gameGrid[(x + 2) + y * width] != ’A’) {
                gameGrid[(x + 2) + y * width] = ’A’;
                return true;
            } else {
                return false;
            }
        } else {
            return true;
        }
        break;
    case RIGHT:
        if (gameGrid[x + (y + 1) * width] ==’A’) {
            if (within_bounds(x, y + 2) && gameGrid[x + (y + 2) * width] !=
   ’.’) {
                if (gameGrid[x + (y + 2) * width] == ’a’) {
                    gameGrid[x + (y + 2) * width] = ’1’;
                    gameGrid[x + (y + 1) * width] = ’@’;
                    gameGrid[x + y * width] = ’.’;
                    empty_locations--;
                    movable_boxes--;
                    return true;
                    break;
                }
            }
            if (within_bounds(x, y + 2) && gameGrid[x + (y + 2) * width]
                 != ’#’ && gameGrid[x + (y + 2) * width] != ’A’) {
                gameGrid[x + (y + 2) * width] = ’A’;
                return true;
            } else {
                return false;
            }
        } else {
            return true;
        }
        break;
        default: break;
    }

    return false;
}

bool Sokoban::within_bounds(size_t x, size_t y) const {
    if (x >= 0 && x < width) {
        if (y >= 0 && y < height)
            return true;
    }
    return false;
}

std::istream& operator>>(std::istream& inStream, Sokoban& object) {
    char* file = new char[10];
    char c;
    int i;
    size_t w;
    size_t h;

    for (i = 0; i < 10; i++) {
        inStream >> c;
        file[i] = c;
    }

    std::ifstream levelFile;
    levelFile.open(file);

    levelFile >> w;
    levelFile.get();
    levelFile >> h;

    object.set_width(w);
    object.set_height(h);
    object.set_gameGrid(w * h);

    char content;
    size_t x;
    size_t y;
    for (x = 0; x < object.get_width(); x++) {
        for (y = 0; y < object.get_height(); y++) {
            levelFile >> content;
            object.get_gameGrid()[x + y * object.get_width()] = content;
            if (content == ’a’) object.set_empty_locations(1);
            if (content == ’A’) object.set_movable_boxes(1);
        }
    }

    return inStream;
}

std::ostream& operator<<(std::ostream& outStream, const Sokoban& object) {
    size_t x;
    size_t y;

    for (x = 0; x < object.get_width(); x++) {
        for (y = 0; y < object.get_height(); y++) {
            outStream << object.get_gameGrid()[x + y * object.get_width()];
        }
        outStream << std::endl;
    }
    return outStream;
}
```

3 PS3: Pythagorean Tree

3.1 Discussion

What I accomplished

This project creates a PTree class with the goal of drawing a Pythagorean Tree to an SFML display window. I implemented a structure called `Branch` to use in my PTree class. My `Branch` structure is similar to a node that would be used in a tree. It contains a self, left, and right which represents the base square and the two squares on top respectively. The PTree contains a function `pTree()` that initializes the first square (base square), and within the pTree function a helper function `extend` is called. This function recursively calls itself until the number of recursions specified by the user is met. Each time the function is called, a new `Branch` is drawn onto the tree, creating the visualization of Pythagoras Tree. However, my program does not display the tree correctly. The implementation of the recursive function does not properly position the next square relative to the previous one.

3.2 Design and Features

The PTree class contains a constructor that initializes the member variables `L, N, recursion_count, root` and also sets up the base of the tree by initializing a `Branch` structure. The class contains a `get_length()` function which returns the value of `L`, and it also contains a `set_length()` function that allows `L` to be updated as more levels of the tree are added. I implemented `pTree` function as a friend function so it has the ability to access the member variables directly. The `extend` function is a member function of PTree. I implemented it this way to make it easy to mutate and access the member functions and variables directly.

I decided to create the `Branch` structure because I thought one variable containing a root, right, and left would be useful when attempting to extend the tree. Having used a similar structure before gave me familiarity that was helpful when implementing.

What I already knew and what I learned

Going into this assignment, I used SFML render windows and draw functions before, so I knew how to display simple shapes such as a square. I also knew how to create a tree-like structure, which is how I chose to implement `Branch`.

When working on this assignment, I learned how to use math skills that I already have to find different distances from a relative point. In order to find the top right and left of the current square, I had to use the current square's origin as a starting point, then find use an equation to find the top corners position.

My program does not produce a proper output.

3.3 Codebase

```
CC = g++
CFLAGS = -g -Wall -Werror -pedantic -std=c++17 -g
LIBS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
DEPS = PTree.hpp
OBJS = PTree.o

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

.PHONY: all clean lint

all: PTree

PTree: main.o $(OBJS)
  $(CC) $(CFLAGS) -o $@ $^ $(LIBS)

```

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

clean:
  rm *.o PTree
```

```
// Copyright 2023 James Walsh
#include <iostream>
#include "PTree.hpp"
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

int main(int argc, char* argv[]) {
    PTree myPTree(40, 3);

    sf::RenderWindow window(sf::VideoMode(6 * myPTree.get_length(),
     4 * myPTree.get_length()), "Pythagoras Tree");

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

    return 0;
}
```

```
// Copyright 2023 James Walsh
#pragma once
#include <iostream>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

struct branch;
typedef struct branch Branch;

struct branch {
  sf::RectangleShape self;
  sf::Vector2f topLeft;
  sf::Vector2f topRight;
  Branch* left;
  Branch* right;
};

class PTree: public sf::Drawable {
 public:
    PTree(double length, int iterations): L(length),
     N(iterations), recursion_count(0), root(new Branch) {
      root->self.setSize(sf::Vector2f(L, L));
      root->topLeft = root->self.getOrigin();
      float x = (root->self.getOrigin()).x + L * 6;
      float y = (root->self.getOrigin()).y * 4;
      root->topRight = sf::Vector2f(x, y);
      root->self.setOrigin((L / 2), (L / 2));
      root->self.setPosition(((6 * L) / 2), (4 * L) - (L / 2));
    }
    ~PTree() {
      delete root;
    }
    double get_length() const { return L; }
    void set_length(double length) { L = length; }
    friend void pTree(PTree& _PTree, sf::RenderWindow& window);
    void extend(Branch* next, sf::RenderWindow& window);

 private:
    void draw(sf::RenderTarget& target, sf::RenderStates states) const override;
    double L;
    int N;
    int recursion_count;
    Branch* root;
};

void pTree(PTree& _PTree, sf::RenderWindow& window);
```

```
// Copyright 2023 James Walsh
#include "PTree.hpp"

void pTree(PTree& _PTree, sf::RenderWindow& window) {
    if (_PTree.recursion_count == 0) {
        window.draw(_PTree.root->self);
        window.display();
        _PTree.set_length(_PTree.L / 2);
    }

    _PTree.extend(_PTree.root, window);
}

void PTree::extend(Branch* next, sf::RenderWindow& window) {
    next->left = new Branch;
    next->right = new Branch;

    set_length(L / 2);
    next->left->self.setSize(sf::Vector2f(L, L));
    next->left->self.setFillColor(sf::Color::Red);
    next->left->self.setOrigin(L / 2, L / 2);
    next->left->self.setPosition(next->topLeft);
    window.draw(next->left->self);
    window.display();

    next->right->self.setSize(sf::Vector2f(L, L));
    next->right->self.setFillColor(sf::Color::Red);
    next->right->self.setOrigin(L / 2, L / 2);
    next->right->self.setPosition(next->topRight);
    window.draw(next->right->self);
    window.display();
    recursion_count++;

    if (recursion_count != N) {
        extend(next->left, window);
        extend(next->right, window);
    }
}
```

4 PS4: Checkers

4.1 Discussion

What I accomplished

For this project, I created a complete checkers game. The board is an 8 x 8 square made up of red and black tiles, starting with a red tile in the top left. Red and Black game pieces are placed on the top 3 rows and the bottom 3 of black tiles, with black game pieces at the top and red at the bottom. The game is displayed using an SFML render window. The game starts by allowing the player to select a piece with the mouse. Black gets to go first, so selecting a piece will turn the tile below blue to indicate the current selection. If you click the same tile, it will remove the selection and allow the same player to select a new piece. Otherwise, selecting any other piece will cause the program to check if it is a valid move. It is a valid move if the tile the player is trying to move to is black, is empty, and one space diagonal. Or the player can jump an oposing piece onto an empty diagonal black tile. Jumping a piece will remove the piece that is jumped. If the move is valid, the piece will be moved to the selected tile, and now the red player can now select a piece. The game will continue until one player has no pieces left. Once one player has won, a red winner icon appears, or a black winner icon appears.

Design and Features

To implement the game, I created a Checkers class. In my Checkers class, there is a default constructor which initializes the member variables and sets up a blank board. To store the game information, I used a `std::vector` of character arrays that are length 8 (the size of each row). I decided to do this because if I changed the vector to be a `std::list`, the iterators in my functions are easy to transfer to the corresponding iterators. After the game is board calls a member function `start_board()`, this function places the game pieces in their initial positions. The class has 3 functions that are used for selecting pieces, `piece_select()`, `move_piece()`, and `can_move()`. The piece_select function is called the main function in main.cpp, and highlights the selected piece and returns true. Then main calls move_piece, which waits for the user to select a destination position, and after that, calls can_move to see if the move is allowed. The can_move function checks to see if the move is allowed (a valid move is described in the previous section). If the move is allowed, the function returns true. However, there is an exception to this, if the player is attempting to "jump" a tile then, the remove_piece function is called. The remove_piece function examines the game grid to see if the tile the player is trying to jump contains an opposing player's game piece. If so, the piece is removed from the board and the remove_piece function returns true. Next, the can_move function updates the game board properly. If the move is not allowed, if can_move returns false, the piece is not moved and unhighlights the current tile, and the player is allowed to select a new destination tile. The class also contains two functions `red_won()` and `black_won()`. These functions return true if the opposing player has no pieces remaining and false otherwise. Lastly, the class contains accessor functions for the dimensions and scale of the board along with an accessor function for the `gameGrid` vector. Also, the insertion operator is overloaded for the class which outputs the components of the gameGrid to the given ostream.

When implementing, I created a lambda expression `is_even()` which checks if a size_t number is even or odd. It returns true if even and false if odd. Also, I used the <algorithm> function `count_if()` in the red_won and black_won functions. I used the algorithm to iterate through the gameGrid vector. At each character array I passed in a lambda function to return true if an oposing teams piece was found.

What I already knew and what I learned

Going into this assignment I was aware of how low-level game movement works due to my experience programming the Sokoban game. Programming the Sokoban game also taught me how important delegation to other functions is very important, so I incorporated that into this program as well.

When completing this assignment, I learned a lot about using std classes to my advantage. Such as `std::pair<>` was very useful for referring to the current position and the destination position. I have not used pairs to the extent I did in this program, so I learned a lot using them. and they will be something I incorporate into my code for future projects.

Below is a screenshot of a New Game: Figure 4.

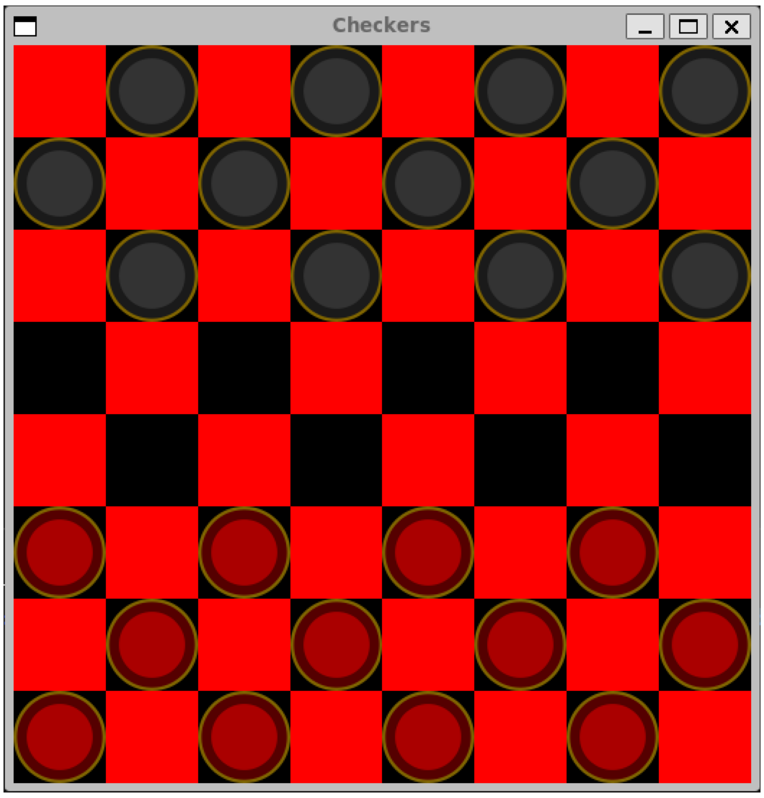


Figure 4: Start of Game.

4.2 Codebase

```
CC = g++
CFLAGS = -g -Wall -Werror -pedantic -std=c++17
LIBS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
DEPS = Checkers.hpp
OBJS = Checkers.o

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

.PHONY: all clean lint

all: Checkers

Checkers: main.o $(OBJS)
  $(CC) $(CFLAGS) -o $@ $^ $(LIBS)

lint:
  cpplint *.cpp *.hpp

clean:
  rm *.o Checkers
```

```
// Copyright 2023 James Walsh
#include <iostream>
#include "Checkers.hpp"
#include <SFML/System.hpp>
```

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

int main(int argc, char* argv[]) {
    Checkers newGame;
    sf::RenderWindow window(sf::VideoMode(
            newGame.get_dimension() * newGame.get_scale(),
            newGame.get_dimension() * newGame.get_scale()), "Checkers");
    sf::Texture winnerImage;
    sf::Sprite redWon, blackWon;
    redWon.scale(.7, .7);
    blackWon.scale(.3, .3);

    char p;

    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                window.close();
            }
            if (!newGame.red_Won() && !newGame.black_Won()) {
                if (event.type == sf::Event::MouseButtonPressed) {
                    size_t x = event.mouseButton.x / 64;
                    size_t y = event.mouseButton.y / 64;
                    std::pair<size_t, size_t> pos(y, x);
                    if (newGame.piece_select(pos, p, window)) {
                        window.draw(newGame);
                        window.display();
                        newGame.move_piece(pos, p, window);
                    }
                }
                window.clear();
                window.draw(newGame);
                window.display();
            } else {
                if (newGame.red_Won()) {
                    winnerImage.loadFromFile("redwin.png");
                    redWon.setTexture(winnerImage);
                    redWon.setPosition(2 * 64, 2 * 46);
                    window.clear();
                    window.draw(newGame);
                    window.draw(redWon);
                    window.display();

                } else if (newGame.black_Won()) {
                    winnerImage.loadFromFile("blackwin.png");
                    blackWon.setTexture(winnerImage);
                    blackWon.setPosition(2 * 64, 2 * 46);
                    window.clear();
                    window.draw(newGame);
                    window.draw(blackWon);
                    window.display();
                }
            }
        }
    }

    return 0;
}
```

```
// Copyright 2023 James Walsh
#pragma once
#include <iostream>
#include <vector>
#include <utility>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

class Checkers: public sf::Drawable {
 public:
    Checkers();

    void start_board();
    bool piece_select(std::pair<size_t, size_t> position, char& p, const sf::RenderWindow& w);
    void move_piece(std::pair<size_t, size_t> position, char piece, sf::RenderWindow& w);
    bool can_move(char piece, std::pair<size_t, size_t> position,
                    std::pair<size_t, size_t> destPosition);
    bool remove_piece(char piece, std::pair<size_t, size_t> position,
                    std::pair<size_t, size_t> destPosition);
    bool red_Won();
    bool black_Won();
    size_t get_dimension() const { return BOARD_DIMENSION; }
    size_t get_scale() const { return GRID_SCALE; }
    char* get_gameGrid_at(std::vector<char[]>::iterator i) const { return *i; }
    friend std::ostream& operator<<(std::ostream& outStream, const Checkers& object);

 private:
    void draw(sf::RenderTarget& target, sf::RenderStates states) const override;
    const size_t BOARD_DIMENSION;
    std::vector<char[8]> gameGrid;
    const int GRID_SCALE;
    sf::Texture redPiece, blackPiece, redKing, blackKing;
    int turn;
};

std::ostream& operator<<(std::ostream& outStream, const Checkers& object);
```

```
// Copyright 2023 James Walsh
#include <algorithm>
#include "Checkers.hpp"

Checkers::Checkers():BOARD_DIMENSION(8), gameGrid(std::vector<char[8]>(BOARD_DIMENSION)),
        GRID_SCALE(64), turn(0) {
    redPiece.loadFromFile("redpawn.png");
    blackPiece.loadFromFile("blackpawn.png");
    redKing.loadFromFile("redking.png");
    blackKing.loadFromFile("blackking.png");
    std::vector<char[8]>::iterator pos;
    size_t i;
    size_t start_color = 0;

    auto is_even = [] (size_t x) -> bool {
        return (x % 2 == 0);
    };

    // start_color is even = R odd = B
    for (pos = gameGrid.begin(); pos != gameGrid.end(); pos++, start_color++) {
        if (is_even(start_color)) {
            for (i = 0; i < BOARD_DIMENSION; i++) {
                if (is_even(i)) {
                    (*pos)[i] = 'T';
                } else {
                    (*pos)[i] = 't';
                }
            }
        } else {
            for (i = 0; i < BOARD_DIMENSION; i++) {
                if (is_even(i)) {
                    (*pos)[i] = 't';
                } else {
                    (*pos)[i] = 'T';
                }
            }
        }
    }
    start_board();

    std::cout << "Game Board:\n" << *this << std::endl;
}

void Checkers::start_board() {
    std::vector<char[8]>::iterator pos;
    size_t row;
    size_t i;

    for (pos = gameGrid.begin(), row= 0; pos != gameGrid.end(); pos++, row++) {
        for (i = 0; i < BOARD_DIMENSION; i++) {
            if (row < 3) {
                if ((*pos)[i] == 't') {
                    (*pos)[i] = 'b';
                }
            } else if (row > 4) {
                if ((*pos)[i] == 't') {
                    (*pos)[i] = 'r';
                }
            }
        }
    }
}

bool Checkers::piece_select(std::pair<size_t, size_t> position, char &p,
                                const sf::RenderWindow& w) {
    char piece;
    piece = gameGrid.at(position.first)[position.second];
    // turn = even: Black turn; turn = odd: Red turn
    if (turn % 2 == 0) {
        if (piece == 'b') {
            gameGrid.at(position.first)[position.second] = 'p';
            p = ’p’;
            return true;
        } else if (piece == ’B’) {
            gameGrid.at(position.first)[position.second] = ’m’;
            p = ’m’;
            return true;
        }
    } else if (turn % 2 != 0) {
        if (piece == ’r’) {
            gameGrid.at(position.first)[position.second] = ’P’;
            p = ’P’;
            return true;
        } else if (piece == ’R’) {
            gameGrid.at(position.first)[position.second] = ’M’;
            p = ’M’;
            return true;
        }
    }
    return false;
}

void Checkers::move_piece(std::pair<size_t, size_t> position, char piece, sf::RenderWindow& w) {
    bool pick = false;

    while (!pick) {
        sf::Event moveTo;
        while (w.pollEvent(moveTo)) {
            if (moveTo.type == sf::Event::MouseButtonPressed) {
                moveTo.type = sf::Event::MouseButtonPressed;
                size_t destX = moveTo.mouseButton.x / 64;
                size_t destY = moveTo.mouseButton.y / 64;
                std::pair<size_t, size_t> destPos(destY, destX);

                if (destPos.first == position.first && destPos.second == position.second) {
                    if (piece == ’P’) {
                        gameGrid.at(position.first)[position.second] = ’r’;
                    } else if (piece == ’p’) {
                        gameGrid.at(position.first)[position.second] = ’b’;
                    } else if (piece == ’M’) {
                        gameGrid.at(position.first)[position.second] = ’R’;
                    } else if (piece == ’m’) {
                        gameGrid.at(position.first)[position.second] = ’B’;
                    }
                    return;
                }
                if (piece == ’P’) {
                    if (can_move(piece, position, destPos)) {
                        if (destPos.first == 0) {
                            gameGrid.at(destPos.first)[destPos.second] = ’R’;
                        } else {
                            gameGrid.at(destPos.first)[destPos.second] = ’r’;
                        }
                        gameGrid.at(position.first)[position.second] = ’t’;
                        turn++;
                        pick = true;
```

```cpp
                    }
                } else if (piece == 'p') {
                    if (can_move(piece, position, destPos)) {
                        if (destPos.first == 7) {
                            gameGrid.at(destPos.first)[destPos.second] = 'B';
                        } else {
                            gameGrid.at(destPos.first)[destPos.second] = 'b';
                        }
                        gameGrid.at(position.first)[position.second] = 't';
                        turn++;
                        pick = true;
                    }
                } else if (piece == 'M') {
                    if (can_move(piece, position, destPos)) {
                        gameGrid.at(destPos.first)[destPos.second] = 'R';
                        gameGrid.at(position.first)[position.second] = 't';
                        turn++;
                        pick = true;
                    }
                } else if (piece == 'm') {
                    if (can_move(piece, position, destPos)) {
                        gameGrid.at(destPos.first)[destPos.second] = 'B';
                        gameGrid.at(position.first)[position.second] = 't';
                        turn++;
                        pick = true;
                    }
                } else {
                    if (piece == 'P') {
                        gameGrid.at(position.first)[position.second] = 'r';
                    } else if (piece == 'p') {
                        gameGrid.at(position.first)[position.second] = 'b';
                    }
                }
            }
        }
    }
}

bool Checkers::can_move(char piece, std::pair<size_t, size_t> position,
                            std::pair<size_t, size_t> destPosition) {
    if (gameGrid.at(destPosition.first)[destPosition.second] == 't') {
        if (piece == 'P') {
            if (destPosition.first == position.first - 1) {
                if (destPosition.second == position.second + 1) {
                    return true;
                } else if (destPosition.second == position.second - 1) {
                    return true;
                }
            } else if (destPosition.first == position.first - 2) {
                if (destPosition.second == position.second + 2) {
                    if (remove_piece(piece, position, destPosition)) {
                        return true;
                    }
                    return false;
                } else if (destPosition.second == position.second - 2) {
                    if (remove_piece(piece, position, destPosition)) {
                        return true;
                    }
                    return false;
                }
            }
        } else if (piece == ’p’) {
            if (destPosition.first == position.first + 1) {
                if (destPosition.second == position.second + 1) {
                    return true;
                } else if (destPosition.second == position.second - 1) {
                    return true;
                }
            } else if (destPosition.first == position.first + 2) {
                if (destPosition.second == position.second + 2) {
                    if (remove_piece(piece, position, destPosition)) {
                        return true;
                    }
                    return false;
                } else if (destPosition.second == position.second - 2) {
                    if (remove_piece(piece, position, destPosition)) {
                        return true;
                    }
                    return false;
                }
            }
        } else if (piece == ’M’) {
            if (destPosition.first == position.first - 1) {
                if (destPosition.second == position.second + 1) {
                    return true;
                } else if (destPosition.second == position.second - 1) {
                    return true;
                }
            } else if (destPosition.first == position.first + 1) {
                if (destPosition.second == position.second + 1) {
                    return true;
                } else if (destPosition.second == position.second - 1) {
                    return true;
                }
            } else if (destPosition.first == position.first - 2) {
                if (destPosition.second == position.second + 2) {
                    if (remove_piece(piece, position, destPosition)) {
                        return true;
                    }
                    return false;
                } else if (destPosition.second == position.second - 2) {
                    if (remove_piece(piece, position, destPosition)) {
                        return true;
                    }
                    return false;
                }
            } else if (destPosition.first == position.first + 2) {
                if (destPosition.second == position.second + 2) {
                    if (remove_piece(piece, position, destPosition)) {
                        return true;
                    }
                    return false;
                } else if (destPosition.second == position.second - 2) {
                    if (remove_piece(piece, position, destPosition)) {
                        return true;
                    }
                    return false;
                }
            }
        } else if (piece == ’m’) {
            if (destPosition.first == position.first - 1) {
                if (destPosition.second == position.second + 1) {
                    return true;
                } else if (destPosition.second == position.second - 1) {
                    return true;
                }
            } else if (destPosition.first == position.first + 1) {
                if (destPosition.second == position.second + 1) {
                    return true;
                } else if (destPosition.second == position.second - 1) {
                    return true;
                }
            } else if (destPosition.first == position.first - 2) {
                if (destPosition.second == position.second + 2) {
                    if (remove_piece(piece, position, destPosition)) {
                        return true;
                    }
                    return false;
                } else if (destPosition.second == position.second - 2) {
                    if (remove_piece(piece, position, destPosition)) {
                        return true;
                    }
                    return false;
                }
            } else if (destPosition.first == position.first + 2) {
                if (destPosition.second == position.second + 2) {
                    if (remove_piece(piece, position, destPosition)) {
                        return true;
                    }
                    return false;
                } else if (destPosition.second == position.second - 2) {
                    if (remove_piece(piece, position, destPosition)) {
                        return true;
                    }
                    return false;
                }
            }
        }
    }
    return false;
}

bool Checkers::remove_piece(char piece, std::pair<size_t, size_t> position,
                    std::pair<size_t, size_t> destPosition) {
    if (piece == ’P’) {
        if (destPosition.second == position.second + 2) {
            if (gameGrid.at(destPosition.first + 1)[destPosition.second - 1] == ’b’ ||
                    gameGrid.at(destPosition.first + 1)[destPosition.second - 1] == ’B’) {
                gameGrid.at(destPosition.first + 1)[destPosition.second - 1] = ’t’;
                return true;
            }
        } else if (destPosition.second == position.second - 2) {
            if (gameGrid.at(destPosition.first + 1)[destPosition.second + 1] == 'b' ||
                    gameGrid.at(destPosition.first + 1)[destPosition.second + 1] == 'B') {
                gameGrid.at(destPosition.first + 1)[destPosition.second + 1] = 't';
                return true;
            }
        }
    } else if (piece == 'p') {
        if (destPosition.second == position.second + 2) {
            if (gameGrid.at(destPosition.first - 1)[destPosition.second - 1] == 'r' ||
                    gameGrid.at(destPosition.first - 1)[destPosition.second - 1] == 'R') {
                gameGrid.at(destPosition.first - 1)[destPosition.second - 1] = 't';
                return true;
            }
        } else if (destPosition.second == position.second - 2) {
            if (gameGrid.at(destPosition.first - 1)[destPosition.second + 1] == 'r' ||
                    gameGrid.at(destPosition.first - 1)[destPosition.second + 1] == 'R') {
                gameGrid.at(destPosition.first - 1)[destPosition.second + 1] = 't';
                return true;
            }
        }
    } else if (piece == 'M') {
        if (destPosition.second == position.second + 2) {
            if (destPosition.first == position.first + 2) {
                if (gameGrid.at(destPosition.first - 1)[destPosition.second - 1] == 'b' ||
                        gameGrid.at(destPosition.first - 1)[destPosition.second - 1] == 'B') {
                    gameGrid.at(destPosition.first - 1)[destPosition.second - 1] = 't';
                    return true;
                }
            } else if (destPosition.first == position.first  - 2) {
                if (gameGrid.at(destPosition.first + 1)[destPosition.second - 1] == 'b' ||
                            gameGrid.at(destPosition.first + 1)[destPosition.second - 1] == 'B') {
                    gameGrid.at(destPosition.first + 1)[destPosition.second - 1] = 't';
                    return true;
                }
            }
        } else if (destPosition.second == position.second - 2) {
            if (destPosition.first == position.first + 2) {
                if (gameGrid.at(destPosition.first - 1)[destPosition.second + 1] == 'b' ||
                        gameGrid.at(destPosition.first - 1)[destPosition.second + 1] == 'B') {
                    gameGrid.at(destPosition.first - 1)[destPosition.second + 1] = 't';
                    return true;
                }
            } else if (destPosition.first == position.first  - 2) {
                if (gameGrid.at(destPosition.first + 1)[destPosition.second + 1] == 'b' ||
                            gameGrid.at(destPosition.first + 1)[destPosition.second + 1] == 'B') {
                    gameGrid.at(destPosition.first + 1)[destPosition.second + 1] = 't';
                    return true;
                }
            }
        }
    } else if (piece == 'm') {
        if (destPosition.second == position.second + 2) {
            if (destPosition.first == position.first + 2) {
                if (gameGrid.at(destPosition.first - 1)[destPosition.second - 1] == 'r' ||
                        gameGrid.at(destPosition.first - 1)[destPosition.second - 1] == 'R') {
                    gameGrid.at(destPosition.first - 1)[destPosition.second - 1] = 't';
                    return true;
                }
            } else if (destPosition.first == position.first  - 2) {
                if (gameGrid.at(destPosition.first + 1)[destPosition.second - 1] == 'r' ||
                            gameGrid.at(destPosition.first + 1)[destPosition.second - 1] == 'R') {
                    gameGrid.at(destPosition.first + 1)[destPosition.second - 1] = 't';
                    return true;
                }
            }
        } else if (destPosition.second == position.second - 2) {
            if (destPosition.first == position.first + 2) {
                if (gameGrid.at(destPosition.first - 1)[destPosition.second + 1] == 'r' ||
                        gameGrid.at(destPosition.first - 1)[destPosition.second + 1] == 'R') {
                    gameGrid.at(destPosition.first - 1)[destPosition.second + 1] = 't';
                    return true;
                }
            } else if (destPosition.first == position.first  - 2) {
                if (gameGrid.at(destPosition.first + 1)[destPosition.second + 1] == 'r' ||
                            gameGrid.at(destPosition.first + 1)[destPosition.second + 1] == 'R') {
                    gameGrid.at(destPosition.first + 1)[destPosition.second + 1] = 't';
                    return true;
                }
            }
        }
    }
    return false;
}

bool Checkers::red_Won() {
    int black = std::count_if(gameGrid.begin(), gameGrid.end(), [](char* arr) -> bool {
        int found = 0;
        for (int i = 0; i < 8; i++) {
            if (arr[i] == ’b’ || arr[i] == ’B’) {
                found++;
            }
        }
        return found;
    });
    return !black;
}

bool Checkers::black_Won() {
    int red = std::count_if(gameGrid.begin(), gameGrid.end(), [](char* arr) -> bool {
        int found = 0;
        for (int i = 0; i < 8; i++) {
            if (arr[i] == ’r’ || arr[i] == ’R’) {
                found++;
            }
        }
        return found;
    });
    return !red;
}

void Checkers::draw(sf::RenderTarget& target, sf::RenderStates states) const {
    std::vector<char[8]>::const_iterator pos;
    size_t x;
    size_t y;

    sf::RectangleShape tile(sf::Vector2f(64, 64));
    sf::Sprite piece;

    for (pos = gameGrid.begin(), x = 0; pos != gameGrid.end(); pos++, x++) {
        for (y = 0; y < BOARD_DIMENSION; y++) {
            switch ((*pos)[y]) {
                case ’T’:
                    tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    tile.setFillColor(sf::Color::Red);
                    target.draw(tile);
                    break;
                case ’r’:
                    tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    tile.setFillColor(sf::Color::Black);
                    target.draw(tile);
                    piece.setTexture(redPiece);
                    piece.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    target.draw(piece);
                    break;
                case ’R’:
                    tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    tile.setFillColor(sf::Color::Black);
                    target.draw(tile);
                    piece.setTexture(redKing);
                    piece.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    target.draw(piece);
                    break;
                case ’t’:
                    tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    tile.setFillColor(sf::Color::Black);
                    target.draw(tile);
                    break;
                case ’b’:
                    tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    tile.setFillColor(sf::Color::Black);
                    target.draw(tile);
                    piece.setTexture(blackPiece);
                    piece.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    target.draw(piece);
                    break;
                case ’B’:
                    tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    tile.setFillColor(sf::Color::Black);
                    target.draw(tile);
                    piece.setTexture(blackKing);
                    piece.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    target.draw(piece);
                    break;
                case ’P’:
                    tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    tile.setFillColor(sf::Color::Blue);
                    target.draw(tile);
                    piece.setTexture(redPiece);
                    piece.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    target.draw(piece);
                    break;
                case ’p’:
                    tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    tile.setFillColor(sf::Color::Blue);
                    target.draw(tile);
                    piece.setTexture(blackPiece);
                    piece.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    target.draw(piece);
                    break;
                case ’M’:
                    tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    tile.setFillColor(sf::Color::Blue);
                    target.draw(tile);
                    piece.setTexture(redKing);
                    piece.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    target.draw(piece);
                    break;
                case ’m’:
                    tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    tile.setFillColor(sf::Color::Blue);
                    target.draw(tile);
                    piece.setTexture(blackKing);
                    piece.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    target.draw(piece);
                    break;
                case ’a’:
                    tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                    tile.setFillColor(sf::Color::Blue);
                    target.draw(tile);
                    break;
                default: break;
            }
        }
    }
}

std::ostream& operator<<(std::ostream& outStream, const Checkers& object) {
    std::vector<char[8]>::const_iterator pos;
    size_t i;

    for (pos = object.gameGrid.begin(); pos != object.gameGrid.end(); pos++) {
        for (i = 0; i < object.BOARD_DIMENSION; i++) {
            outStream << (*pos)[i];
        }
        outStream << std::endl;
    }
    return outStream;
}
```

5 PS5: DNA Alignment

5.1 Discussion

What I accomplished

The goal of this project was to create an algorithm that finds the optimal alignment of two DNA strands using dynamic programming. The program will measure the similarity of two genetic sequences by finding the "Edit-distance." The edit distance is calculated by finding the sum of all costs. There are three different cost operations, inserting a gap cost 2, aligning two unidentical characters costs 1, and aligning two identical characters has a cost of 0. To implement this, I created an EDistance class. The class contains a constructor that takes two `std::string`s representing two strands of DNA. Also, member functions `penalty()` that returns the cost of aligning the two characters given, `min()` that finds the minimum value of three integers, `optDistance()` which computes the optimization matrix and returns the value at `[0][0]` which is the optimal distance, `alignment()` that traces the matrix and returns a string that is the actual alignment of the two strands, and `display_matirx()` that outputs each element in the matrix. Along with four private variables `opt`, which is a 2D integer array that represents the matrix, `sampleA` and `sampleB` which are two `std::string`, `M` and `N` which are lengths `sampleA` and `sampleB` but also the dimensions of the matrix. When running the program, the user enters a text file that contains the two DNA strands. The program then outputs the edit distance, a table displaying the optimal alignment with the cost of the alignment, and the execution time of the program.

Design and Features

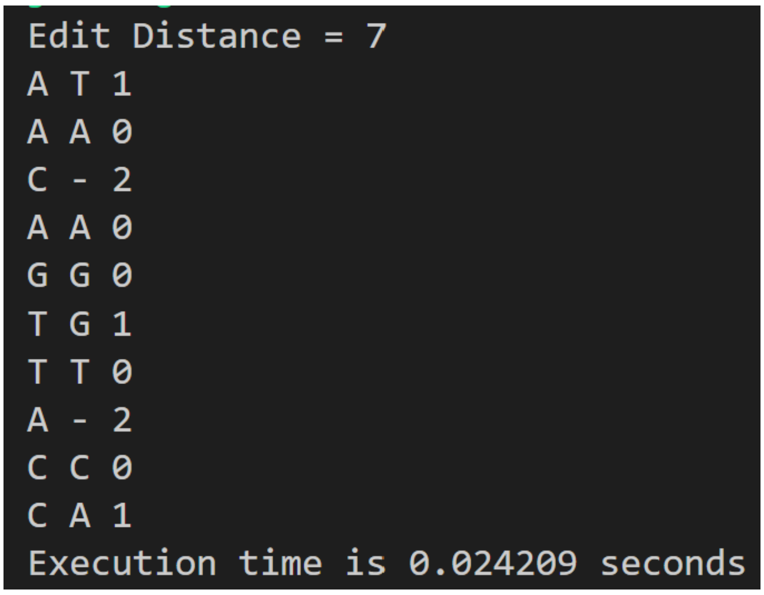
To find the alignment of the given pair of DNA samples, I created an `MxN` matrix (`opt`) with each element computed by finding the minimum value of three numbers. The first value is `opt[i+1][j+1]` divided by the penalty of aligning the two characters, found by passing the two characters into the `penalty()` function. Second is `opt[i+1][j] + 2`, and the third is `[i][j+1] + 2`. After the matrix is completed, the program iterates through the matrix and calculates the optimal alignment. It does this starting at `opt[0][0]` and moving diagonally, down, or right depending on the comparison of particular elements in the matrix, it does this unit [M][N] is reached. There are two main comparasions, `pt[i][j] == opt[i + 1][j + 1]` and `opt[i][j] == opt[i + 1][j + 1] + 1`. If the first comparison is true, there are three sub-comparisons checked, `sampleA[i] == sampleB[j]` means the characters are equal, `opt[i][j] == opt[i + 1][j] + 2` and `opt[i][j] == opt[i][j + 1] + 2` both mean they are not equal, and there is a gap inserted. If the second main comparison is true, then the characters are not equal, and no gap is inserted. During the described process, the 2 aligned strings are output to the screen along with the cost of the alignment (0 if equal, 1 if unequal characters, 2 if a character is aligned with a gap).

What I already knew and what I learned

Before completing this project, I used matrices and implemented them in multiple different ways, so iterating through the matrix and using different elements to solve a problem is something I am used to doing. I also knew how DNA alignment is performed, but I have never implemented it into a program.

I learned how to use dynamic programming to implement an algorithm. This project took a concept I am familiar with in the real world, so I found it interesting to create a program for it. It taught me that using programming skills and algorithms I am familiar with makes implementing real word problems simpler than I initially thought.

Below is a screenshot of a Sample DNA Alignment of the two DNA Strands:
Strand 1: AACAGTTACC
Strand 2: TAAGGTCA
Figure 5.



```
Edit Distance = 7
A T 1
A A 0
C - 2
A A 0
G G 0
T G 1
T T 0
A - 2
C C 0
C A 1
Execution time is 0.024209 seconds
```

Figure 5: Sample Alignment.

5.2 Codebase

```
CC = g++
CFLAGS = -g -Wall -Werror -pedantic -std=c++17
LIBS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
DEPS = EDistance.hpp
OBJS = EDistance.o

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

.PHONY: all clean lint

all: EDistance

EDistance: main.o $(OBJS)
	$(CC) $(CFLAGS) -o $@ $^ $(LIBS)

lint:
	cpplint *.cpp *.hpp

clean:
	rm *.o EDistance
```

```
// Copyright 2023 James Walsh
#include <iostream>
#include <fstream>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include "EDistance.hpp"

int main(int argc, char* argv[]) {
    std::ifstream file;
    std::string s1, s2;
    char* filename = argv[1];
```

```cpp
    sf::Clock clock;
    sf::Time t;

    file.open(filename);
    std::getline(file, s1);
    std::getline(file, s2);
    EDistance myEDistance(s1, s2);

    std::cout << "Edit Distance = " <<myEDistance.optDistance() << std::endl;
    std::cout << myEDistance.alignment();
    t = clock.getElapsedTime();
    std::cout << "Execution time is " << t.asSeconds() << " seconds\n";

    return 0;
}
```

```cpp
// Copyright 2023 James Walsh
#pragma once
#include <string>
#include <vector>
#include <algorithm>

class EDistance {
 public:
    EDistance(std::string stringA, std::string stringB);
    ~EDistance();
    static int penalty(char a, char b);
    static int min(int a, int b, int c);
    int optDistance();
    std::string alignment();
    void display_maxtrix();
 private:
    int** opt;
    std::string sampleA;
    std::string sampleB;
    int M;
    int N;
};
```

```cpp
// Copyright 2023 James Walsh
#include <algorithm>
#include <iostream>
#include "EDistance.hpp"

EDistance::EDistance(std::string stringA, std::string stringB):
    sampleA(stringA), sampleB(stringB), M(stringA.size()), N(stringB.size()) {
    opt = new int*[M + 2];
    for (int i = 0; i <= M; i++) {
        opt[i] = new int[N + 2];
    }
}

int EDistance::penalty(char a, char b) {
    auto penalty_point = [a, b]() -> int {
        if (a == b) {
            return 0;
        } else {
            return 1;
        }
    };
    return penalty_point();
}

int EDistance::min(int a, int b, int c) {
    if (a < b) {
        if (a < c) {
            return a;
        }
    } else {
        if (b < c) {
            return b;
        }
    }
    return c;
}

int EDistance::optDistance() {
    int i, j;

    opt[M][N] = 0;
    for (i = 0; i < M; i++) {
        opt[i][N] = 2 * (M - i);
    }
    for (j = 0; j < N; j++) {
        opt[M][j] = 2 * (N- j);
    }
    for (i = M - 1; i >= 0; i--) {
        for (j = N - 1; j >= 0; j--) {
            opt[i][j] = min(opt[i+1][j+1] + penalty(sampleA[i], sampleB[j]),
                opt[i+1][j] + 2, opt[i][j+1] + 2);
        }
    }

    return opt[0][0];
}

std::string EDistance::alignment() {
    std::string optimal;
    int i = 0, j = 0;

    while (i <= M - 1 && j <= N - 1) {
        if (opt[i][j] == opt[i + 1][j + 1]) {
            if (sampleA[i] == sampleB[j]) {
                optimal.push_back(sampleA[i]);
                optimal.push_back(’ ’);
                optimal.push_back(sampleB[j]);
                optimal.push_back(’ ’);
                optimal.push_back(’0’);
                i++;
                j++;
                optimal.push_back(’\n’);
            } else if (opt[i][j] == opt[i + 1][j] + 2) {
                optimal.push_back(sampleA[i]);
                optimal.push_back(’ ’);
                optimal.push_back(’-’);
                optimal.push_back(’ ’);
                optimal.push_back(’2’);
                i++;
                optimal.push_back('\n');
            } else if (opt[i][j] == opt[i][j + 1] + 2) {
                optimal.push_back(sampleA[i]);
                optimal.push_back(' ');
                optimal.push_back('-');
                optimal.push_back(' ');
                optimal.push_back('2');
                i++;
                optimal.push_back('\n');
            } else {
                i++;
                j++;
                optimal.push_back('\n');
            }
        } else if (opt[i][j] == opt[i + 1][j + 1] + 1) {
            optimal.push_back(sampleA[i]);
            optimal.push_back(' ');
            optimal.push_back(sampleB[j]);
            optimal.push_back(' ');
            optimal.push_back('1');
            i++;
            j++;
            optimal.push_back('\n');
        } else {
            i++;
            j++;
        }
    }
    return optimal;
}

EDistance::~EDistance() {
    int i;
    for (i = 0; i <= M; i++) {
        delete[] opt[i];
    }
    delete[] opt;
}

void EDistance::display_maxtrix() {
    int i, j;
    for (i = 0; i <= M; i++) {
        for (j = 0; j <= N; j++) {
            std::cout.width(3); std::cout << opt[i][j] << "  ";
        }
        std::cout << std::endl;
    }
}
```

6 PS6: RandWriter

6.1 Discussion

What I accomplished

This project turns a given text into a random text that is somewhat reasonable and readable using a Markov model symbol table. To implement this, I created a RandWriter class. The object contains a constructor, five public member functions, and four private member variables. The constructor takes a `std::string` "text" and an int "k" as parameters, "text" is used as the original text and "k" is the length of each kgram. The `orderK()` is an accessor function that returns the value of the variable "K". The `freq()` function takes a `std::string` "kgram" as a parameter and returns the frequency of the given kgram in the symbol table. The `freq()` function is overloaded to take a `std::string` "kgram" and a char "c" and returns the frequency that c follows the kgram in the text. The `kRand()` function takes a `std::string` "kgram" and returns a random character that follows the given kgram. The random character is based on the frequency the given character follows the kgram in the original text. Lastly, the class contains a `generate()` function that takes a `std::string` "kgram" and an int "L". The function generates and returns a string of length "L" by simulating a trajectory through the symbol table or kgrams and kgrams+1. The class also overloads the insertion operator which prints out the symbol table to the given ostream.

Design and Features

When implementing this class, I decided to represent the symbol table as a `std::map` that contains a `std::string` and another`std::map` which contains a char and int. The string represents each kgram, the map represents the next character and its frequency, and the frequency of the kgram is found by finding the total of the next character frequencies. I decided to implement the symbol table this way because I found it easy to navigate as well as able to store the kgram and kgram + 1 frequency well and accessible.

I implemented the insertion operator as a friend function, so I was able to access the object's private member variables directly, however, the function takes a const RandWriter object ensuring the function cannot modify the member variables.

When producing random numbers for the kRand function, I used the c++ library <random>. Then I stored a `std::minstd_rand0` as a class member function. In the constructor, I created a seed and set it to the time. Lastly, in the kRand function, I declare a `std::uniform_int_distribution<int>` then pass the `std::minstd_rand0` as a parameter to calculate the random number.

What I already knew and what I learned

Before programming this assignment, I had experience using `std::map`, so that is why I decided to implement the symbol table as so. I also have used `std::string` often, so I was aware of its member functions which were helpful for this assignment. I learned how to use the <random> library when completing this project. I have used `rand()` before so I am aware of the usage of seeds, but I have not used <random> to generate random numbers. I also learned a lot about how using frequencies of the next character of kgrams makes a random text drastically more reasonable. If I programmed a random text generator that did not generate the next character by using frequencies from the original text, the text generated would be unreadable. However, implementing frequencies into the random character generator makes the test readable.

Unit Testing

For this program, I wrote four different unit tests using the Boost Unit Test Framework. All tests created a RandWriter object with "gagggagaggcgagaaa" as the text and 1 as k. I tested all member functions of the calls, starting with `orderK()`. For this function, I simply checked if it returned the proper 'K' value, 1. I did this using `BOOST_REQUIRE_EQUAL()`. The purpose of the next three tests was to see if the functions

throw and did not throw a `std::runtime_error` when appropriate. These test tested both versions of *freq()*, `kRand()`, and `generate()`. To do this I used `BOOST_REQUIRE_THROW()` and `BOOST_REQUIRE_NO_THROW()`. My program passed all tests.
Below is a result of the generating function, using the first paragraph as the original text, 3 as "K" and 1000 as "L".

```
This, and reate() is a std::string "kgrams+1.  The symbol table.  The
constring "k" as privated to a character function that constring "kgrameter
is overloaded on geream.  The variable used to this paracter class a given
kgram in the freq() is based a param" and returns a Markov model symbol take
and kgram.  The frequency that follows the kRand returns a random character
function the symbol text.  Lastly, that take a trajector follows the the
kgram, the kRand readable "K".  The rand returns the given kgram" as a in
table.  The frequency of the object cons, an accessor functionstreate()
functions, "text" and as the in the to a std::string a strints overloads the
text.  Lastly, the symbol text into table.  The text and returns a
std::structor takes a constring of the function generated to a string "text
the to a std::string "kgram" and returns a std::structor which kgram in the
ins that the rand readable and reated to a cons, "text" is out takes a
std::structor follows that returns the class cont
```

6.2 Codebase

```
CC = g++
CFLAGS = -Wall -Werror -pedantic -std=c++17 -g
LIBS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
    lboost_unit_test_framework
DEPS = RandWriter.hpp
OBJS = RandWriter.o

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

.PHONY: all clean lint

all: TextWriter test

TextWriter: TextWriter.o $(OBJS)
  $(CC) $(CFLAGS) -o $@ $^ $(LIBS)

test: $(OBJS) test.o
  $(CC) $(FLAGS) -o test $^ $(LIBS)

lint:
  cpplint *.cpp *.hpp

clean:
  rm *.o TextWriter test
```

```
// Copyright 2023 James Walsh
#include <iostream>
#include <fstream>
#include "RandWriter.hpp"

int main(int argc, char* argv[]) {
    std::string text;
    std::string current_str;
    std::ifstream textFile;
    char* k_arr = argv[1];
    char* l_arr = argv[2];
    char* filename = argv[3];
    int k = atoi(k_arr);
    int l = atoi(l_arr);

    textFile.open(filename);
    while (!textFile.eof()) {
        textFile >> current_str;
        text.append(current_str);
        text.push_back(' ');
        current_str.clear();
    }
    text.pop_back();

    RandWriter myWriter(text, k);
    std::string resulting_text;
    std::string first_gram;
    for (auto i = 0; i < k; i++) {
        first_gram.push_back(text.at(i));
    }

    resulting_text = myWriter.generate(first_gram, l);

    std::cout << resulting_text << std::endl;

    return 0;
}
```

```
// Copyright 2023 James Walsh
#include <string>
#include <map>
#include <random>

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

    // Order k of Markov model
    int orderK() const { return K; }

    // Number of occurences of kgram in text
    // Throw an exception if kgram is not length k
    int freq(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(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(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 excpetion if kgram is not of length k.
    // Assume that L is at least k
    std::string generate(std::string kgram, int L);

    friend std::ostream& operator<<(std::ostream& Out, const RandWriter& object);

 private:
    std::map<std::string, std::map<char, int>> table;
    std::minstd_rand0 gen;
    int textLen;
    int K;
};
// Overload the stream insertion operator << and display the internal state
// of the Markov model. Print out the order, alphabet, and the frequencies
// of the k-grams and k+1-grams

std::ostream& operator<<(std::ostream& out, const RandWriter& object);
```

```
// Copyright 2023 James Walsh
#include <iostream>
#include <ios>
#include <algorithm>
#include <numeric>
#include <utility>
#include <vector>
#include <chrono>
#include <stdexcept>
#include "RandWriter.hpp"

RandWriter::RandWriter(std::string text, int k):textLen(text.size()), K(k) {
    int i, j, c;
    std::string current_gram;
    std::map<std::string, std::map<char, int>>::iterator kgram;
    std::map<char, int>::iterator next_char;

    for (i = 0, j = 0; i < textLen - k; i++) {
        current_gram.clear();
        while (j < k) {
            c = text.at(i + j);
            if (c >= 0 && c < 127) {
                current_gram.push_back(text.at(i + j));
            }
            j++;
        }
        if (i == 0) {
            text.append(current_gram);
        }
        j--;
        kgram = table.find(current_gram);
        // if the table already contains the kgram
        if (kgram != table.end()) {
            next_char = (*kgram).second.find(text.at(i + j + 1));
            // if the kgram already contains the next possible char
            if (next_char != (*kgram).second.end()) {
                (*next_char).second++;
            } else {
            // if the kgram doesnt contain the next possible char
                (*kgram).second.insert(std::pair<char, int>(text.at(i + j + 1), 1));
            }
        } else {
```

```cpp
            // if the table doesnt contain the kgram
            table.insert(std::pair<std::string,
                            std::map<char, int>>(current_gram, std::map<char, int>()));
            kgram = table.find(current_gram);
            (*kgram).second.insert(std::pair<char, int>(text.at(i + j + 1), 1));
        }
        j = 0;
    }
    unsigned int seed = std::chrono::system_clock::now().time_since_epoch().count();
    gen = std::minstd_rand0(seed);
}

int RandWriter::freq(std::string kgram) const {
    std::map<std::string, std::map<char, int>>::const_iterator kgram_iter;
    std::map<char, int>::const_iterator next_char;
    int totalFreq = 0;

    if (static_cast<int>(kgram.size()) != K) {
        throw std::runtime_error("Invalid kgram length.");
    }
    kgram_iter = table.find(kgram);
    if (kgram_iter != table.end()) {
        totalFreq = std::accumulate((*kgram_iter).second.begin(),
            (*kgram_iter).second.end(), 0, [](int &totalFreq, auto next_char) {
            return totalFreq += next_char.second;
        });
    }
    return totalFreq;
}

int RandWriter::freq(std::string kgram, char c) const {
    std::map<std::string, std::map<char, int>>::const_iterator kgram_iter;
    std::map<char, int>::const_iterator next_char;
    int totalFreq = 0;

    if (static_cast<int>(kgram.size()) != K) {
        throw std::runtime_error("Invalid kgram length.");
    }
    kgram_iter = table.find(kgram);
    if (kgram_iter != table.end()) {
        for (next_char = (*kgram_iter).second.begin();
            next_char != (*kgram_iter).second.end(); next_char++) {
            if ((*next_char).first == c) {
                totalFreq = (*next_char).second;
                return totalFreq;
            }
        }
    }
    return totalFreq;
}

char RandWriter::kRand(std::string kgram) {
    std::map<std::string, std::map<char, int>>::const_iterator kgram_iter;
    std::map<char, int>::const_iterator next_char;
    std::vector<char> char_list;
    int i;
    int krandNext = 0;

    kgram_iter = table.find(kgram);
    if (static_cast<int>(kgram.size()) != K) {
        throw std::runtime_error("Invalid kgram length.");
    }
    if (kgram_iter != table.end()) {
        for (next_char = (*kgram_iter).second.begin();
                next_char != (*kgram_iter).second.end(); next_char++) {
            for (i = 0; i < (*next_char).second; i++) {
                char_list.push_back((*next_char).first);
                krandNext++;
            }
        }
        std::uniform_int_distribution<int> dist(0, krandNext - 1);
        krandNext = dist(gen);
        return char_list.at(krandNext);
    } else {
        throw std::runtime_error("No such kgram found.");
    }
    return char();
}

std::string RandWriter::generate(std::string kgram, int L) {
    std::string result;
    std::string current_gram;
    int i, j;
    char c;
    if (static_cast<int>(kgram.size()) != K) {
        throw std::runtime_error("Invalid kgram length.");
    } else {
        result.append(kgram);
        c = kRand(kgram);
        result.push_back(c);
        for (i = 1; i < L - K; i++) {
            for (j = 0; j < static_cast<int>(K); j++) {
                current_gram.push_back(result.at(i + j));
            }
            c = kRand(current_gram);
            current_gram.clear();
            result.push_back(c);
        }
    }
    return result;
}

std::ostream& operator<<(std::ostream& out, const RandWriter& object) {
    std::map<std::string, std::map<char, int>>::const_iterator kgram;
    std::map<char, int>::const_iterator next_char;
    kgram = object.table.begin();

    for (kgram = object.table.begin(); kgram != object.table.end(); kgram++)
    {
        out << (*kgram).first << std::endl;
        for (next_char = (*kgram).second.begin(); next_char != (*kgram).
   second.end(); next_char++) {
            out << "\t" << (*next_char).first << " feq: " << (*next_char).
   second << std:: endl;
```

```
        }
        out << std::endl;
    }
    return out;
}
```

```
// Copyright 2123 James Walsh

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

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

BOOST_AUTO_TEST_CASE(test_orderK) {
    RandWriter test("gagggagaggcgagaaa", 1);
    BOOST_REQUIRE_EQUAL(test.orderK(), 1);
}

BOOST_AUTO_TEST_CASE(test_freq) {
    RandWriter test("gagggagaggcgagaaa", 1);
    BOOST_REQUIRE_THROW(test.freq("ga"), std::runtime_error);
    BOOST_REQUIRE_NO_THROW(test.freq("g"));
    BOOST_REQUIRE_EQUAL(test.freq("g"), 9);
    BOOST_REQUIRE_THROW(test.freq("ga", 'g'), std::runtime_error);
    BOOST_REQUIRE_NO_THROW(test.freq("g", 'g'));
    BOOST_REQUIRE_EQUAL(test.freq("g", 'a'), 5);
}

BOOST_AUTO_TEST_CASE(test_kRand) {
    RandWriter test("gagggagaggcgagaaa", 1);
    BOOST_REQUIRE_THROW(test.kRand("ga"), std::runtime_error);
    BOOST_REQUIRE_THROW(test.kRand("b"), std::runtime_error);
    BOOST_REQUIRE_NO_THROW(test.kRand("g"));
}

BOOST_AUTO_TEST_CASE(test_generate) {
    RandWriter test("gagggagaggcgagaaa", 1);
    BOOST_REQUIRE_THROW(test.generate("ga", 10), std::runtime_error);
    BOOST_REQUIRE_NO_THROW(test.generate("g", 10));
    BOOST_REQUIRE_EQUAL(test.generate("g", 10).size(), 10);
}
```

7 PS7: Kronos Log Parsing

7.1 Discussion

What I accomplished

The purpose of this project review InTouch log(s) of a Kronos InTouch device. A log file contains information about the operations of the device. The program scans the files looking for lines that indicate a boot-up, which looks like `(log.c.166) server started`. After the program looks for a line that marks the completion of the boot-up sequence, the line will include `oejs.AbstractConnector:Started SelectChannelConnector`. Both of these lines will also include the log file line number and the date and time of the particular boot-up or completion. A successful boot will start with a start message and will be followed by a completion message. However, this is not always the case. If a start message is followed by another start without a completion, this indicates an "incomplete boot".

Design and Features

My program scans the file searching for these indicators and creates a report file tracking each successful and unsuccessful boot. To do this I used regex expressions. I had two regex expressions, one for start and one for complete. While the program had not reached the end of the file, it reads the line, then used the `regex_search()` function to see if the line was a start or completion. If it was, the report is updated with the date and time of the start or completion. To indicate if the boot was complete or incomplete, I used a boolean `in_progress`. The boolean was set to true when a start line is found. Then is set to false when a completion line is found. However, if a start line is found while `in_progress` is true, the report file marks the current boot as incomplete. After a complete boot, the report file marks the time difference from the start line to the completion line as the "boot time", but my program does not implement the boot time algorithm properly. The report file uses the name of the log file but apends".rpt" to the end.

The regex expressions I used were:

Start boot-up: `std::regex start("(log.c.d{1,3}) server started")`

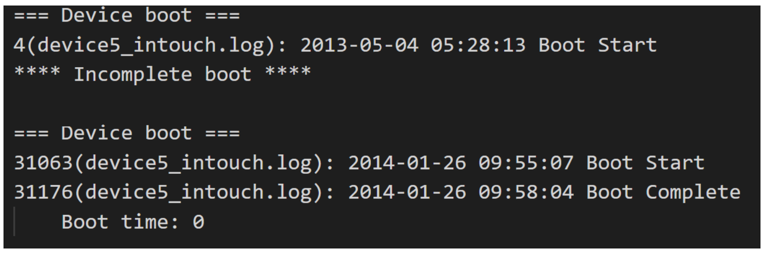
Completion of boot: `std::regex testC("oejs.AbstractConnector:Started SelectChannelConnector")`

What I already knew and what I learned

Going into this project I was familiar with input and output files. I have created programs that opened files, read until the end of the file, and done a variety of things with the data acquired many times. Also, I have outputted data to different files in various formats as well.

When completing this project I learned and gained a lot of knowledge on using regex expressions. Regex is something brand knew to me so it took me time to get used to how an expression works, and how I could then use that expression to perform some time of a task. I found the library relatively easy to use, and I found implementing them into my code to perform a task straightforward. I can see regex expressions being very useful for me in the future, and will definitely be something I find myself using when programming.

Below is an example result file of a Complete and Incomplete boot: Figure 6.



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

=== Device boot ===
31063(device5_intouch.log): 2014-01-26 09:55:07 Boot Start
31176(device5_intouch.log): 2014-01-26 09:58:04 Boot Complete
    Boot time: 0
```

Figure 6: Sample.

7.2 Codebase

```makefile
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic
LIB = -lboost_unit_test_framework -lboost_regex

.PHONY: all clean lint

all: ps7

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

ps7: main.o
  $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
  rm *.o ps7

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

```cpp
// Copyright 2023 James Walsh
#include <iostream>
#include <fstream>
#include <string>
#include <boost/regex.hpp>
#include "boost/date_time/gregorian/gregorian.hpp"

char* create_output_file(char* input);

int main(int argc, char* argv[]) {
  std::ifstream inputFile;
  std::ofstream outputFile;
  char* filename = argv[1];
  char* output = create_output_file(filename);

  inputFile.open(filename);
  outputFile.open(output);

  boost::regex start("\\(log\\.c\\.\\d{1,3}\\) server started");
  boost::regex complete(
    "\\oejs\\.AbstractConnector:Started SelectChannelConnector");

  std::string date = "";
  std::string time = "";
  std::string startTime;
  std::string curretLog = "";
  std::string log;
  int lineNum = 1;
  int bootTime = 0;
  bool in_progress;
  bool first = true;
  int i;
  char c;

  while (!inputFile.eof()) {
    char line[1000];
    inputFile.getline(line, 999);
    i = 28;
    if (regex_search(line, start) && first) {
      c = line[i];
      while (c != ’)’) {
        curretLog.push_back(c);
        i++;
        c = line[i];
      }
      i = 0;
      c = line[i];
      while (c != ’ ’) {
        date.push_back(c);
        i++;
        c = line[i];
      }
      i++;
      c = line[i];
      while (c != ’ ’) {
        time.push_back(c);
        i++;
        c = line[i];
      }
      time.pop_back();
      startTime = time;
      outputFile << "=== Device boot ===" << std::endl;
      outputFile << lineNum << "(" << filename << "): ";
      outputFile << date <<  " " << time << " Boot Start" << std::endl;
      std::cout << line << std::endl;
      first = false;
      in_progress = true;
    } else if (regex_search(line, start) && !in_progress) {
      c = line[i];
      while (c != ’)’) {
        curretLog.push_back(c);
        i++;
        c = line[i];
      }
      i = 0;
      c = line[i];
      while (c != ’ ’) {
        date.push_back(c);
        i++;
        c = line[i];
      }
      i++;
      c = line[i];
      while (c != ’ ’) {
        time.push_back(c);
        i++;
        c = line[i];
      }
      time.pop_back();
      startTime = time;
      outputFile << "=== Device boot ===" << std::endl;
      outputFile << lineNum << "(" << filename << "): ";
      outputFile << date <<  " " << time << " Boot Start" << std::endl;
      std::cout << line << std::endl;
      in_progress = true;
    } else if (regex_search(line, start) && in_progress) {
      c = line[i];
      while (c != ')') {
        log.push_back(c);
        i++;
        c = line[i];
      }
      if (curretLog != log) {
        i = 0;
        c = line[i];
        while (c != ' ') {
          date.push_back(c);
          i++;
          c = line[i];
        }
        i++;
        c = line[i];
        while (c != ' ') {
          time.push_back(c);
          i++;
          c = line[i];
        }
        time.pop_back();
        startTime = time;
        outputFile << "**** Incomplete boot ****" << std::endl << std::endl;
        outputFile << "=== Device boot ===" << std::endl;
        outputFile << lineNum << "(" << filename << "): ";
        outputFile << date <<  " " << time << " Boot Start" << std::endl;
        std::cout << line << std::endl;
        in_progress = true;
        curretLog = log;
      }
    }
    if (regex_search(line, complete) && in_progress) {
      i = 0;
      c = line[i];
      while (c != ' ') {
        date.push_back(c);
        i++;
        c = line[i];
      }
      i++;
      c = line[i];
      while (c != '.') {
        time.push_back(c);
        i++;
        c = line[i];
      }
      outputFile << lineNum << "(" << filename << "): ";
      outputFile << date <<  " " << time << " Boot Complete" << std::endl;
      outputFile << "\tBoot time: " << bootTime << std::endl << std::endl;
      std::cout << line << std::endl;
      in_progress = false;
      curretLog.clear();
    }
    lineNum++;
    date.clear();
    time.clear();
    log.clear();
  }
  return 0;
```

```cpp
}

char* create_output_file(char* input) {
  std::string temp;
  temp.append(input);
  temp.append(".rpt");
  char* out = new char[temp.size()];
  int i = 0;
  std::for_each(temp.begin(), temp.end(), [&out, &i] (char c) {
    out[i] = c;
    i++;
  });

  return out;
}
```