Matt Zagame
COMP IV: Project Portfolio
Fall 2020

Contents:

PS0 Hello World with SFML

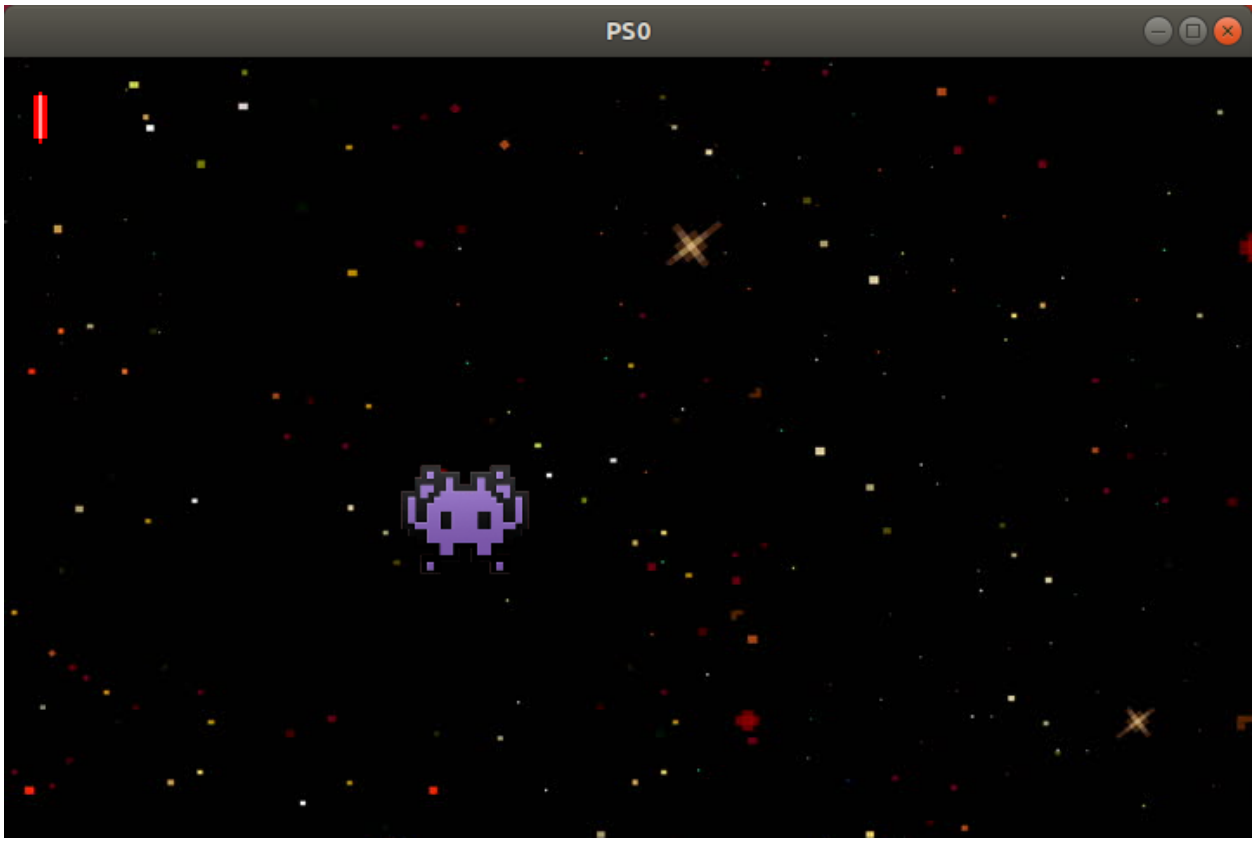
This first assignment was all about setting up a build environment on a virtual Linux machine and using it to demo the Simple and Fast Multimedia Libraries (SFML). After getting up and running on Linux, we were to extend the SFML demo code to have it draw our own sprite and respond to keyboard input.

What I Did:
I started with the demo code available on the SFML tutorial website which, when compiled and ran, provides a window with a green circle drawn in the middle. My approach to this assignment was to create a representation of a small arcade style game. I began setting up the window to be half the dimensions of the user's screen size, then I loaded in my own sprite images to replace the green circle with an alien and a background, and also set up a few game variables. Then, inside the window display loop I made an idle loop for the alien sprite when no keyboard input has been made yet. Once a directional key has been pressed, the alien sprite will move in the direction corresponding to the key pressed. I also added the ability to fire a laser beam from the alien by pressing the spacebar.

Key Algorithms, Data Structures, and Object-Oriented Designs:
This code consisted of a single main function and did not use an object-oriented design. The main objective of this assignment was to become more familiar with the Linux development environment and SFML, therefore no noteworthy algorithms or data structures were used.

What I Learned:
While I was already familiar with setting up a virtual machine, I still learned a lot about Linux and how to compile code using Makefiles and the Bash shell. I also learned about SFML and how it can be used to make applications with simple graphics, event handling, audio, etc. for all platforms. This library proves especially useful when it comes to programming 2D games using any of the multiple supported languages.

PS0 Demo



Makefile

```
all: main

main: main.o
    g++ -o main main.cpp -lsfml-graphics -lsfml-window -lsfml-system

clean:
    rm main main.o
```

main.cpp

```
main.cpp        Tue Dec 01 19:08:46 2020        1
    1: // Copyright 2020 Matt Zagame
    2: // SFML program that draws a sprite that can be controlled
    3:
    4: #include <SFML/Graphics.hpp>
    5:
    6: int main()
    7: {
    8:     sf::RenderWindow window(sf::VideoMode(
    9:       (sf::VideoMode::getDesktopMode().width / 2),
   10:       (sf::VideoMode::getDesktopMode().height / 2)), "PS0");
   11:     window.setFramerateLimit(60);
   12:
   13:     // space background
   14:     sf::Texture space_texture;
   15:     if (!space_texture.loadFromFile("space.jpg"))
   16:         return -1;
   17:     sf::Sprite background(space_texture);
   18:
   19:     // alien sprite
   20:     sf::Texture alien_texture;
   21:     if (!alien_texture.loadFromFile("sprite.png"))
   22:         return -1;
   23:     sf::Sprite sprite(alien_texture);
   24:     sprite.setScale(0.5, 0.5);
   25:
   26:     // initial position of alien
   27:     int xPos = window.getSize().x / 2;
   28:     int yPos = window.getSize().y / 2;
   29:     sprite.setPosition(xPos, yPos);
   30:
   31:     // laser beam sprite
   32:     sf::Texture laser_texture;
   33:     if (!laser_texture.loadFromFile("laserbeam.png"))
   34:         return -1;
   35:     sf::Sprite laser_beam(laser_texture);
   36:     laser_beam.setScale(0.25, 0.25);
   37:
   38:     // game vars
   39:     bool isIdle = true;
   40:     bool firingLaser = false;
   41:     sf::Clock clock;
   42:     sf::Time time;
   43:
   44:     while (window.isOpen())
   45:     {
   46:         // process events
   47:         sf::Event event;
   48:         while (window.pollEvent(event))
   49:         {
   50:             // close window
   51:             if (event.type == sf::Event::Closed)
   52:                 window.close();
   53:         }
   54:
   55:         // idle loop
   56:         if (isIdle) {
   57:             if (time.asSeconds() < 2) {
   58:                 sprite.move(-2, 0);
   59:             }
   60:             else if (time.asSeconds() > 2 && time.asSeconds() < 4) {
   61:                 sprite.move(2, 0);
```

main.cpp Tue Dec 01 19:08:46 2020 2

```
    62:             }
    63:             else if (time.asSeconds() > 4) {
    64:               time = clock.restart();
    65:             }
    66:             time = clock.getElapsedTime();
    67:         }
    68:
    69:         // keyboard inputs
    70:         if (sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {
    71:           sprite.move(0, -2);
    72:           isIdle = false;
    73:         }
    74:         else if (sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {
    75:           sprite.move(-2, 0);
    76:           isIdle = false;
    77:         }
    78:         else if (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) {
    79:           sprite.move(0, 2);
    80:           isIdle = false;
    81:         }
    82:         else if (sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {
    83:           sprite.move(2, 0);
    84:           isIdle = false;
    85:         }
    86:         else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
    87:           sprite.move(-2, 0);
    88:           isIdle = false;
    89:         }
    90:         else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {
    91:           sprite.move(2, 0);
    92:           isIdle = false;
    93:         }
    94:         else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
    95:           sprite.move(0, -2);
    96:           isIdle = false;
    97:         }
    98:         else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
    99:           sprite.move(0, 2);
   100:           isIdle = false;
   101:         }
   102:         else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space)) {
   103:           isIdle = false;
   104:           firingLaser = true;
   105:           laser_beam.setPosition(sprite.getPosition().x +
   106:           (sprite.getLocalBounds().width / 8), sprite.getPosition().y);
   107:         }
   108:
   109:         if (firingLaser) {
   110:           laser_beam.move(0, -10);
   111:         }
   112:
   113:         window.clear();
   114:         window.draw(background);
   115:         window.draw(laser_beam);
   116:         window.draw(sprite);
   117:         window.display();
   118:     }
   119:
   120:     return 0;
   121: }
```

PS1 Linear Feedback Shift Register and Image Encoding

This project was split into two parts, PS1a and PS1b. For PS1a, we were asked to implement a Fibonacci linear feedback shift register (LFSR) which would produce pseudo-random bits to be used for image encryption in PS1b. The assignment required the LFSR to take a 16-bit binary seed and XOR the designated tap bits from its current state to generate the new rightmost bit of its next state after shifting each bit once to the left. PS1b would then have a main function to take an image file as input and use the LFSR to encrypt its pixel data and output the result.

What I Did:
PS1a - FibLFSR stores a sequence of bits which are shifted to the left and uses tap bits to acquire the rightmost bit. I used a dynamic array of integers to store the supplied seed (input bits). The class has a generate() function which uses step() as a helper function to step through the LFSR. For this part of the assignment I used the Boost C++ libraries to run test cases on the FibLFSR class, this is documented in test.cpp. I made sure to test that both the step() and generate() functions produced the expected output, I also tested for an exception where a supplied seed was shorter/longer than the required 16 bits for the register.

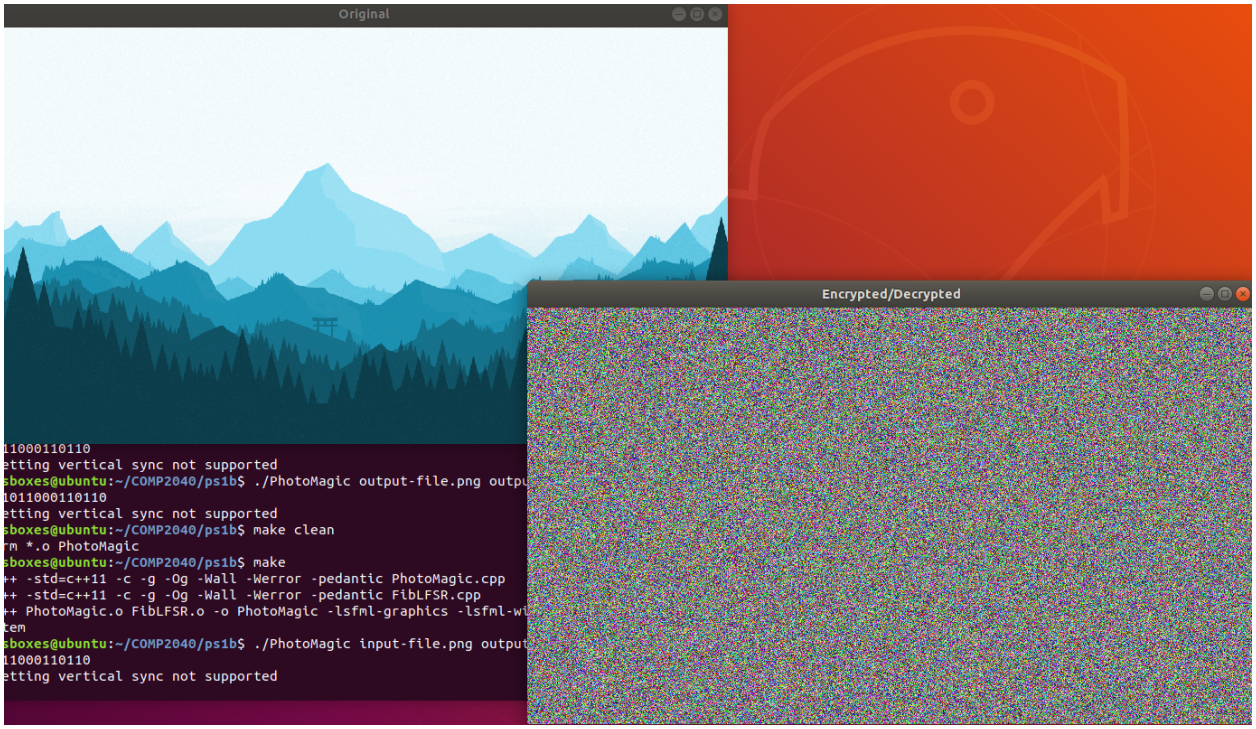
PS1b - PhotoMagic takes three command line arguments, the input filename, output filename, and a 16-bit binary seed. Once the command has been entered, a FibLFSR object is created and the transform() function is called on the image which encrypts or decrypts images by XORing pixel data with bits from the register. The program also checks to make sure all command line arguments are entered correctly. In the SFML display loop, two new windows will display both the original and the encrypted/decrypted image.

Key Algorithms, Data Structures, and OOP Designs:
The FibLFSR class is a container for a dynamic array which has a size equal to the length of the binary seed passed to it upon construction, however for this assignment an exception is thrown if the string is not exactly 16 bits long. The PhotoMagic class will handle any exceptions that may be thrown by FibLFSR. Since FibLFSR has dynamically allocated memory, the appropriate copy/move/destructor functions are present in the class.

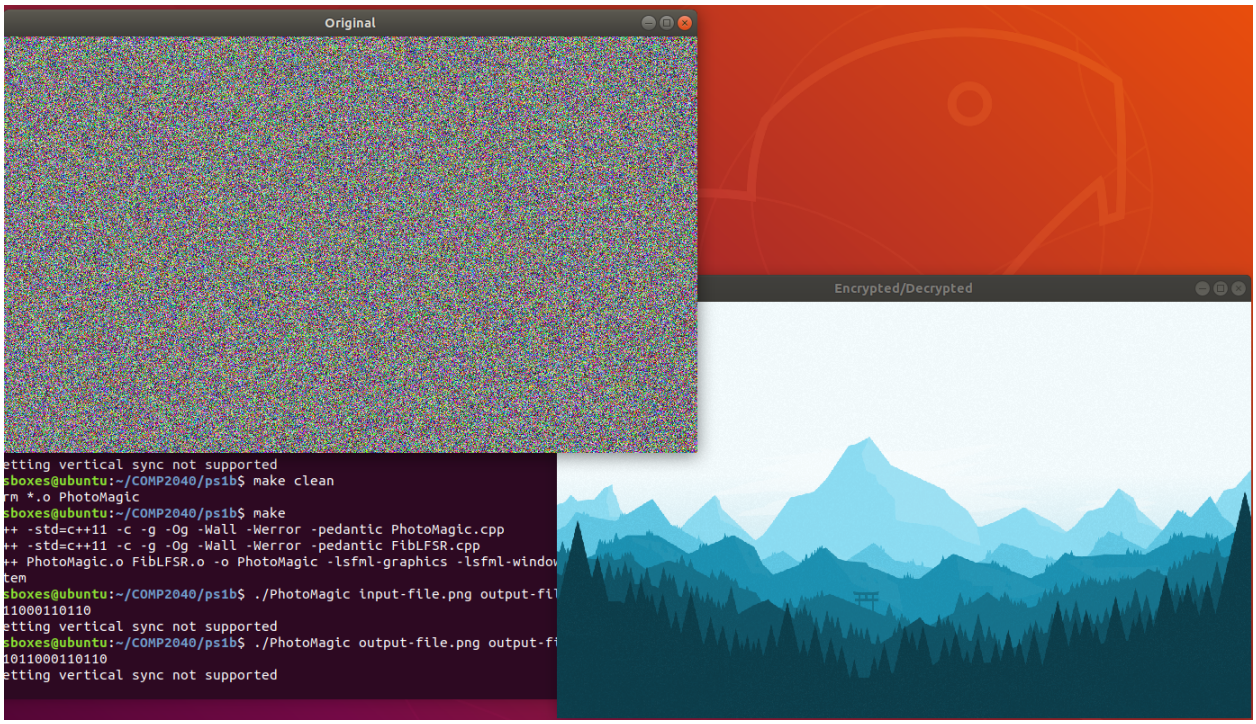
What I Learned:
This assignment was a good exercise in reviewing C++ programming techniques I had learned previously, such as the rule of 5, operator overloading, exception handling, and SFML. This was also the first time I used header files to define classes and have them linked to my other source code. On top of that I also learned how to

use the Boost libraries to easily run unit tests on my FibLFSR class while it was still in development.

Encryption Process



Decryption Process



Makefile

```
CC = g++
CFLAGS = -std=c++11 -c -g -Og -Wall -Werror -pedantic
LIBS = -lsfml-graphics -lsfml-window -lsfml-system
OBJS = PhotoMagic.o FibLFSR.o
EXE = PhotoMagic

all : $(EXE)

PhotoMagic : $(OBJS)
    $(CC) $(OBJS) -o $(EXE) $(LIBS)

PhotoMagic.o : PhotoMagic.cpp
    $(CC) $(CFLAGS) PhotoMagic.cpp

FibLFSR.o : FibLFSR.cpp FibLFSR.h
    $(CC) $(CFLAGS) FibLFSR.cpp

clean :
    \rm *.o $(EXE)
```

FibLFSR.h

```
FibLFSR.h       Tue Dec 01 19:14:37 2020       1
    1: // Copyright 2020 Matt Zagame
    2: /** FibLFSR.h
    3:  * FibLFSR is a 16-bit linear feedback shift register
    4:  * Matt Zagame 9/19/2020
    5:  */
    6: #include <iostream>
    7: #include <exception>
    8: #include <math.h>
    9: #ifndef FIBLFSR_H
   10: #define FIBLFSR_H
   11: #define ASCII_OFFSET 48
   12:
   13: class FibLFSR {
   14: public:
   15:     FibLFSR();
   16:     FibLFSR(std::string seed);  // constructor to create FibLFSR with the
   17:                                 // given initial seed and tap
   18:     explicit FibLFSR(const FibLFSR& copyFibLFSR);
   19:     explicit FibLFSR(FibLFSR&& moveFibLFSR) noexcept;
   20:     ~FibLFSR();
   21:     FibLFSR& operator=(const FibLFSR& rvalue);
   22:     FibLFSR& operator=(FibLFSR&& rvalue) noexcept;
   23:
   24:     int step();             // simulate one step and return the new bit
   25:                             // as 0 or 1
   26:     int generate(int k);    // simulate k steps and return k-bit integer
   27:
   28:     // displays LFSR
   29:     friend std::ostream& operator<<(std::ostream& out, const FibLFSR&
   30:     flfsr);
   31:
   32: private:
   33:     int size;               // size of register
   34:     int* reg;               // register as an array of integers
   35: };
   36:
   37: #endif
```

FibLFSR.cpp

```
FibLFSR.cpp        Tue Dec 01 19:13:29 2020        1
    1: // Copyright 2020 Matt Zagame
    2: /** FibLFSR.cpp
    3:  * FibLFSR is a 16-bit linear feedback shift register
    4:  * Matt Zagame 9/19/2020
    5:  */
    6: #include "FibLFSR.h"
    7: #define TAP_BIT1 13
    8: #define TAP_BIT2 12
    9: #define TAP_BIT3 10
   10:
   11: /** constructors **/
   12: FibLFSR::FibLFSR() : size(0), reg(nullptr) {}
   13:
   14: FibLFSR::FibLFSR(std::string seed) {
   15:     size = seed.length();
   16:     if (size == 16) {
   17:         reg = new int[size];
   18:         int bit = 0;
   19:         int count = 0;
   20:         for (int i = size - 1; i >= 0; i--) {
   21:             bit = (int)seed[count] - ASCII_OFFSET;
   22:             if (bit == 0 || bit == 1) {
   23:                 reg[i] = bit;
   24:                 count++;
   25:             }
   26:             else {
   27:                 reg = nullptr;
   28:                 size = 0;
   29:                 throw std::invalid_argument("Incorrect seed, seed must be"
   30:                 " in binary form");
   31:             }
   32:         }
   33:     }
   34:     else {
   35:         reg = nullptr;
   36:         size = 0;
   37:         throw std::invalid_argument("Incorrect seed size, seed must be 16"
   38:         " bits long");
   39:     }
   40: }
   41:
   42: FibLFSR::FibLFSR(const FibLFSR& copyLFSR) {
   43:     size = copyLFSR.size;
   44:     if (size > 0) {
   45:         reg = new int[size];
   46:         for (int i = 0; i < size; i++) {
   47:             reg[i] = copyLFSR.reg[i];
   48:         }
   49:     }
   50:     else reg = nullptr;
   51: }
   52:
   53: FibLFSR::FibLFSR(FibLFSR&& moveLFSR) noexcept {
   54:     size = moveLFSR.size;
   55:     reg = moveLFSR.reg;
   56:     moveLFSR.reg = nullptr;
   57:     moveLFSR.size = 0;
   58: }
   59:
   60: FibLFSR::~FibLFSR() {
   61:     if (reg != nullptr) delete[] reg;
```

```
    62:     reg = nullptr;
    63:     size = 0;
    64: }
    65:
    66: /** overloaded assignment operators **/
    67: FibLFSR& FibLFSR::operator=(const FibLFSR& rvalue) {
    68:     if (this == &rvalue) return *this;
    69:     if (reg != nullptr) delete[] reg;
    70:
    71:     size = rvalue.size;
    72:     if (size > 0) {
    73:         reg = new int[size];
    74:         for (int i = 0; i < size; i++) {
    75:             reg[i] = rvalue.reg[i];
    76:         }
    77:     }
    78:     return *this;
    79: }
    80:
    81: FibLFSR& FibLFSR::operator=(FibLFSR&& rvalue) noexcept {
    82:     if (this == &rvalue) return *this;
    83:     if (reg != nullptr) delete[] reg;
    84:
    85:     size = rvalue.size;
    86:     reg = rvalue.reg;
    87:     rvalue.reg = nullptr;
    88:     rvalue.size = 0;
    89:
    90:     return *this;
    91: }
    92:
    93: /** step simulates one step of the LFSR by XORing the leftmost bit
    94:  * through the tap bits and returns the result as the rightmost bit **/
    95: int FibLFSR::step() {
    96:     int rightmostBit = reg[size - 1] ^ reg[TAP_BIT1];
    97:     rightmostBit ^= reg[TAP_BIT2];
    98:     rightmostBit ^= reg[TAP_BIT3];
    99:
   100:     // shift each bit to the left except the rightmost bit
   101:     for (int i = size - 1; i >= 1; i--) {
   102:         reg[i] = reg[i - 1];
   103:     }
   104:     reg[0] = rightmostBit;  // assign new rightmost bit
   105:
   106:     return rightmostBit;
   107: }
   108:
   109: /** generate takes k as a parameter and generates k number of steps
   110:  * in the LFSR and returns a k-bit integer **/
   111: int FibLFSR::generate(int k) {
   112:     int result = 0;
   113:     for (int i = k; i > 0; i--) {
   114:         result *= 2;
   115:         result += step();
   116:     }
   117:     return result;
   118: }
   119:
   120: /** overloaded stream insertion operator displays the LFSR bits in order of,
   121:  * from left to right, the highest index bit to the lowest index bit **/
   122: std::ostream& operator<<(std::ostream& out, const FibLFSR& flfsr) {

   123:     for (int i = flfsr.size - 1; i >= 0; i--) {
   124:         out << flfsr.reg[i];
   125:     }
   126:     return out;
   127: }
```

PhotoMagic.cpp

PhotoMagic.cpp Tue Dec 01 14:40:13 2020 1

```
    1: // Copyright 2020 Matt Zagame
    2: /** PhotoMagic.cpp
    3:  * encrypts or decrypts images using a 16-bit linear feedback shift register
    4:  * given an input filename, output filename, and a 16-bit binary seed
    5:  * Matt Zagame 9/19/2020
    6:  */
    7: #include "FibLFSR.h"
    8: #include <SFML/System.hpp>
    9: #include <SFML/Window.hpp>
   10: #include <SFML/Graphics.hpp>
   11:
   12: // transforms image using FibLFSR
   13: void transform(sf::Image& img, FibLFSR* flfsr);
   14:
   15: int main(int argc, char* argv[]) {
   16:     if (argc != 4) {
   17:         std::cerr << "Usage: " << argv[0] <<
   18:             " <input file> <output file> <16 bit binary seed>" << std::endl;
   19:         return -1;
   20:     }
   21:
   22:     FibLFSR flfsr;
   23:     try {
   24:         flfsr = FibLFSR(argv[3]);
   25:     }
   26:     catch (std::invalid_argument err) {
   27:         std::cout << err.what() << std::endl;
   28:         return -1;
   29:     }
   30:
   31:     sf::Image image;
   32:     if (!image.loadFromFile(argv[1])) { return -1; }
   33:
   34:     sf::Vector2u size = image.getSize();
   35:     sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "Original");
   36:     sf::RenderWindow window2(sf::VideoMode(size.x, size.y), "Encrypted/"
   37:     "Decrypted");
   38:
   39:     sf::Texture texture1;
   40:     texture1.loadFromImage(image);
   41:     sf::Sprite sprite1;
   42:     sprite1.setTexture(texture1);
   43:
   44:     transform(image, &flfsr);     // transform the image file
   45:
   46:     sf::Texture texture2;
   47:     texture2.loadFromImage(image);
   48:     sf::Sprite sprite2;
   49:     sprite2.setTexture(texture2);
   50:
   51:     while (window1.isOpen() && window2.isOpen()) {
   52:         sf::Event event;
   53:         while (window1.pollEvent(event)) {
   54:             if (event.type == sf::Event::Closed) { window1.close(); }
   55:         }
   56:         while (window2.pollEvent(event)) {
   57:             if (event.type == sf::Event::Closed) { window2.close(); }
   58:         }
   59:         window1.clear();
   60:         window1.draw(sprite1);
   61:         window1.display();
```

```
   62:         window2.clear();
   63:         window2.draw(sprite2);
   64:         window2.display();
   65:     }
   66:
   67:     if (!image.saveToFile(argv[2])) { return -1; }
   68:
   69:     return 0;
   70: }
   71:
   72: /** transform alters an image's pixel data using a 16-bit LFSR **/
   73: void transform(sf::Image& img, FibLFSR* flfsr) {
   74:     sf::Vector2u size = img.getSize();
   75:     sf::Color p;        // pixel
   76:
   77:     for (int x = 0; x < (int)size.x; x++) {
   78:         for (int y = 0; y < (int)size.y; y++) {
   79:             p = img.getPixel(x, y);
   80:             p.r ^= flfsr->generate(8);
   81:             p.g ^= flfsr->generate(8);
   82:             p.b ^= flfsr->generate(8);
   83:             img.setPixel(x, y, p);
   84:         }
   85:     }
   86: }
```

test.cpp

test.cpp Wed Dec 02 16:29:52 2020 1

```
 1: // Copyright 2020 Matt Zagame
 2: #include <iostream>
 3: #include <string>
 4: #include "FibLFSR.h"
 5:
 6: #define BOOST_TEST_DYN_LINK
 7: #define BOOST_TEST_MODULE Main
 8: #include <boost/test/unit_test.hpp>
 9:
10: BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {
11:
12:   FibLFSR l("1011011000110110");
13:   BOOST_REQUIRE(l.step() == 0);
14:   BOOST_REQUIRE(l.step() == 0);
15:   BOOST_REQUIRE(l.step() == 0);
16:   BOOST_REQUIRE(l.step() == 1);
17:   BOOST_REQUIRE(l.step() == 1);
18:   BOOST_REQUIRE(l.step() == 0);
19:   BOOST_REQUIRE(l.step() == 0);
20:   BOOST_REQUIRE(l.step() == 1);
21:
22:   FibLFSR l2("1011011000110110");
23:   BOOST_REQUIRE(l2.generate(9) == 51);
24: }
25:
26: /** A simple test case to print out the starting and resulting bit
27:  * patterns and check to make sure the right result is printed **/
28: BOOST_AUTO_TEST_CASE(my_test1) {
29:   std::cout << "\n************ My Test Case 1 ************" << std::endl;
30:   FibLFSR lfsr("1011011000110110");
31:   std::cout << "    Original seed: " << lfsr << std::endl;
32:
33:   int result = lfsr.generate(5);
34:   BOOST_REQUIRE(result == 3);
35:
36:   std::cout << "After generate(5): " << lfsr << " " << result << std::endl;
37:   std::cout << std::endl;
38: }
39:
40: /** A test case to make sure the invalid_argument exception is thrown
41:  * when given a seed either too long or too short */
42: BOOST_AUTO_TEST_CASE(my_test2) {
43:   std::cout << "\n************ My Test Case 2 ************" << std::endl;
44:   std::string testSeed1 = "0101100101";
45:   std::string testSeed2 = "01110110001101101011001O1";
46:
47:   std::cout << "Testing exception thrown when seed is too short: 0101100101"
48:   << std::endl;
49:   BOOST_REQUIRE_THROW(FibLFSR("0101100101"), std::invalid_argument);
50:
51:   std::cout << "Testing exception thrown when seed is too long : "
52:   "01110110001101101011OO" << std::endl;
53:   BOOST_REQUIRE_THROW(FibLFSR("01110110001101101011OO"),
54:   std::invalid_argument);
55: }
```

PS2 N-Body Simulation

The goal of this assignment was to simulate the effect of pairwise forces on particles in a universe. This assignment had two parts, PS2a and PS2b. The first half of the assignment involved creating a display of our galaxy using two classes, one to represent each planet and one to manage each of the planets. The objective of the first part was just to make sure planets could be read from a text file and that planets were displayed correctly in the SFML window. The second half of the assignment involved using the proper physics calculations to simulate the effects of pairwise force. A main function was required to take input in the form of total time of the simulation, the amount of time in one step of the simulation, and the input file with each planet's information.

What I Did:
PS2a - NBody displays celestial bodies in a universe using the information found in the provided text file. This file contains the number of planets to display and the radius of the universe on the first two lines. Each subsequent line contains a planet's x position, y position, x and y velocities, mass, and image filename. I overloaded the insertion operator in the Universe class so that the program could use input redirection from the provided text file to store the planets in the universe. I then wrote the main function which would create the universe given the text file as a command line argument.

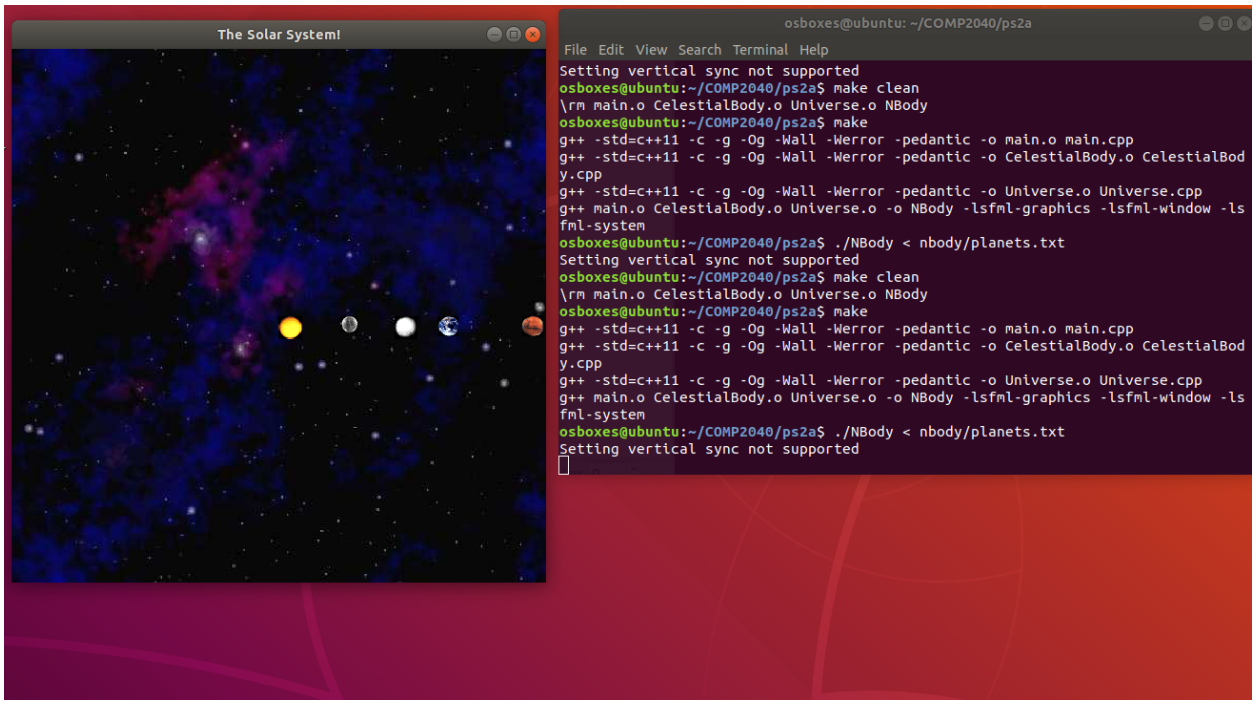
PS2b - NBody simulates the effect of gravitational forces on several CelestialBody objects in the Universe. The user can run NBody using any of the universe text files found in the provided directory. For this portion of the assignment I implemented the step() function in Universe which calculates pairwise forces and increments the simulation by a given amount of time in seconds. The main function of NBody is responsible for displaying the particles in the universe and managing the time loop.

Key Algorithms, Data Structures, and OOP Designs:
Overloading the insertion operator in the Universe class allows for input redirection from a file to set up the planets for simulation. Both CelestialBody and Universe are sf::Drawable so that they may be drawn to the window. Each CelestialBody is instantiated using a std::unique_ptr and stored in a std::vector within the Universe class. The planets are then drawn in a loop in main using the draw function on each CelestialBody in the vector. The step() function in Universe is where the bulk of the calculations are performed for this program.

What I Learned:

I learned quite a bit in this assignment. For starters, this was the first C++ program where I used smart pointers to initialize objects. I also was able to output the final position information of the simulation to a text file at the end of the main function using fstream. Outside of C++, I also learned a lot about the physics involved in the N-Body simulation using the leapfrog finite difference approximation scheme.

PS2a Demo



Makefile

```
CC = g++
CFLAGS = -std=c++14 -c -g -Og -Wall -Werror -pedantic
OBJS = main.o CelestialBody.o Universe.o
LIBS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
EXE = NBody

all: $(EXE)

NBody: $(OBJS)
    $(CC) $(OBJS) -o $(EXE) $(LIBS)

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

clean:
    \rm $(OBJS) $(EXE)
```

CelestialBody.h

CelestialBody.h Tue Dec 01 19:21:04 2020 1

```
 1: // Copyright 2020 Matt Zagame
 2: #ifndef CELESTIALBODY_H
 3: #define CELESTIALBODY_H
 4: #include <iostream>
 5: #include <exception>
 6: #include <SFML/Graphics.hpp>
 7:
 8: class CelestialBody : public sf::Drawable {
 9: public:
10:     // constructors
11:     CelestialBody();
12:     CelestialBody(double x_position, double y_position, double x_velocity,
13:     double y_velocity, double init_mass, std::string image_filename);
14:
15:     // accessors and mutators
16:     double getXPos() const;
17:     double getYPos() const;
18:     double getXVel() const;
19:     double getYVel() const;
20:     double getMass() const;
21:     sf::Image getImage() const;
22:     std::string getImageFilename() const;
23:
24:     void setXPos(double x_position);
25:     void setYPos(double y_position);
26:     void setXVel(double x_velocity);
27:     void setYVel(double y_velocity);
28:     void setMass(double _mass);
29:     void setImage(std::string image_filename);
30:
31:     // update x and y position of planet given the scale of the universe
32:     void updatePosition(double scale, sf::Vector2u windowSize);
33:
34: private:
35:     // override virtual draw function
36:     void draw(sf::RenderTarget& target, sf::RenderStates states) const
37:     override;
38:
39:     double xPos, yPos;      // x and y position
40:     double xVel, yVel;      // x and y velocity
41:     double mass;
42:     std::string imageFilename;
43:     sf::Image image;
44:     sf::Texture texture;
45:     sf::Sprite sprite;
46: };
47: #endif
```

CelestialBody.cpp

CelestialBody.cpp Tue Dec 01 19:19:00 2020 1

```
 1: // Copyright 2020 Matt Zagame
 2: #include "CelestialBody.h"
 3:
 4: // constructors
 5: CelestialBody::CelestialBody() : xPos(0), yPos(0), xVel(0), yVel(0),
 6: mass(0) {}
 7:
 8: CelestialBody::CelestialBody(double x_position, double y_position, double
 9: x_velocity, double y_velocity, double init_mass,
10: std::string image_filename) {
11:     if (!image.loadFromFile(image_filename)) {
12:         throw std::invalid_argument("Celestial Body image file not found,"
13:         " exiting");
14:     }
15:     imageFilename = image_filename;
16:     texture.loadFromImage(image);
17:     sprite.setTexture(texture);
18:     xPos = x_position;   yPos = y_position;
19:     xVel = x_velocity;   yVel = y_velocity;
20:     mass = init_mass;
21: }
22:
23: // accessors and mutators
24: double CelestialBody::getXPos() const { return xPos; }
25: double CelestialBody::getYPos() const { return yPos; }
26: double CelestialBody::getXVel() const { return xVel; }
27: double CelestialBody::getYVel() const { return yVel; }
28: double CelestialBody::getMass() const { return mass; }
29: sf::Image CelestialBody::getImage() const { return image; }
30: std::string CelestialBody::getImageFilename() const {
31:     return imageFilename;
32: }
33:
34: void CelestialBody::setXPos(double x_position) { xPos = x_position; }
35: void CelestialBody::setYPos(double y_position) { yPos = y_position; }
36: void CelestialBody::setXVel(double x_velocity) { xVel = x_velocity; }
37: void CelestialBody::setYVel(double y_velocity) { yVel = y_velocity; }
38: void CelestialBody::setMass(double _mass) { mass = _mass; }
39:
40: void CelestialBody::setImage(std::string image_filename) {
41:     if(!image.loadFromFile("nbody/" + image_filename)) {
42:         throw std::invalid_argument("Celestial Body image file not found");
43:     }
44:     imageFilename = image_filename;
45:     texture.loadFromImage(image);
46:     sprite.setTexture(texture);
47: }
48:
49: // update x and y position of planet given the scale of the universe
50: void CelestialBody::updatePosition(double scale, sf::Vector2u windowSize) {
51:     int x, y;
52:     x = ((xPos / 2) / scale * windowSize.x) + (windowSize.x / 2);
53:     y = ((yPos / 2) / scale * windowSize.y) + (windowSize.y / 2);
54:     sprite.setPosition(x, y);
55: }
56:
57: // override virtual draw function
58: void CelestialBody::draw(sf::RenderTarget& target, sf::RenderStates states)
59: const {
60:     target.draw(sprite, states);
61: }
```

Universe.h

Universe.h Tue Dec 01 19:30:20 2020 1

```
 1: // Copyright 2020 Matt Zagame
 2: #ifndef UNIVERSE_H
 3: #define UNIVERSE_H
 4: #include "CelestialBody.h"
 5: #include <memory>
 6: #include <vector>
 7: #include <cmath>
 8: #define GRAVITY_CONST 6.67e-11
 9:
10: class Universe : public sf::Drawable {
11: public:
12:     // constructors
13:     Universe();
14:     Universe(std::string image_filename);
15:     explicit Universe(const Universe& copyUniverse);
16:     explicit Universe(Universe&& moveUniverse) noexcept;
17:     ~Universe();
18:
19:     // accessors and mutators
20:     int getNumOfPlanets() const;
21:     double getRadius() const;
22:     sf::Image getImage() const;
23:
24:     void setNumOfPlanets(int n);
25:     void setRadius(double r);
26:     void setImage(std::string image_filename);
27:
28:     /* step moves forward the simulation a single interval of time in
29:     seconds */
30:     void step(double seconds);
31:
32:     // vector of pointers to celestial bodies
33:     std::vector<std::unique_ptr<CelestialBody>> planets;
34:
35:     // overloaded assignment operators for Universe setup in main
36:     Universe& operator=(Universe& rvalue);
37:     Universe& operator=(Universe&& rvalue) noexcept;
38:
39:     // overloaded insertion operator for file input through terminal
40:     friend std::istream& operator>>(std::istream& in, Universe& universe);
41:
42: private:
43:     // override virtual draw function
44:     void draw(sf::RenderTarget& target, sf::RenderStates states) const
45:     override;
46:
47:     int numOfPlanets;
48:     double radius;              // radius of the universe
49:     sf::Image image;            // universe background
50:     sf::Texture texture;
51:     sf::Sprite sprite;
52: };
53: #endif
```

Universe.cpp

Universe.cpp Tue Dec 01 19:29:08 2020 1

```
    1: // Copyright 2020 Matt Zagame
    2: #include "Universe.h"
    3:
    4: // constructors
    5: Universe::Universe() : numOfPlanets(0), radius(0) {}
    6:
    7: Universe::Universe(std::string image_filename) {
    8:     if(!image.loadFromFile(image_filename)) {
    9:         throw std::invalid_argument("Universe image file not found,"
   10:         " exiting");
   11:     }
   12:     texture.loadFromImage(image);
   13:     sprite.setTexture(texture);
   14:     numOfPlanets = 0;
   15:     radius = 0;
   16: }
   17:
   18: Universe::Universe(const Universe& copyUniverse) {
   19:     numOfPlanets = copyUniverse.numOfPlanets;
   20:     radius = copyUniverse.radius;
   21:     image = copyUniverse.image;
   22:     texture.loadFromImage(image);
   23:     sprite.setTexture(texture);
   24: }
   25:
   26: Universe::Universe(Universe&& moveUniverse) noexcept {
   27:     numOfPlanets = moveUniverse.numOfPlanets;
   28:     radius = moveUniverse.radius;
   29:     image = moveUniverse.image;
   30:     texture.loadFromImage(image);
   31:     sprite.setTexture(texture);
   32:     moveUniverse.numOfPlanets = 0;
   33:     moveUniverse.radius = 0;
   34: }
   35:
   36: Universe::~Universe() {
   37:     numOfPlanets = 0;
   38:     radius = 0;
   39: }
   40:
   41: // accessors and mutators
   42: int Universe::getNumOfPlanets() const { return numOfPlanets; }
   43: double Universe::getRadius() const { return radius; }
   44: sf::Image Universe::getImage() const { return image; }
   45:
   46: void Universe::setNumOfPlanets(int n) { numOfPlanets = n; }
   47: void Universe::setRadius(double r) { radius = r; }
   48:
   49: void Universe::setImage(std::string image_filename) {
   50:     if(!image.loadFromFile(image_filename)) {
   51:         throw std::invalid_argument("Universe image file not found");
   52:     }
   53:     texture.loadFromImage(image);
   54:     sprite.setTexture(texture);
   55: }
   56:
   57: // step moves forward the simulation a single interval of time in seconds
   58: void Universe::step(double seconds) {
   59:     double d, deltaX, deltaY;       // used to calculate distance d
   60:     double force, forceX, forceY;   // used to calculate force on x/y
   61:     double accelX, accelY;          // acceleration on x/y
```

Universe.cpp Tue Dec 01 19:29:08 2020 2

```
   62:
   63:     for (int i = 0; i < numOfPlanets; i++) {
   64:         forceX = 0; forceY = 0;
   65:
   66:         // calculate the net force on x and y
   67:         for (int j = 0; j < numOfPlanets; j++) {
   68:             if (i != j) {
   69:                 deltaX = planets[j]->getXPos() - planets[i]->getXPos();
   70:                 deltaY = planets[j]->getYPos() - planets[i]->getYPos();
   71:                 d = sqrt(deltaX * deltaX + deltaY * deltaY);
   72:
   73:                 /* pairwise force is the product of the gravitational
   74:                 constant and two planets' mass divided by the square of the
   75:                 distance between them */
   76:                 force = (GRAVITY_CONST * planets[i]->getMass() *
   77:                 planets[j]->getMass()) / (d * d);
   78:
   79:                 forceX += force * (deltaX / d);
   80:                 forceY += force * (deltaY / d);
   81:             }
   82:         }
   83:
   84:         // calculate the acceleration
   85:         accelX = forceX / planets[i]->getMass();
   86:         accelY = forceY / planets[i]->getMass();
   87:
   88:         // calculate the velocity
   89:         planets[i]->setXVel(planets[i]->getXVel() + seconds * accelX);
   90:         planets[i]->setYVel(planets[i]->getYVel() + seconds * accelY);
   91:     }
   92:
   93:     // update positions
   94:     for (int i = 0; i < numOfPlanets; i++) {
   95:         planets[i]->setXPos(planets[i]->getXPos() + seconds *
   96:         planets[i]->getXVel());
   97:         planets[i]->setYPos(planets[i]->getYPos() + seconds *
   98:         planets[i]->getYVel());
   99:         planets[i]->updatePosition(radius, image.getSize());
  100:     }
  101: }
  102:
  103: // overloaded assingment operators
  104: Universe& Universe::operator=(Universe& rvalue) {
  105:     if (this == &rvalue) { return *this; }
  106:
  107:     numOfPlanets = rvalue.numOfPlanets;
  108:     radius = rvalue.radius;
  109:     image = rvalue.image;
  110:     texture.loadFromImage(image);
  111:     sprite.setTexture(texture);
  112:
  113:     return *this;
  114: }
  115:
  116: Universe& Universe::operator=(Universe&& rvalue) noexcept {
  117:     if (this == &rvalue) { return *this; }
  118:
  119:     numOfPlanets = rvalue.numOfPlanets;
  120:     radius = rvalue.radius;
  121:     image = rvalue.image;
  122:     texture.loadFromImage(image);
```

Universe.cpp Tue Dec 01 19:29:08 2020 3

```
  123:     sprite.setTexture(texture);
  124:     rvalue.numOfPlanets = 0;
  125:     rvalue.radius = 0;
  126:
  127:     return *this;
  128: }
  129:
  130: // overloaded insertion operator for file input through terminal
  131: std::istream& operator>>(std::istream& in, Universe& universe) {
  132:     double dval;
  133:     char buffer[30];    // only take filenames less than 30 characters long
  134:     std::string image;
  135:
  136:     in >> universe.numOfPlanets;
  137:     if (universe.numOfPlanets < 0) { universe.numOfPlanets = 0; }
  138:     else if (universe.numOfPlanets > 1000) {
  139:         universe.numOfPlanets = 1000;
  140:     }
  141:     in >> universe.radius;
  142:
  143:     // get data for each planet
  144:     for (int i = 0; i < universe.numOfPlanets; i++) {
  145:         std::unique_ptr<CelestialBody> planet(
  146:             std::make_unique<CelestialBody>(CelestialBody()));
  147:
  148:         // get x and y positions, velocity, mass and image file
  149:         in >> dval;
  150:         planet->setXPos(dval);
  151:         in >> dval;
  152:         planet->setYPos(dval);
  153:         planet->updatePosition(universe.radius, universe.image.getSize());
  154:         in >> dval;
  155:         planet->setXVel(dval);
  156:         in >> dval;
  157:         planet->setYVel(dval);
  158:         in >> dval;
  159:         planet->setMass(dval);
  160:         in >> buffer;
  161:         image = buffer;
  162:
  163:         try {
  164:             planet->setImage(image);
  165:         }
  166:         catch (const std::invalid_argument& err) {
  167:             std::cout << err.what() << std::endl;
  168:             exit(-1);
  169:         }
  170:         universe.planets.push_back(std::move(planet));
  171:     }
  172:
  173:     return in;
  174: }
  175:
  176: // override virtual draw function
  177: void Universe::draw(sf::RenderTarget& target, sf::RenderStates states)
  178: const {
  179:     target.draw(sprite, states);
  180: }
```

main.cpp

main.cpp Tue Dec 01 19:25:24 2020 1

```
 1: // Copyright 2020 Matt Zagame
 2: #include "Universe.h"
 3: #include <fstream>
 4: #include <SFML/Audio.hpp>
 5:
 6: int main(int argc, char* argv[]) {
 7:     if (argc != 3) {
 8:         std::cerr << "usage: ./NBody T Î\224t < input_file" << std::endl;
 9:         exit(-1);
10:     }
11:     double time = strtod(argv[1], nullptr);
12:     double deltaT = strtod(argv[2], nullptr);
13:     int elapsed_time = 0;
14:
15:     Universe universe;
16:     try {
17:         universe = Universe("nbody/starfield.jpg");
18:         std::cin >> universe;       // get universe data from file input
19:     }
20:     catch (const std::invalid_argument& err) {
21:         std::cout << err.what() << std::endl;
22:         exit(-1);
23:     }
24:     sf::Vector2u windowSize = universe.getImage().getSize();
25:     sf::RenderWindow window(sf::VideoMode(windowSize.x, windowSize.y),
26:     "The Solar System!");
27:     window.setFramerateLimit(60);
28:
29:     sf::Music music;
30:     if (!music.openFromFile("nbody/2001.wav")) {
31:         std::cout << "Cannot load music nbody/2001.wav" << std::endl;
32:     }
33:     music.play();
34:     music.setLoop(true);
35:
36:     sf::Text timer;
37:     sf::Font font;
38:     if (!font.loadFromFile("nbody/verdana.ttf")) {
39:         std::cout << "Cannot load font nbody/verdana.ttf" << std::endl;
40:     }
41:     timer.setFont(font);
42:     timer.setFillColor(sf::Color::White);
43:     timer.setCharacterSize(16);
44:
45:     // SFML display loop
46:     while (window.isOpen()) {
47:         sf::Event event;
48:         while (window.pollEvent(event)) {
49:             if (event.type == sf::Event::Closed) {
50:                 window.close();
51:             }
52:         }
53:         timer.setString("Elapsed time: " + std::to_string(elapsed_time) +
54:         "seconds");
55:         window.clear();
56:         window.draw(universe);
57:         for (int i = 0; i < universe.getNumOfPlanets(); i++) {
58:             window.draw(*universe.planets[i]);
59:         }
60:         window.draw(timer);
61:         if (elapsed_time < time) {
```

```
   62:             universe.step(deltaT);      // call to step function
   63:             elapsed_time += deltaT;
   64:         }
   65:         else music.stop();
   66:         window.display();
   67:     }
   68:
   69:     // print output to file
   70:     std::ofstream outfile;
   71:     outfile.open("output.txt");
   72:     outfile << universe.getNumOfPlanets() << std::endl;
   73:     outfile << std::scientific << universe.getRadius() << std::endl;
   74:     for (int i = 0; i < universe.getNumOfPlanets(); i++) {
   75:         outfile << std::scientific << universe.planets[i]->getXPos()
   76:         << "  " << std::scientific << universe.planets[i]->getYPos()
   77:         << "  " << std::scientific << universe.planets[i]->getXVel()
   78:         << "  " << std::scientific << universe.planets[i]->getYVel()
   79:         << "  " << std::scientific << universe.planets[i]->getMass()
   80:         << "  " << universe.planets[i]->getImageFilename() << std::endl;
   81:     }
   82:     outfile.close();
   83:
   84:     return 0;
   85: }
```

PS3 Karplus-Strong Guitar Simulation

This assignment focused on simulating a plucked guitar string sound using the Karplus-Strong algorithm. The assignment was split into two parts, PS3a and PS3b. In PS3a, we were to implement the class CircularBuffer which would be used to maintain a ring buffer feedback mechanism. PS3b would then implement the class StringSound which would initialize a CircularBuffer with N samples based on a sample rate of 44,100Hz and a given frequency. These frequencies would be associated with key presses in the main function and output using SFML audio.

What I Did:
PS3a - CircularBuffer emulates a ring buffer feedback mechanism using an array of type int16_t which stores N samples of a vibration. The class implements the array as a queue, with functions such as enqueue and dequeue, to work with the Karplus-Strong algorithm. The CircularBuffer constructor and its queue functions include exception handling which I tested using Boost libraries in test.cpp.

PS3b - StringSound takes a frequency and creates a CircularBuffer of length equal to sample rate / frequency. The function pluck() which fills the buffer of a string sound with random values in the Int16 range to simulate noise. The function tic() is responsible for advancing the sound's buffer to its next state, and the sample() function returns the first sample from the buffer. The main function first creates a vector of Int16 vectors of each string sounds' samples, which is then loaded into a vector of sf::SoundBuffers, which are lastly loaded into a vector of sf::Sounds. Inside the SFML display loop, a sound will be played based on the corresponding key pressed.

Key Algorithms, Data Structures, and OOP Designs:
StringSound uses CircularBuffer as a dynamic array and thus each class had dynamic memory allocation and required copy/move/destructor functions. Exceptions were used in both classes for several functions. For example, the StringSound(frequency) constructor has one exception that prevents construction with a frequency of 0. StringSound handles exceptions thrown by CircularBuffer when dynamically allocating memory for the buffer during construction and when using the CircularBuffer functions in its own functions.

What I Learned:
I learned a bunch about how digital audio is streamed by using the Karplus-Strong algorithm and how to work with audio using SFML. I also learned about the mersenne twister random number generator which I used to implement the pluck()

function. This was also the first assignment that I wrote that would conform to Google's style guidelines for C++.

Makefile

```
CC = g++
CFLAGS = -std=c++11 -c -g -Og -Wall -Werror -pedantic
LIBS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
OBJS = KSGuitarSim.o StringSound.o CircularBuffer.o
EXE = KSGuitarSim

all : $(EXE)

$(EXE) : $(OBJS)
    $(CC) $(OBJS) -o $(EXE) $(LIBS)

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

clean :
    \rm $(OBJS) $(EXE)
```

CircularBuffer.h

CircularBuffer.h Tue Dec 01 19:35:18 2020 1

```
 1: // Copyright 2020 Matt Zagame
 2: #ifndef CIRCULARBUFFER_H   //NOLINT
 3: #define CIRCULARBUFFER_H   //NOLINT
 4:
 5: #include <stdint.h>
 6: #include <iostream>
 7: #include <exception>
 8:
 9: class CircularBuffer {
10:  public:
11:     explicit CircularBuffer(int capacity);   // create an empty ring buffer
12:                                              // with given max capacity
13:     CircularBuffer(const CircularBuffer& copyCB);
14:     CircularBuffer(CircularBuffer&& moveCB) noexcept;
15:     ~CircularBuffer();
16:
17:     int size() const;      // return number of items currently in the buffer
18:     int capacity() const;     // return capacity of buffer
19:     bool isEmpty() const;     // is the buffer empty (size equals zero)?
20:     bool isFull() const;      // is the buffer full (size equals capacity)?
21:     void enqueue(int16_t x);  // add item x to the end
22:     int16_t dequeue();        // delete and return item from the front
23:     int16_t peek();        // return (but do not delete) item from the front
24:     void empty();             // set the buffer to an empty state
25:
26:     CircularBuffer& operator=(const CircularBuffer& rightSide);
27:     CircularBuffer& operator=(CircularBuffer&& rvalue) noexcept;
28:
29:  private:
30:     int _size;
31:     int _capacity;
32:     int _first;
33:     int _last;
34:     int16_t* _buffer;
35: };
36: #endif  //NOLINT
```

CircularBuffer.cpp

```
CircularBuffer.cpp       Tue Dec 01 19:33:21 2020       1
    1: // Copyright 2020 Matt Zagame
    2: #include "CircularBuffer.h"
    3:
    4: /** constructors **/
    5: CircularBuffer::CircularBuffer(int capacity) {
    6:     if (capacity < 1) {
    7:         throw std::invalid_argument("CircularBuffer constructor: capacity"
    8:         " must be greater than zero.");
    9:     }
   10:     _capacity = capacity;
   11:     _size = 0;
   12:     _first = 0;
   13:     _last = 0;
   14:     _buffer = new int16_t[capacity];
   15: }
   16:
   17: CircularBuffer::CircularBuffer(const CircularBuffer& copyCB) {
   18:     _capacity = copyCB._capacity;
   19:     if (_capacity > 0) {
   20:         _size = copyCB._size;
   21:         _first = copyCB._first;
   22:         _last = copyCB._last;
   23:         _buffer = new int16_t[_capacity];
   24:         for (int i = 0; i < _size; i++) {
   25:             _buffer[i] = copyCB._buffer[i];
   26:         }
   27:     } else {
   28:         _size = 0;
   29:         _first = 0;
   30:         _last = 0;
   31:         _buffer = nullptr;
   32:     }
   33: }
   34:
   35: CircularBuffer::CircularBuffer(CircularBuffer&& moveCB) noexcept {
   36:     _capacity = moveCB._capacity;
   37:     _size = moveCB._size;
   38:     _first = moveCB._first;
   39:     _last = moveCB._last;
   40:     _buffer = moveCB._buffer;
   41:     moveCB._capacity = 0;
   42:     moveCB._size = 0;
   43:     moveCB._first = 0;
   44:     moveCB._last = 0;
   45:     moveCB._buffer = nullptr;
   46: }
   47:
   48: CircularBuffer::~CircularBuffer() {
   49:     if (_buffer != nullptr) { delete[] _buffer; }
   50:     _capacity = 0;
   51:     _size = 0;
   52:     _first = 0;
   53:     _last = 0;
   54:     _buffer = nullptr;
   55: }
   56:
   57: /** member functions **/
   58: int CircularBuffer::size() const { return _size; }
   59: int CircularBuffer::capacity() const { return _capacity; }
   60: bool CircularBuffer::isEmpty() const { return _size == 0; }
   61: bool CircularBuffer::isFull() const { return _size == _capacity; }
```

```
   62:
   63: void CircularBuffer::enqueue(int16_t x) {
   64:     if (isFull()) {
   65:         throw std::runtime_error("enqueue: can't enqueue to a full ring.");
   66:     }
   67:     // add element to end of buffer and increment last
   68:     _buffer[_last] = x;
   69:     _last++;
   70:     if (_last == _capacity) { _last = 0; }
   71:     _size++;
   72: }
   73:
   74: int16_t CircularBuffer::dequeue() {
   75:     if (isEmpty()) {
   76:         throw std::runtime_error("dequeue: can't dequeue from an empty"
   77:         " ring.");
   78:     }
   79:     /* return first element of buffer and increment index of new first
   80:     element */
   81:     int16_t result = _buffer[_first];
   82:     _first++;
   83:     if (_first == _capacity) { _first = 0; }
   84:     _size--;
   85:     return result;
   86: }
   87:
   88: int16_t CircularBuffer::peek() {
   89:     if (isEmpty()) {
   90:         throw std::runtime_error("peek: can't peek from an empty ring.");
   91:     }
   92:     return _buffer[_first];  // return first element
   93: }
   94:
   95: void CircularBuffer::empty() {
   96:     _size = 0;
   97:     _first = 0;
   98:     _last = 0;
   99: }
  100:
  101: /** overloaded assignment operators **/
  102: CircularBuffer& CircularBuffer::operator=(const CircularBuffer& rightSide)
  103: {
  104:     if (this != &rightSide) {
  105:         if (_buffer != nullptr) { delete[] _buffer; }
  106:         _capacity = rightSide._capacity;
  107:         if (_capacity > 0) {
  108:             _size = rightSide._size;
  109:             _first = rightSide._first;
  110:             _last = rightSide._last;
  111:             _buffer = new int16_t[_capacity];
  112:             for (int i = 0; i < _size; i++) {
  113:                 _buffer[i] = rightSide._buffer[i];
  114:             }
  115:         } else {
  116:             _size = 0;
  117:             _first = 0;
  118:             _last = 0;
  119:             _buffer = nullptr;
  120:         }
  121:     }
  122:     return *this;
```

CircularBuffer.cpp Tue Dec 01 19:33:21 2020 3

```
123: }
124:
125: CircularBuffer& CircularBuffer::operator=(CircularBuffer&& rvalue) noexcept
126: {
127:     if (this != &rvalue) {
128:         if (_buffer != nullptr) { delete[] _buffer; }
129:         _capacity = rvalue._capacity;
130:         _size = rvalue._size;
131:         _first = rvalue._first;
132:         _last = rvalue._last;
133:         _buffer = rvalue._buffer;
134:         rvalue._capacity = 0;
135:         rvalue._size = 0;
136:         rvalue._first = 0;
137:         rvalue._last = 0;
138:         rvalue._buffer = nullptr;
139:     }
140:     return *this;
141: }
```

StringSound.h

StringSound.h Tue Dec 01 19:38:32 2020 1

```
 1: // Copyright 2020 Matt Zagame
 2: #ifndef STRINGSOUND_H   //NOLINT
 3: #define STRINGSOUND_H   //NOLINT
 4:
 5: #include <math.h>
 6: #include <vector>
 7: #include <string>
 8: #include <random>
 9: #include <SFML/Audio/SoundBuffer.hpp>
10: #include "CircularBuffer.h"
11:
12: #define SAMPLE_RATE 44100  // Hz
13: #define DECAY_FACTOR 0.996
14:
15: class StringSound {
16:  public:
17:     // default constructor
18:     StringSound();
19:
20:     /* create a guitar string sound of the given frequency using a sampling
21:     rate of 44,100 */
22:     explicit StringSound(double frequency);
23:
24:     /* create a guitar string with size and initial values given by the
25:     vector */
26:     explicit StringSound(std::vector<sf::Int16> init);
27:
28:     StringSound(const StringSound& copySS);
29:     StringSound(StringSound&& moveSS) noexcept;
30:     ~StringSound();
31:
32:     /* pluck the guitar string by replacing the buffer with random values,
33:     representing white noise */
34:     void pluck();
35:     void tic();            // advance the simulation one step
36:     sf::Int16 sample();    // return the current sample
37:     int time();            // return the number of times tic has been called
38:
39:     StringSound& operator=(const StringSound& rightSide);
40:     StringSound& operator=(StringSound&& rvalue) noexcept;
41:
42:  private:
43:     CircularBuffer* _rb;   // ring buffer
44:     int _time;
45: };
46: #endif  //NOLINT
```

StringSound.cpp

```
StringSound.cpp        Tue Dec 01 19:37:36 2020       1
    1: // Copyright 2020 Matt Zagame
    2: #include "StringSound.h"
    3: #include <cstdlib>
    4: #include <ctime>
    5:
    6: /** constructors **/
    7: StringSound::StringSound() : _rb(nullptr), _time(0) {}
    8:
    9: StringSound::StringSound(double frequency) {
   10:     if (frequency == 0) {
   11:         throw std::invalid_argument("StringSound constructor: frequency"
   12:         " cannot be zero");
   13:     }
   14:     try {
   15:         _rb = new CircularBuffer(ceil(SAMPLE_RATE / frequency));
   16:     }
   17:     catch (std::invalid_argument err) {
   18:         std::cerr << err.what() << std::endl;
   19:     }
   20:     _time = 0;
   21: }
   22:
   23: StringSound::StringSound(std::vector<sf::Int16> init) {
   24:     try {
   25:         _rb = new CircularBuffer(init.size());
   26:         for (int i = 0; i < static_cast<int>(init.size()); i++) {
   27:             _rb->enqueue(init[i]);
   28:         }
   29:     }
   30:     catch (std::invalid_argument err) {
   31:         std::cerr << err.what() << std::endl;
   32:     }
   33:     catch (std::runtime_error err) {
   34:         std::cerr << err.what() << std::endl;
   35:     }
   36:     _time = 0;
   37: }
   38:
   39: StringSound::StringSound(const StringSound& copySS) {
   40:     _rb = copySS._rb;
   41:     _time = copySS._time;
   42: }
   43:
   44: StringSound::StringSound(StringSound&& moveSS) noexcept {
   45:     _rb = moveSS._rb;
   46:     _time = moveSS._time;
   47:     moveSS._rb = nullptr;
   48:     moveSS._time = 0;
   49: }
   50:
   51: StringSound::~StringSound() {
   52:     if (_rb != nullptr) { delete _rb; }
   53:     _rb = nullptr;
   54:     _time = 0;
   55: }
   56:
   57: void StringSound::pluck() {
   58:     std::random_device device;
   59:     std::mt19937 mt_rand(device());     // random number generator
   60:     std::uniform_int_distribution<int16_t> distribution(INT16_MIN,
   61:     INT16_MAX);
```

```
 62:
 63:     _rb->empty();   // reset the ring buffer
 64:     _time = 0;
 65:
 66:     // fill ring buffer with random values
 67:     try {
 68:         for (int i = 0; i < _rb->capacity(); i++) {
 69:             _rb->enqueue(distribution(mt_rand));
 70:         }
 71:     }
 72:     catch (std::runtime_error err) {
 73:         std::cerr << err.what() << std::endl;
 74:     }
 75: }
 76:
 77: void StringSound::tic() {
 78:     try {
 79:         int16_t next = DECAY_FACTOR * (_rb->dequeue() + _rb->peek()) / 2;
 80:         _rb->enqueue(next);
 81:     }
 82:     catch (std::runtime_error err) {
 83:         std::cerr << err.what() << std::endl;
 84:     }
 85:     _time++;
 86: }
 87:
 88: sf::Int16 StringSound::sample() {
 89:     sf::Int16 result = 0;
 90:     try {
 91:         result = _rb->peek();
 92:     }
 93:     catch (std::runtime_error err) {
 94:         std::cerr << err.what() << std::endl;
 95:     }
 96:     return result;
 97: }
 98:
 99: int StringSound::time() { return _time; }
100:
101: StringSound& StringSound::operator=(const StringSound& rightSide) {
102:     if (this != &rightSide) {
103:         if (_rb != nullptr) { delete _rb; }
104:         _rb = rightSide._rb;
105:         _time = rightSide._time;
106:     }
107:     return *this;
108: }
109:
110: StringSound& StringSound::operator=(StringSound&& rvalue) noexcept {
111:     if (this != &rvalue) {
112:         if (_rb != nullptr) { delete _rb; }
113:         _rb = rvalue._rb;
114:         _time = rvalue._time;
115:         rvalue._rb = nullptr;
116:         rvalue._time = 0;
117:     }
118:     return *this;
119: }
```

KSGuitarSim.cpp

KSGuitarSim.cpp Tue Dec 01 19:36:03 2020 1

```
 1: // Copyright 2020 Matt Zagame
 2: #include <SFML/Graphics.hpp>
 3: #include <SFML/System.hpp>
 4: #include <SFML/Audio.hpp>
 5: #include <SFML/Window.hpp>
 6: #include "StringSound.h"
 7:
 8: /* takes a string sound and returns a vector sample stream of 16 bit ints */
 9: std::vector<sf::Int16> makeSamples(StringSound* gs);
10:
11: /* initializes a vector of sound buffers from a vector of samples */
12: void setupSoundBuffers(const std::vector<std::vector<sf::Int16>>& samples,
13:  std::vector<sf::SoundBuffer>* soundBuffers);
14:
15: /* initializes a vector of sounds from a vector of sound buffers */
16: void setupSounds(const std::vector<sf::SoundBuffer>& soundBuffers,
17:  std::vector<sf::Sound>* sounds);
18:
19: int main(int argc, char* argv[]) {
20:     // SFML window setup vars
21:     sf::Image image;
22:     sf::Texture texture;
23:     sf::Sprite sprite;
24:
25:     if (!image.loadFromFile("piano.png")) {
26:         throw std::runtime_error("sf::Image: could not load piano.png");
27:     }
28:     texture.loadFromImage(image);
29:     sprite.setTexture(texture);
30:
31:     sf::RenderWindow window(sf::VideoMode(image.getSize().x,
32:     image.getSize().y), "KS Guitar Sim");
33:
34:     // guitar sim vars
35:     // keys used to play guitar string sounds
36:     std::string keys = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
37:     std::vector<std::vector<sf::Int16>> samples;
38:     std::vector<sf::SoundBuffer> soundBuffers;
39:     std::vector<sf::Sound> sounds;
40:     StringSound gs;
41:     double freq;    // frequency for string sounds
42:
43:     // create samples from string sounds for each key
44:     for (int i = 0; i < static_cast<int>(keys.length()); i++) {
45:         freq = 440 * pow(2, (static_cast<double>(i) - 24) / 12);
46:         try {
47:             gs = StringSound(freq);
48:         }
49:         catch (std::invalid_argument err) {
50:             std::cerr << err.what() << std::endl;
51:         }
52:         samples.push_back(makeSamples(&gs));
53:     }
54:
55:     setupSoundBuffers(samples, &soundBuffers);
56:     setupSounds(soundBuffers, &sounds);
57:
58:     while (window.isOpen()) {
59:         sf::Event event;
60:         while (window.pollEvent(event)) {
61:             if (event.type == sf::Event::Closed) {
```

```
   62:                 window.close();
   63:             }
   64:             // check key pressed
   65:             if (event.type == sf::Event::TextEntered) {
   66:                 char key = static_cast<char>(event.text.unicode);
   67:                 for (int i = 0; i < static_cast<int>(keys.length()); i++) {
   68:                     if (key == keys[i]) {
   69:                         sounds[i].play();   // play the sound sample
   70:                                             // equivalent to the key pressed
   71:                     }
   72:                 }
   73:             }
   74:         }
   75:         window.clear();
   76:         window.draw(sprite);
   77:         window.display();
   78:     }
   79:
   80:     return 0;
   81: }
   82:
   83: std::vector<sf::Int16> makeSamples(StringSound* gs) {
   84:     std::vector<sf::Int16> sampleStream;
   85:     gs->pluck();
   86:     int duration = 8;   // seconds
   87:     for (int i = 0; i < SAMPLE_RATE * duration; i++) {
   88:         gs->tic();
   89:         sampleStream.push_back(gs->sample());
   90:     }
   91:     return sampleStream;
   92: }
   93:
   94: void setupSoundBuffers(const std::vector<std::vector<sf::Int16>>& samples,
   95:  std::vector<sf::SoundBuffer>* soundBuffers) {
   96:     sf::SoundBuffer buffer;
   97:     for (int i = 0; i < static_cast<int>(samples.size()); i++) {
   98:         if (!buffer.loadFromSamples(&(samples[i])[0], samples[i].size(), 2,
   99:         SAMPLE_RATE)) {
  100:             throw std::runtime_error("setupSoundBuffers(): could not load"
  101:             " SoundBuffer");
  102:         }
  103:         soundBuffers->push_back(buffer);
  104:     }
  105: }
  106:
  107: void setupSounds(const std::vector<sf::SoundBuffer>& soundBuffers,
  108:  std::vector<sf::Sound>* sounds) {
  109:      sf::Sound sound;
  110:      for (int i = 0; i < static_cast<int>(soundBuffers.size()); i++) {
  111:         sound.setBuffer(soundBuffers[i]);
  112:         sounds->push_back(sound);
  113:      }
  114: }
```

test.cpp

test.cpp Wed Dec 02 16:36:32 2020 1

```
 1: // Copyright 2020 Matt Zagame
 2: #include "CircularBuffer.h"
 3: #define BOOST_TEST_DYN_LINK
 4: #define BOOST_TEST_MODULE Main
 5: #include <boost/test/unit_test.hpp>
 6:
 7: /** test to make sure CircularBuffer works as intended **/
 8: BOOST_AUTO_TEST_CASE(functions_test) {
 9:     std::cout << "\n************ Member Function Test ************" <<
10:     std::endl;
11:     std::cout << "Testing construction of CircularBuffer(4)" << std::endl;
12:     BOOST_REQUIRE_NO_THROW(CircularBuffer(4));
13:
14:     std::cout << "Testing enqueue() for CircularBuffer(4)" << std::endl;
15:     CircularBuffer cb(4);
16:     BOOST_REQUIRE_NO_THROW(cb.enqueue(1));
17:     BOOST_REQUIRE_NO_THROW(cb.enqueue(2));
18:     BOOST_REQUIRE_NO_THROW(cb.enqueue(3));
19:     BOOST_REQUIRE_NO_THROW(cb.enqueue(4));
20:     std::cout << "CircularBuffer(4) is full: " << std::boolalpha <<
21:     cb.isFull() << std::endl;
22:
23:     std::cout << "Testing dequeue() for CircularBuffer(4)" << std::endl;
24:     BOOST_REQUIRE_NO_THROW(cb.dequeue());
25:     BOOST_REQUIRE_NO_THROW(cb.dequeue());
26:     BOOST_REQUIRE_NO_THROW(cb.dequeue());
27:     BOOST_REQUIRE_NO_THROW(cb.dequeue());
28:     std::cout << "CircularBuffer(4) is empty: " << std::boolalpha <<
29:     cb.isEmpty() << std::endl << std::endl;
30: }
31:
32: /** test constructor exception: set capacity to less than 1 **/
33: BOOST_AUTO_TEST_CASE(excpetion_test1) {
34:     std::cout << "************** Exception Test 1 **************" <<
35:     std::endl << "Testing CircularBuffer(0) throws std::invalid_argument" <<
36:     std::endl << std::endl;
37:     BOOST_REQUIRE_THROW(CircularBuffer cb1(0), std::invalid_argument);
38: }
39:
40: /** test enqueue exception: attempt to add to a full buffer **/
41: BOOST_AUTO_TEST_CASE(excpetion_test2) {
42:     std::cout << "************** Exception Test 2 **************" <<
43:     std::endl << "Testing if full then enqueue() throws std::runtime_error"
44:     << std::endl << std::endl;
45:     CircularBuffer cb2(4);
46:     cb2.enqueue(1);
47:     cb2.enqueue(2);
48:     cb2.enqueue(3);
49:     cb2.enqueue(4);
50:     BOOST_REQUIRE_THROW(cb2.enqueue(5), std::runtime_error);
51: }
52:
53: /** test dequeue/peek exception: attempt to dequeue/peek first element of an
54:  * empty buffer **/
55: BOOST_AUTO_TEST_CASE(exception_test3) {
56:     std::cout << "************** Exception Test 3 **************" <<
57:     std::endl << "Testing if empty then peek() and dequeue() throw "
58:     "std::runtime_error" << std::endl;
59:     CircularBuffer cb3(1);
60:     BOOST_REQUIRE_THROW(cb3.peek(), std::runtime_error);
61:     BOOST_REQUIRE_THROW(cb3.dequeue(), std::runtime_error);

62: }
```

PS4 Edit Distance

This assignment asked us to compare two ASCII strings to compute their edit distance and to perform space and time analysis on the program during runtime. We were supposed to write a program that would compute an optimal sequence alignment on two DNA sequences using a dynamic programming approach. We could then use a programming tool known as Valgrind to measure the space complexity of our program.

What I Did:
My edit distance algorithm uses the dynamic programming approach of filling up a matrix with each calculation from bottom right to top left, resulting in the optimal distance at position [0][0]. The alignment path is also displayed by tracing the matrix in reverse order from top left to bottom right. The dynamic programming approach allows for the calculations of the alignment to be broken up into subproblems and then stored, instead of calculating the same subproblem several times.

Key Algorithms, Data Structures, and OOP Designs:
The ED (Edit Distance) class uses a vector of vectors of type int to represent a matrix of integers. The function optDistance() would then fill up the matrix from bottom to top with the minimum of three distances using the min() and penalty() functions, then return the optimal distance of the two sequences. The alignment() function would then traverse the populated matrix in reverse order and build the output alignment string based on the optimal path.

What I Learned:
Aside from the many applications of Edit Distance, I learned about the benefits of dynamic programming which, when compared to the recursive solution for this assignment, would have had a much higher space complexity due to the number of recursive calls exceeding 2^N when comparing two strings of length N. I also learned about how to effectively monitor program memory use using the Valgrind massif tool. This tool reports how much memory was used by the heap during execution. By applying the doubling method, I was able to calculate the estimated run time and memory use of a larger sample of strings of length N.

Makefile

```
CC = g++
CFLAGS = -std=c++11 -c -g -O2 -Wall -Werror -pedantic
LIBS = -lsfml-system
OBJS = main.o ED.o
EXE = ED

all : $(EXE)

$(EXE) : $(OBJS)
    $(CC) -o $(EXE) $(OBJS) $(LIBS)

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

clean :
    \rm $(EXE) $(OBJS)
```

ED.h

ED.h Fri Oct 30 18:18:31 2020 1

```
 1: // Copyright 2020 Matt Zagame
 2: #ifndef ED_H    //NOLINT
 3: #define ED_H    //NOLINT
 4:
 5: #include <iostream>
 6: #include <string>
 7: #include <vector>
 8: #include <algorithm>
 9:
10: class ED {
11:  public:
12:     ED();
13:
14:     // construct an Edit Distance object with two strings
15:     ED(std::string x, std::string y);
16:
17:     // returns the penalty for aligning chars a and b
18:     static int penalty(char a, char b);
19:
20:     // returns the minimum of three integers or a if they are all the same
21:     static int min(int a, int b, int c);
22:
23:     // populates the matrix and returns the optimal distance from [0][0]
24:     int optDistance();
25:
26:     /* trace the matrix and return a string of the alignment which can be
27:     displayed */
28:     std::string alignment();
29:
30:     // operator overloads
31:     friend std::istream& operator>>(std::istream& in, ED& ed);
32:
33:  private:
34:     int _M;       // length of string x
35:     int _N;       // length of string y
36:     std::string _x, _y;
37:
38:     // matrix used to measure optimal sequence alignment
39:     std::vector<std::vector<int>> _opt;
40: };
41: #endif  //NOLINT
```

ED.cpp

ED.cpp Tue Dec 01 19:39:47 2020 1

```
 1: // Copyright 2020 Matt Zagame
 2: #include "ED.h"
 3:
 4: ED::ED() : _M(0), _N(0) {}
 5:
 6: ED::ED(std::string x, std::string y) {
 7:     _x = x; _y = y;
 8:     _M = static_cast<int>(_x.length());
 9:     _N = static_cast<int>(_y.length());
10:
11:     // initialize vector matrix of ints
12:     _opt = std::vector<std::vector<int>>(_M+2, std::vector<int>(_N+2, 0));
13:
14:     // setup base cases
15:     for (int i = _M; i >= 0; i--) {
16:         _opt[i][_N+1] = _opt[i+1][_N+1] + 2;
17:     }
18:     for (int j = _N; j >= 0; j--) {
19:         _opt[_M+1][j] = _opt[_M+1][j+1] + 2;
20:     }
21: }
22:
23: int ED::penalty(char a, char b) {
24:     if (a == b)
25:         return 0;
26:     else
27:         return 1;
28: }
29:
30: int ED::min(int a, int b, int c) {
31:     if (b < a && b < c)
32:         return b;
33:     if (c < a && c < b)
34:         return c;
35:     else
36:         return a;
37: }
38:
39: int ED::optDistance() {
40:     // fill matrix with cost values from bottom right to top left
41:     for (int i = _M; i >= 0; i--) {
42:         for (int j = _N; j >= 0; j--) {
43:             _opt[i][j] = min(_opt[i+1][j+1] + penalty(_x[i], _y[j]),
44:             _opt[i+1][j] + 2, _opt[i][j+1] + 2);
45:         }
46:     }
47:     return _opt[0][0];
48: }
49:
50: std::string ED::alignment() {
51:     std::string result;     // result is in table form
52:
53:     if (_M != 0 && _N != 0) {   // make sure neither string is empty
54:         for (int i = 0; i < _M; i++) {
55:             for (int j = 0; j < _N; j++) {
56:                 // move diagonal on matrix
57:                 if (_opt[i][j] == _opt[i + 1][j + 1] && _x[i] == _y[j]) {
58:                     result.push_back(_x[i]);
59:                     result.append(" ");
60:                     result.push_back(_y[j]);
61:                     result.append(" 0\n");
```

```
   62:                     i++;
   63:                 } else if (_opt[i][j] == _opt[i + 1][j + 1] + 1) {
   64:                     result.push_back(_x[i]);
   65:                     result.append(" ");
   66:                     result.push_back(_y[j]);
   67:                     result.append(" 1\n");
   68:                     i++;
   69:                 } else if (_opt[i][j] == _opt[i + 1][j] + 2) {
   70:                     // move down
   71:                     result.push_back(_x[i]);
   72:                     result.append(" â\200\224 2\n");
   73:                     i++;
   74:                     j--;
   75:                 } else if (_opt[i][j] == _opt[i][j + 1] + 2) {
   76:                     // move right
   77:                     result.append("â\200\224 ");
   78:                     result.push_back(_y[j]);
   79:                     result.append(" 2\n");
   80:                 }
   81:             }
   82:         }
   83:     } else if (_M != 0 && _N == 0) {
   84:         result.append("String y is empty, returning string x: ");
   85:         result.append(_x);
   86:         result.append("\n");
   87:     } else if (_M == 0 && _N != 0) {
   88:         result.append("String x is empty, returning string y: ");
   89:         result.append(_y);
   90:         result.append("\n");
   91:     } else {
   92:         result.append("No two strings provided\n");
   93:     }
   94:     return result;
   95: }
   96:
   97: std::istream& operator>>(std::istream& in, ED& ed) {
   98:     std::string x;
   99:     std::string y;
  100:
  101:     in >> x;
  102:     in >> y;
  103:     ed = ED(x, y);
  104:
  105:     return in;
  106: }
```

main.cpp

main.cpp Fri Oct 30 21:20:36 2020 1

```
 1: // Copyright 2020 Matt Zagame
 2: #include "ED.h"
 3: #include <SFML/System.hpp>
 4:
 5: int main(int argc, char* argv[]) {
 6:     if (argc != 1) {
 7:         std::cerr << "usage: ./ED < example10.txt" << std::endl;
 8:         exit(-1);
 9:     }
10:     ED ed;
11:     std::cin >> ed;
12:     sf::Clock clock;
13:     sf::Time t;
14:
15:     std::cout << "Edit distance = " << ed.optDistance() << std::endl;
16:     std::cout << ed.alignment() << std::endl;
17:
18:     t = clock.getElapsedTime();
19:     std::cout << "Execution time is " << t.asSeconds() << " seconds" <<
20:     std::endl;
21:
22:     return 0;
23: }
```

PS5 Markov Model of Natural Language

In this assignment we were tasked with creating a program to represent a Markov chain that can analyze k-grams (a fixed number of characters) and keep a record of the probabilities of each character's occurrence. The program would then generate text using a Markov chain of given order k and any input text.

What I Did:
The Markov Model uses probabilistic analysis on text to determine the next character(s) in a sequence of k length words called k-grams. The MModel class takes a string and order k as input and maps each kgram in the string to each character following that kgram and it's frequency. The class can then generate a new string given a kgram based on the probability of each next character. The TextGenerator class uses this string generation function to analyze words in text files and produce a pseudorandom string of length L that attempts to recreate the given text.

Key Algorithms, Data Structures, and OOP Designs:
The MModel class uses a map containing a kgram key and a map of characters and their frequency as the respective value. MModel also makes use of the Mersenne Twister random number generator in the kRand() function. MModel's constructor sets up a map that can be displayed as a table-like output which holds each kgram and it's frequency, each next character and it's frequency as well as probability. The function generate() uses kRand() as a helper function, which selects a random next character from a kgram string when building a new string out of previously generated characters.

What I Learned:
Prefacing this assignment, I learned about how Markov chains are widely used in systems such as speech recognition, information retrieval, gene prediction, web search, etc. During the assignment, I was able to see first hand how the Markov model can produce reasonable text via a trajectory through a table of probabilities of k-grams.

Makefile

```
CC = g++
CFLAGS = -std=c++11 -c -g -O1 -Wall -Werror -pedantic
LIBS = -lboost_unit_test_framework
OBJS = TextGenerator.o MModel.o
EXE = TextGenerator

all : $(EXE)

$(EXE) : $(OBJS)
    $(CC) -o $(EXE) $(OBJS) $(LIBS)

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

clean :
    \rm $(EXE) $(OBJS)
```

MModel.h

MModel.h Tue Dec 01 19:45:14 2020 1

```
 1: // Copyright 2020 Matt Zagame
 2: #ifndef MMODEL_H  //NOLINT
 3: #define MMODEL_H  //NOLINT
 4:
 5: #include <iostream>
 6: #include <string>
 7: #include <map>
 8: #include <exception>
 9: #include <utility>
10: #include <random>
11:
12: class MModel {
13:  public:
14:     /* Create a Markov model of order k from given text (assume that text
15:     has length at least k). */
16:     MModel(std::string text, int k);
17:
18:     // Return the order k of the Markov model.
19:     int kOrder() const;
20:
21:     // Return the input text.
22:     std::string getText() const;
23:
24:     // Return the Markov Model table map.
25:     std::map<std::string, std::map<char, int>> getMTable() const;
26:
27:     /* Return the number of occurrences of kgram in text.
28:     Throws an exception if kgram is not at least of length k. */
29:     int freq(std::string kgram) const;
30:
31:     /* Return the number of times char c follows kgram.
32:     If order = 0, return the number of times char c occurs.
33:     Throws an exception if kgram is not at least of length k. */
34:     int freq(std::string kgram, char c) const;
35:
36:     /* Return a random character following given kgram.
37:     Throws an exception if kgram is not at least of length k.
38:     Throws an exception if no such kgram. */
39:     char kRand(std::string kgram) const;
40:
41:     /* Generate a string of length L characters by simulating a trajectory
42:     through the corresponding Markov chain. The first k characters of the
43:     newly generated string should be the argument kgram. Assume that L is at
44:     least k.
45:     Throws an exception if kgram is not at least of length k. */
46:     std::string generate(std::string kgram, int L) const;
47:
48:     /* Overloaded stream insertion operator displays the internal state of
49:     the Markov model. */
50:     friend std::ostream& operator<<(std::ostream& out, const MModel&
51:     mmodel);
52:
53:  private:
54:     int _k;             // order of Markov Model
55:     std::string _text;  // text to analyze
56:
57:     // map of kgram to map of frequency of next char
58:     std::map<std::string, std::map<char, int>> _mtable;
59: };
60: #endif   //NOLINT
```

MModel.cpp

MModel.cpp Tue Dec 01 19:43:45 2020 1

```
 1: // Copyright 2020 Matt Zagame
 2: #include "MModel.h"
 3:
 4: MModel::MModel(std::string text, int k) {
 5:     _text = text;
 6:     _k = k;
 7:
 8:     if (_text.length() < static_cast<unsigned int>(_k)) {
 9:         throw std::invalid_argument("MModel(string text, int k): order k"
10:         " must be less than or equal to text length.");
11:     }
12:
13:     // setup Markov table
14:     unsigned int pos = 0;
15:     for (unsigned int i = 0; i < _text.length(); i++) {
16:         std::string kgram;
17:         std::map<char, int> _ftable;    // map of next char to its frequency
18:
19:         // parse text for kgrams
20:         for (unsigned int j = i; j < i + _k; j++) {
21:             // get characters for kgrams
22:             if (j >= _text.length()) {
23:                 pos = j - _text.length();
24:             } else {
25:                 pos = j;
26:             }
27:             kgram += _text.at(pos);
28:         }
29:
30:         // setup next char frequency table
31:         pos++;
32:         if (pos >= _text.length()) { pos -= _text.length(); }
33:         _ftable.insert(std::make_pair(_text.at(pos), 0));
34:
35:         // insert kgram and frequency table into map
36:         if (_mtable.count(kgram) == 0) {
37:             _mtable.insert(std::make_pair(kgram, _ftable));
38:         }
39:
40:         // update next char frequency table
41:         _mtable[kgram][_text.at(pos)]++;
42:     }
43: }
44:
45: int MModel::kOrder() const { return _k; }
46: std::string MModel::getText() const { return _text; }
47: std::map<std::string, std::map<char, int>> MModel::getMTable() const {
48:     return _mtable;
49: }
50:
51: int MModel::freq(std::string kgram) const {
52:     if (kgram.length() < static_cast<unsigned int>(_k)) {
53:         throw std::runtime_error("freq(string kgram): kgram must be of"
54:         " length greater than or equal to order k.");
55:     }
56:     int count = 0;
57:     for (unsigned int i = 0; i < _text.length(); i++) {
58:         unsigned int pos = 0;
59:         std::string kg;
60:         // parse input text for kgrams
61:         for (unsigned int j = i; j < i + _k; j++) {
```

```
   62:             // get characters for kgrams
   63:             if (j >= _text.length()) {
   64:                 pos = j - _text.length();
   65:             } else {
   66:                 pos = j;
   67:             }
   68:             kg += _text.at(pos);
   69:         }
   70:         if (kgram == kg) { count++; }
   71:     }
   72:     return count;
   73: }
   74:
   75: int MModel::freq(std::string kgram, char c) const {
   76:     if (kgram.length() < static_cast<unsigned int>(_k)) {
   77:         throw std::runtime_error("freq(string kgram, char c): kgram must be"
   78:         " of length greater than or equal to order k.");
   79:     }
   80:     return _mtable.at(kgram).at(c);
   81: }
   82:
   83: char MModel::kRand(std::string kgram) const {
   84:     if (kgram.length() < static_cast<unsigned int>(_k)) {
   85:         throw std::runtime_error("kRand(string kgram): kgram must be of"
   86:         " length greater than or equal to order k.");
   87:     }
   88:     if (_mtable.count(kgram) == 0) {
   89:         throw std::runtime_error("kRand(string kgram): kgram does not"
   90:         " exist.");
   91:     }
   92:     std::string alphabet;   // string of next chars
   93:     for (auto const &var1 : _mtable) {
   94:         if (var1.first == kgram) {
   95:             for (auto const &var2 : var1.second) {
   96:                 alphabet += var2.first;
   97:             }
   98:         }
   99:     }
  100:     std::random_device device;
  101:     std::mt19937 mt_rand(device());
  102:     std::uniform_int_distribution<int> distribution(0, alphabet.length()
  103:     - 1);
  104:
  105:     return alphabet[distribution(mt_rand)];
  106: }
  107:
  108: std::string MModel::generate(std::string kgram, int L) const {
  109:     if (kgram.length() < static_cast<unsigned int>(_k)) {
  110:         throw std::runtime_error("generate(string kgram, int L): kgram must"
  111:         " be of length greater than or equal to order k.");
  112:     }
  113:     std::string generated = kgram;
  114:     // generate new characters based on kgrams
  115:     for (int i = _k; i < L; i++) {
  116:         generated += kRand(generated.substr(i - _k, _k));
  117:     }
  118:     return generated;
  119: }
  120:
  121: std::ostream& operator<<(std::ostream& out, const MModel& mmodel) {
  122:     out << "Markov Model\tOrder: " << mmodel._k << std::endl;
```

MModel.cpp Tue Dec 01 19:43:45 2020 3

```
  123:     out << "kgram:\tfrequency:\tfrqncy of next char:\tprob of next char:" <<
  124:     std::endl;
  125:
  126:     for (auto const &var1 : mmodel._mtable) {
  127:         // var1.first = kgram
  128:         out << var1.first << "\t";
  129:         out << mmodel.freq(var1.first) << "\t\t";
  130:         for (auto const &var2 : var1.second) {
  131:             // var2.first = next char
  132:             // var2.second = data
  133:             out << var2.first << ":" << var2.second << " ";
  134:         }
  135:         out << "\t\t\t";
  136:         for (auto const &var2 : var1.second) {
  137:             out << var2.first << ":" << var2.second << "/" <<
  138:             mmodel.freq(var1.first) << " ";
  139:         }
  140:         out << std::endl;
  141:     }
  142:     return out;
  143: }
```

TextGenerator.cpp

TextGenerator.cpp Wed Nov 11 22:49:13 2020 1

```
 1: // Copyright 2020 Matt Zagame
 2: #include "MModel.h"
 3: #include <fstream>
 4:
 5: int main(int argc, char *argv[]) {
 6:     if (argc != 3) {
 7:         std::cerr << "Usage: ./TextGenerator k L < input.txt" << std::endl;
 8:         exit(-1);
 9:     }
10:     int k = std::atoi(argv[1]);
11:     int L = std::atoi(argv[2]);
12:
13:     int count = 0;
14:     int length = 0;
15:     std::string input;
16:     std::string output;
17:
18:     // read input line by line and generate pseudo-random text
19:     while (std::getline(std::cin, input) && count < L) {
20:         if (input.length() > static_cast<unsigned int>(k)) {
21:             try {
22:                 MModel mmodel(input, k);
23:                 if (static_cast<int>(input.length()) > L) {
24:                     length = L;
25:                 } else if (static_cast<int>(input.length()) + count > L) {
26:                     length = L - count;
27:                 } else {
28:                     length = input.length();
29:                 }
30:                 output = mmodel.generate(input.substr(0, k), length);
31:                 count += output.length();
32:                 std::cout << output << std::endl;
33:             }
34:             catch (std::invalid_argument err) {
35:                 std::cerr << err.what() << std::endl;
36:                 exit(-1);
37:             }
38:             catch (std::runtime_error err) {
39:                 std::cerr << err.what() << std::endl;
40:                 exit(-1);
41:             }
42:         }
43:     }
44:
45:     return 0;
46: }
```

test.cpp

test.cpp Wed Nov 11 18:33:44 2020 1

```
 1: // Copyright 2020 Matt Zagame
 2: #include "MModel.h"
 3:
 4: #define BOOST_TEST_DYN_LINK
 5: #define BOOST_TEST_MODULE Main
 6: #include <boost/test/unit_test.hpp>
 7:
 8: BOOST_AUTO_TEST_CASE(base_test) {
 9:     std::cout << "******************** Test Case 1 ********************" <<
10:     std::endl;
11:
12:     int k = 2;
13:     std::string str = "gagggagaggcgagaaa";
14:     MModel mmodel(str, k);
15:
16:     std::cout << "Printing out Markov Table for string:\n" <<
17:     str << std::endl << std::endl;
18:     std::cout << mmodel << std::endl;
19:
20:     std::cout << "Testing kOrder and freq functions" << std::endl;
21:     BOOST_REQUIRE(mmodel.kOrder() == k);
22:     BOOST_REQUIRE(mmodel.freq("gg") == 3);
23:     BOOST_REQUIRE(mmodel.freq("ga", 'g') == 4);
24:
25:     std::cout << "Testing kRand function" << std::endl;
26:     char rand = mmodel.kRand("aa");
27:     BOOST_REQUIRE(rand == 'a' || rand == 'g');
28:
29:     std::cout << "Testing generate function" << std::endl << std::endl;
30:     BOOST_REQUIRE(mmodel.generate("ga", 10).length() == 10);
31: }
32:
33: BOOST_AUTO_TEST_CASE(exception_test) {
34:     std::cout << "******************** Test Case 2 ********************" <<
35:     std::endl;
36:     std::cout << "Testing construction exception: MModel('ADF', 4)" <<
37:     std::endl;
38:
39:     BOOST_REQUIRE_THROW(MModel("ADF", 4), std::invalid_argument);
40:
41:     std::cout << "Testing function exceptions" << std::endl;
42:     MModel testMM("abc", 3);
43:     BOOST_REQUIRE_THROW(testMM.freq("a"), std::runtime_error);
44:     BOOST_REQUIRE_THROW(testMM.freq("ab", 'b'), std::runtime_error);
45:     BOOST_REQUIRE_THROW(testMM.kRand("g"), std::runtime_error);
46: }
```

PS6 Kronos InTouch Log Parsing

This assignment focused on parsing Kronos InTouch time clock logs with regular expressions. Using regular expressions, we were to verify certain lines in the log and output whether a boot was successful or not in a separate file.

What I Did:
The program reads a log file line by line and determines whether a boot has started or completed depending on the message scanned. If it contains a boot start message then the date and time is saved and reported. Once a boot completed message is scanned, the date and time is saved again along with the total boot time from start to completion. This information is saved to a report file (.rpt) containing each successful and unsuccessful boot.

Key Algorithms, Data Structures, and OOP Designs:
The code for this assignment consisted of a single source file, main.cpp. I used the Boost libraries for both the regex functionality and working with the date and time. The only regex I used was for capturing the date and time reported on each boot started line or boot completed line. The actual boot messages were constant and could be scanned without the use of regex. The regex I used is as follows:

```
"^\\d{4}[-](0[1-9]|1[012])[-](0[1-9]|[12][0-9]|3[01])\\s\\d{2}[:]\\d{2}[:]\\d{2
}"
```

What I Learned:
Although this assignment was not a lengthy one, I learned the importance of regular expressions and how they can be useful when looking to scan for or extract string/character data. I also learned how to use the Boost date/time library to get the total boot time between the boot start and boot complete events in the log.

Makefile

```
CC = g++
CFLAGS = -std=c++11 -c -g -O1 -Wall -Werror -pedantic
LIBS = -lboost_regex -lboost_date_time
OBJS = main.o
EXE = ps6

all : $(EXE)

$(EXE) : $(OBJS)
    $(CC) -o $(EXE) $(OBJS) $(LIBS)

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

clean :
    \rm $(EXE) $(OBJS)
```

main.cpp

main.cpp Tue Dec 01 19:47:08 2020 1

```
 1: // Copyright 2020 Matt Zagame
 2: #include <iostream>
 3: #include <fstream>
 4: #include <string>
 5: #include <exception>
 6: #include <boost/regex.hpp>
 7: #include <boost/date_time.hpp>
 8:
 9: using boost::regex;
10: using boost::regex_search;
11: using boost::smatch;
12: using boost::posix_time::ptime;
13: using boost::posix_time::time_duration;
14: using boost::posix_time::time_from_string;
15:
16: int main(int argc, char* argv[]) {
17:     int lineNum = 1, bootStartCount = 0, bootCompleteCount = 0;
18:     bool bootStarted = false;
19:     const std::string bootStartMsg = "(log.c.166) server started";
20:     const std::string bootCompleteMsg = "oejs.AbstractConnector:Started "
21:     "SelectChannelConnector@0.0.0.0:9080";
22:     std::string s;
23:     std::string fileName;
24:     std::ifstream inputFile;
25:     std::ofstream outputFile;
26:     regex e("^\\d{4}[-](0[1-9]|1[012])[-](0[1-9]|[12][0-9]|3[01])\\s\\d{2}"
27:     "[:]\\d{2}[:]\\d{2}");  // regular expression to capture date and time
28:     smatch m;               // regex match results
29:     ptime t1, t2;           // t1 = boot start time, t2 = boot complete time
30:
31:     if (argc != 2) {
32:         std::cerr << "Usage: ./ps6 device1_intouch.log" << std::endl;
33:         return -1;
34:     }
35:
36:     // setup file I/O
37:     inputFile.open(argv[1]);
38:     if (!inputFile.is_open()) {
39:         std::cerr << "Could not open file: " << argv[1] << std::endl;
40:         return -1;
41:     }
42:     s = fileName = argv[1];
43:     outputFile.open(s.append(".rpt.tmp"));
44:     fileName = fileName.substr(fileName.find_last_of("\\/") + 1);
45:
46:     // scan boot info from log file into temp report file
47:     while (std::getline(inputFile, s)) {
48:         if (bootStarted) {
49:             if (s.find(bootCompleteMsg) != std::string::npos) {
50:                 // device boot complete reported
51:                 regex_search(s, m, e);    // get date and time
52:                 t2 = ptime(time_from_string(m[0]));
53:                 time_duration td = t2 - t1;     // get total boot time
54:                 outputFile << lineNum << "(" << fileName << ") " << m[0]
55:                 << " Boot Completed" << std::endl
56:                 << "\tBoot Time: " << td.total_milliseconds() << "ms"
57:                 << std::endl << std::endl;
58:                 bootStarted = false;
59:                 bootCompleteCount++;
60:             } else if (s.find(bootStartMsg) != std::string::npos) {
61:                 // device boot start reported after unsuccessful boot
```

```
   62:                 regex_search(s, m, e);    // get date and time
   63:                 t1 = ptime(time_from_string(m[0]));
   64:                 outputFile << "**** Incomplete boot ****" << std::endl <<
   65:                 std::endl
   66:                 << "=== Device boot ===" << std::endl
   67:                 << lineNum << "(" << fileName << ") " << m[0]
   68:                 << " Boot Start" << std::endl;
   69:                 bootStartCount++;
   70:             }
   71:         } else if (s.find(bootStartMsg) != std::string::npos) {
   72:             // device boot start reported
   73:             regex_search(s, m, e);    // get date and time
   74:             t1 = ptime(time_from_string(m[0]));
   75:             outputFile << "=== Device boot ===" << std::endl
   76:             << lineNum << "(" << fileName << ") " << m[0]
   77:             << " Boot Start" << std::endl;
   78:             bootStarted = true;
   79:             bootStartCount++;
   80:         }
   81:         lineNum++;
   82:     }
   83:     inputFile.close();
   84:     outputFile.close();
   85:
   86:     // finalize report file
   87:     s = argv[1];
   88:     s.append(".rpt");
   89:     outputFile.open(s);
   90:
   91:     s.append(".tmp");
   92:     inputFile.open(s);
   93:     if (!inputFile.is_open()) {
   94:         std::cerr << "Could not open file: " << s << std::endl;
   95:         return -1;
   96:     }
   97:
   98:     outputFile << "Device Boot Report" << std::endl << std::endl
   99:     << "InTouch log file: " << fileName << std::endl
  100:     << "Lines Scanned: " << lineNum - 1 << std::endl << std::endl
  101:     << "Device boot count: initiated: " << bootStartCount << ", completed: "
  102:     << bootCompleteCount << std::endl << std::endl << std::endl;
  103:
  104:     outputFile << inputFile.rdbuf();    // copy over temp file data
  105:
  106:     inputFile.close();
  107:     outputFile.close();
  108:
  109:     // remove temp report file
  110:     if (std::remove(s.c_str()) != 0) {
  111:         std::cerr << "Error deleting temp file: " << s << std::endl;
  112:         return -1;
  113:     }
  114:
  115:     return 0;
  116: }
```