Evan Mitchell

COMP IV: Project Portfolio

Spring 2019

Contents:

PS0 Hello World With SFML

Discussion

For the first assignment of the semester, we were tasked with writing code that would display a sprite on the screen and make it move around based on the keys we touched. This was my first time using the SFML API, so I was able to start learning how it works. Several objects are used in the code, such as sf::RenderWindow, sf::Sprite, and sf::Texture.

For the actual implementation, I created a window and used a loop to keep it open. Before displaying the window, however, I created a Texture which loaded my sprite from a file within my PS0 directory and created the sprite from that texture. Within the window loop I used the sf::Event class to check if any of the arrow keys were pressed, and moved the sprite accordingly. In addition, the "S" and "D" keys rotated the sprite.

Sprite:

PS0.cpp Source Code:

```
#include <SFML/Graphics.hpp>
int main()
{
    sf::RenderWindow window(sf::VideoMode(600, 600), "SFML window");
    sf::Texture texture;
    if (!texture.loadFromFile("sprite.png"))
      {
        return EXIT_FAILURE;
      }
    sf::Sprite my_sprite(texture);
    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(event.key.code == sf::Keyboard::Left)
                  my_sprite.move(-10, 0);
                if(event.key.code == sf::Keyboard::Right)
                  my_sprite.move(10, 0);
                if(event.key.code == sf::Keyboard::Up)
                  my_sprite.move(0, -10);
                if(event.key.code == sf::Keyboard::Down)
                  my_sprite.move(0, 10);

                if(event.key.code == sf::Keyboard::S)
                  my_sprite.rotate(-10);
                if(event.key.code == sf::Keyboard::D)
                  my_sprite.rotate(10);
              }
        }
        window.clear();
        window.draw(my_sprite);
        window.display();
    }
    return EXIT_SUCCESS;
}

```

PS1 LFSR and Image Encoding

Part A

Discussion

For part a of this assignment, we created a Load Feedback Shift Register (LFSR), which produces pseudo random numbers based on a seed. Given a binary seed, the LFSR will use an xor gate at a specific tap position to produce a new bit, which it inserts to the number being created. The LFSR was implemented using a class.

This assignment enabled me to explore Object Oriented programming and how it can be useful. Once I had the LFSR working and fully tested, part b of the assignment was much easier since I was able to use the member functions of the which I had already created. The LFSR constructor took a binary number in the form of a string, and an int which was the tap position.

In addition, the assignment helped me learn about test driven design, since we used the boost unit testing framework to test our code. I found the framework very helpful since I was able to use the boost API instead of writing my own tests from scratch.

Makefile:

```
CC=g++
CFLAGS= -std=c++11 -c -Wall
LDFLAGS=
SOURCES=LFSR.cpp test.cpp LFSR.hpp -lboost_unit_test_framework
OBJECTS=$(SOURCES:.cpp=.o)
EXECUTABLE= ps1a

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
	$(CC) $(LDFLAGS) $(OBJECTS) -o $@

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

clean:
	rm LFSR.o test.o ps1a LFSR.hpp~ LFSR.cpp~ test.cpp~ Makefile~
```

Test.cpp Source code (Unit Testing):

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

#include <iostream>
#include <string>
#include <sstream>

#include "LFSR.hpp"

 BOOST_AUTO_TEST_CASE(fiveBitsTapAtTwo) {

  LFSR l("00111", 2);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 0);

  LFSR l2("00111", 2);
  BOOST_REQUIRE(l2.generate(8) == 198);
}

 BOOST_AUTO_TEST_CASE(thirtyTwoBitsTapAtFive)
{
  LFSR l("11100101111001011110010111100101", 5);
  LFSR l2 = l;
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l2.generate(4) == 7);
}

 BOOST_AUTO_TEST_CASE(overloadedInsertionOperatorOutputsCorrectData)
{
  LFSR l("110010", 2);
  stringstream ss;
  ss << l;
  BOOST_REQUIRE(ss.str() == "110010");
}
```

LFSR.hpp Source Code:

```
#include <vector>
#include <string>
#include <iostream>
using namespace std;

class LFSR
{
 public:

 LFSR(string seed, int t);

 int step();

 int generate(int k);

 friend ostream& operator<<(ostream& out, const LFSR& right);

 ~LFSR();

private:
 string bit_sequence;
 int tap;

};
```

LFSR.cpp Source Code:

```
#include "LFSR.hpp"
```

```cpp
#include <vector>
#include <string>
#include <iostream>
using namespace std;

LFSR::LFSR(string seed, int t)
{
  tap = t;
  bit_sequence = seed;
}

int LFSR::step()
{
  int result = 0;
  if(bit_sequence[0] != bit_sequence[bit_sequence.size() - 1 - tap])
    {
      result = 1;
    }
  for(int i = 0; i < bit_sequence.size() - 1; i++)
    {
      bit_sequence[i] =  bit_sequence[i + 1];
    }
  bit_sequence[bit_sequence.size() - 1] = result + 48;
  return result;
}

int LFSR::generate(int k)
{
  int val = 0;
  for(int i = 0; i < k; i++)
    {
      val *= 2;
      val += step();
    }
  return val;
}

LFSR::~LFSR()
{


}

ostream& operator<<(ostream& out, const LFSR& right)
{
  out << right.bit_sequence;
  return out;
}
```

Part B

Discussion

For this part of the assignment, we used the LFSR we had created in part a to encode an image by randomizing the bits. Due to the xor property, we were able to decode the image as well if we knew the original seed number.

The key part of the assignment begins on line 21, where I work with the sf::Pixel class on the bit level to scramble the rgb values by xoring them with the pseudo random value generated by the LFSR seed. This effectively scrambled the image into something undistinguishable. Due to the xor property of xoring any bit string twice with the bit string resulting in the same original string, doing this process twice with the same tap and seed reverts the image back to its unscrambled state.

Due to already having created the LFSR in part a, creating the pseudo-random bit string was trivial. I needed bit strings of length 8, since r, b and g are represented with 8 bits each in this case. The LFSR::generate function takes an int argument, k, and returns a psuedo-random integer with the least significant k bits randomly set. The program takes two command line arguments, the seed length and the tap position.

Running the program twice on the same image with the same command line arguments produced the original image, after initially scrambling it. I further solidified my understand of object oriented programming through this assignment, in addition to getting experience using and designing my own classes.

PhotoMagic.cpp Source Code:

```
1 #include <iostream>
2 #include "LFSR.hpp"
3 #include <SFML/System.hpp>
4 #include <SFML/Window.hpp>
5 #include <SFML/Graphics.hpp>
6 #include <string>
7
8 int main(int argc, char* argv[])
9 {
10          sf::Image input_image;
11          if (!input_image.loadFromFile(argv[1]))
12                  return -1;
13
14          sf::Image output_image;
15          if (!output_image.loadFromFile(argv[1]))
16                  return -1;
17
18          sf::Vector2u size = output_image.getSize();
19          sf::Color p;
```

```
        LFSR scrambler(argv[3], stoi(argv[4]));
        for (int x = 0; x< size.x; x++) {
                for (int y = 0; y< size.y; y++) {
                        p = output_image.getPixel(x, y);
                        p.r ^= scrambler.generate(8);
                        p.g ^= scrambler.generate(8);
                        p.b ^= scrambler.generate(8);
                        output_image.setPixel(x, y, p);
                }
        }


        sf::RenderWindow input_window(sf::VideoMode(size.x, size.y), "Input_Window");
        sf::Texture input_texture;
        input_texture.loadFromImage(input_image);
        sf::Sprite input_sprite;
        input_sprite.setTexture(input_texture);

        sf::RenderWindow output_window(sf::VideoMode(size.x, size.y), "Output_Window");
        sf::Texture output_texture;
        output_texture.loadFromImage(output_image);
        sf::Sprite output_sprite;
       output_sprite.setTexture(output_texture);

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

        if (!output_image.saveToFile(argv[2]))
               return -1;

        return 0;
}

```

PS2 Recursive Graphics (Pythagoras Tree)

Discussion

In this assignment we used recursion to draw the Pythagoras Tree. The Pythagoras Tree is a fractal, meaning every part of the tree is a smaller version of itself. We used the SFML libraries to create the graphics, and also created a class that actually draws the tree.

The Pythagoras tree draws itself based on a given recursion depth, however the final tree with a L x L base square will always fit into a 6L x 4L rectangle, which is the size of the display window. The tree is drawn recursively, and a solution that I found that works is to use an 8 way switch statement that draws the corresponding two squares based on its given orientation, which is passed in the recursive function.

I used an enum to represent directions, with 0 being north up to 7 being north west. Each square calls the recursive function two times, since each individual square has two squares attached to it. All the calculations necessary to draw the new square can be found with the preceding squares size and position, as well as the new orientation which is passed as a parameter. For example, the new square size is the size of the old square divided by the square root of 2. The terminating condition is when the recursion count variable reaches 0. The recursive function is called with a decremented value with each call to account for this.

I was able to get a lot of practice implementing recursive algorithms doing the assignment. In addition, I learned about using the SFML API.

Makefile:

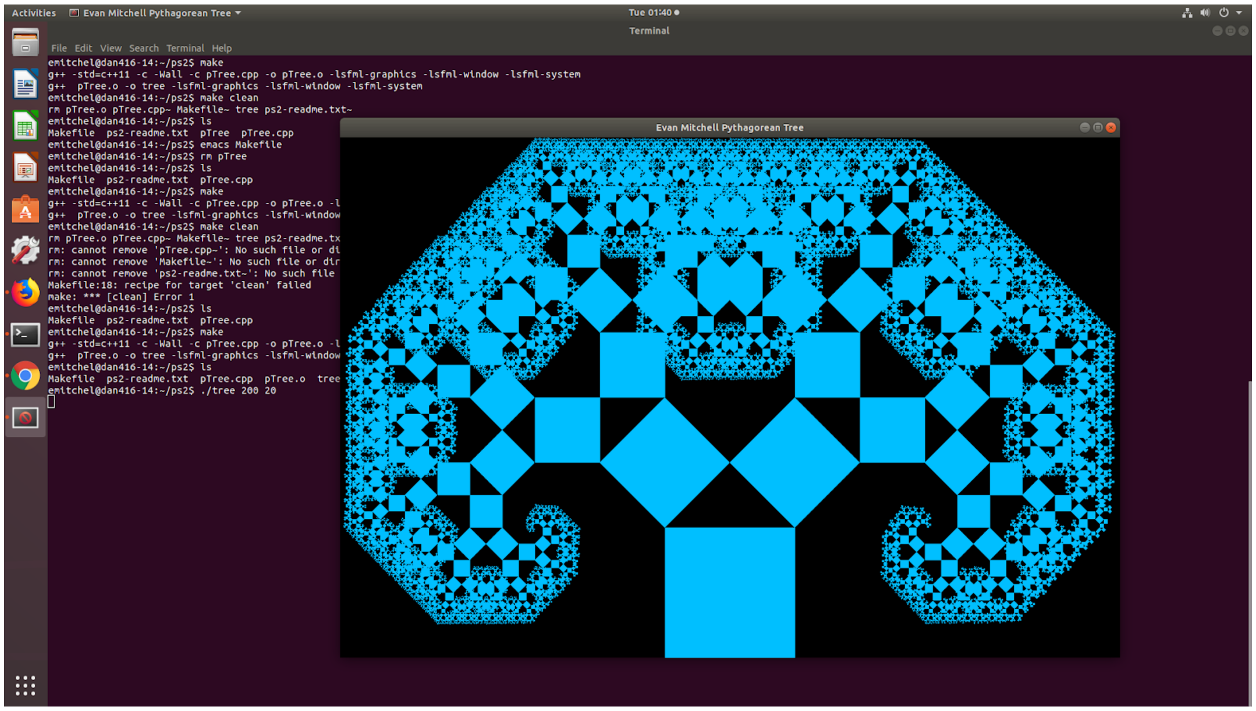
```
CC = g++
CFLAGS= -std=c++11 -c -Wall
LDFLAGS=
SOURCES= pTree.cpp
OBJECTS=$(SOURCES:.cpp=.o)
LIBS= -lsfml-graphics -lsfml-window -lsfml-system
EXECUTABLE = tree

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
	$(CC) $(LDFLAGS) $(OBJECTS) -o $@ $(LIBS)

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

clean:
	rm pTree.o pTree.cpp~ Makefile~ tree ps2-readme.txt~

```

PTree.cpp Source Code:

```cpp
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include <iostream>
#include <math.h>
#include <string>

enum direction
  {
    N = 0, NE = 1, E = 2, SE = 3, S = 4, SW = 5, W = 6, NW = 7
  };

class PTree : public sf::Drawable
{
public:

  //constructor
  PTree(double start_size, int depth)
  {
    base_square_size = start_size;
    recursion_depth = depth;
  }
  virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const
  {
    sf::Vector2f init_square(base_square_size, base_square_size);
    sf::RectangleShape rect(init_square);
    rect.setPosition(2.5 * base_square_size, 3 * base_square_size);
    rect.setFillColor(sf::Color( 0, 191, 255, 255));
    target.draw(rect);
    pTree(rect, target, recursion_depth, N);
  }
private:
  double base_square_size;
  int recursion_depth;


  void pTree(sf::RectangleShape old_rect, sf::RenderTarget& target, int recursion_count, direction dir) const
  {
    if(recursion_count <= 0)
      {
        return;
      }

    sf::Transform t_left, t_right;
    sf::Vector2f old_size = old_rect.getSize();
    double new_size = (old_size.x / sqrt(2));
    sf::Vector2f new_rect_size(new_size, new_size);
    sf::RectangleShape new_rect_left(new_rect_size);
    sf::RectangleShape new_rect_right(new_rect_size);
    new_rect_left.setPosition(old_rect.getPosition());
    new_rect_right.setPosition(old_rect.getPosition());
    new_rect_left.setFillColor(sf::Color(0, 191, 255, 255));
    new_rect_right.setFillColor(sf::Color(0, 191, 255, 255));

    switch(dir)
      {
      case N: //north
        {
          new_rect_left.move(-0.5*new_size, -0.5 * (old_size.x +
new_size));
          sf::Vector2f new_position_left = new_rect_left.getPosition();
          sf::Vector2f rotate_center_left(new_position_left.x + (0.5 *
new_size),
  new_position_left.y + (0.5 * new_size));
          t_left.rotate(45, rotate_center_left);

          target.draw(new_rect_left, t_left);
          new_rect_left.setPosition(new_position_left);


          new_rect_right.move(old_size.x - 0.5 * new_size, -0.5 *
(old_size.x + n
ew_size));
          sf::Vector2f new_position_right = new_rect_right.getPosition();
          sf::Vector2f rotate_center_right(new_position_right.x + (0.5 *
new_size
), new_position_right.y + (0.5 * new_size));
          t_right.rotate(45, rotate_center_right);

          target.draw(new_rect_right, t_right);
          new_rect_right.setPosition(new_position_right);

      pTree(new_rect_left, target, --recursion_count, NW);
      pTree(new_rect_right, target, --recursion_count, NE);
        }
      break;


      case NE: //NE
      {
        sf::Vector2f old_position = old_rect.getPosition();
        sf::Vector2f old_center(old_position.x + (old_size.x * 0.5),
old_positio
n.y  + (old_size.x * 0.5));

        sf::Vector2f offset_left(0, -2 * new_size);
        sf::Vector2f new_position_left(old_center + offset_left);
        new_rect_left.setPosition(new_position_left);
        target.draw(new_rect_left);

        sf::Vector2f offset_right(new_size, -1 * new_size);
        sf::Vector2f new_position_right(old_center + offset_right);
        new_rect_right.setPosition(new_position_right);
        target.draw(new_rect_right);

      pTree(new_rect_left, target, --recursion_count, N);
      pTree(new_rect_right, target, --recursion_count, E);
      }
      break;

      case E: //E
      {
        new_rect_left.move((1.5*old_size.x) - (0.5 * new_size), -0.5 * new_size);
         sf::Vector2f new_position_left = new_rect_left.getPosition();
         sf::Vector2f rotate_center_left(new_position_left.x + (0.5 * new_size), new_position_left.y + (0.5 * new_size));
         t_left.rotate(45, rotate_center_left);

         target.draw(new_rect_left, t_left);
         new_rect_left.setPosition(new_position_left);


         new_rect_right.move((1.5 * old_size.x) - (0.5 * new_size), old_size.x - (0.5 * new_size));
         sf::Vector2f new_position_right = new_rect_right.getPosition();
         sf::Vector2f rotate_center_right(new_position_right.x + (0.5 * new_size), new_position_right.y + (0.5 * new_size));
         t_right.rotate(45, rotate_center_right);

         target.draw(new_rect_right, t_right);
         new_rect_right.setPosition(new_position_right);

      pTree(new_rect_left, target, --recursion_count, NE);
      pTree(new_rect_right, target, --recursion_count, SE);
      }
      break;


      case SE: //SE
      {
       sf::Vector2f old_position = old_rect.getPosition();
        sf::Vector2f old_center(old_position.x + (old_size.x * 0.5), old_position.y  + (old_size.x * 0.5));

        sf::Vector2f offset_left(0, new_size);
        sf::Vector2f new_position_left(old_center + offset_left);
        new_rect_left.setPosition(new_position_left);
        target.draw(new_rect_left);

        sf::Vector2f offset_right(new_size, 0);
        sf::Vector2f new_position_right(old_center + offset_right);
        new_rect_right.setPosition(new_position_right);
        target.draw(new_rect_right);

      pTree(new_rect_left, target, --recursion_count, S);//S
      pTree(new_rect_right, target, --recursion_count, E); //E
      }
      break;


      case S://S
      {
         new_rect_left.move(-0.5 * new_size, (1.5 * old_size.x) - (0.5 * new_size) );
         sf::Vector2f new_position_left = new_rect_left.getPosition();
         sf::Vector2f rotate_center_left(new_position_left.x + (0.5 * new_size), new_position_left.y + (0.5 * new_size));
         t_left.rotate(45, rotate_center_left);

         target.draw(new_rect_left, t_left);
         new_rect_left.setPosition(new_position_left);


         new_rect_right.move( old_size.x - (0.5 * new_size),  (1.5 * old_size.x) - (0.5 * new_size) );
         sf::Vector2f new_position_right = new_rect_right.getPosition();
         sf::Vector2f rotate_center_right(new_position_right.x + (0.5 * new_size), new_position_right.y + (0.5 * new_size));
         t_right.rotate(45, rotate_center_right);

         target.draw(new_rect_right, t_right);
         new_rect_right.setPosition(new_position_right);

         pTree(new_rect_left, target, --recursion_count, SW); //SW
         pTree(new_rect_right, target, --recursion_count, SE); //SE
      }
      break;

      case SW: //SW
      {
        sf::Vector2f old_position = old_rect.getPosition();
        sf::Vector2f old_center(old_position.x + (old_size.x * 0.5), old_position.y  + (old_size.x * 0.5));

        sf::Vector2f offset_left(-2 * new_size, 0);
        sf::Vector2f new_position_left(old_center + offset_left);
        new_rect_left.setPosition(new_position_left);
        target.draw(new_rect_left);

        sf::Vector2f offset_right(-1 * new_size, new_size);
        sf::Vector2f new_position_right(old_center + offset_right);
        new_rect_right.setPosition(new_position_right);
        target.draw(new_rect_right);

        pTree(new_rect_right, target, --recursion_count, S);//S
        pTree(new_rect_left, target, --recursion_count, W); //W
      }
      break;

    case W:
      {
         new_rect_left.move(-0.5 * (new_size + old_size.x), -0.5 * new_size);
         sf::Vector2f new_position_left = new_rect_left.getPosition();
         sf::Vector2f rotate_center_left(new_position_left.x + (0.5 *
new_size),
 new_position_left.y + (0.5 * new_size));
         t_left.rotate(45, rotate_center_left);

         target.draw(new_rect_left, t_left);
         new_rect_left.setPosition(new_position_left);


         new_rect_right.move( -0.5 * (new_size + old_size.x), old_size.x -
 (0.5
* new_size));
         sf::Vector2f new_position_right = new_rect_right.getPosition();
         sf::Vector2f rotate_center_right(new_position_right.x + (0.5 *
new_size
), new_position_right.y + (0.5 * new_size));
         t_right.rotate(45, rotate_center_right);

         target.draw(new_rect_right, t_right);
         new_rect_right.setPosition(new_position_right);

      pTree(new_rect_right, target, --recursion_count, SW);
      pTree(new_rect_left, target, --recursion_count, NW);
      break;
      }
    case NW:
      {
        sf::Vector2f old_position = old_rect.getPosition();
        sf::Vector2f old_center(old_position.x + (old_size.x * 0.5),
old_positio
n.y  + (old_size.x * 0.5));

        sf::Vector2f offset_left(-2 * new_size,  -1 * new_size);
        sf::Vector2f new_position_left(old_center + offset_left);
        new_rect_left.setPosition(new_position_left);
        target.draw(new_rect_left);

        sf::Vector2f offset_right(-1 * new_size, -2 * new_size);
        sf::Vector2f new_position_right(old_center + offset_right);
        new_rect_right.setPosition(new_position_right);
        target.draw(new_rect_right);

      pTree(new_rect_right, target, --recursion_count, N);
      pTree(new_rect_left, target, --recursion_count, W);
      }
      break;

      }

  }

};



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

```
const double BASE_SQUARE_SIZE = std::stoi(argv[1]);
const int RECURSION_DEPTH = std::stoi(argv[2]);
sf::RenderWindow window(sf::VideoMode(6 * BASE_SQUARE_SIZE, 4 *
BASE_SQUARE_SI
ZE), "Evan Mitchell Pythagorean Tree");
PTree tree(BASE_SQUARE_SIZE, RECURSION_DEPTH);
while (window.isOpen())
  {
      // Process events
      sf::Event event;
      while (window.pollEvent(event))
      {
          // Close window: exit
          if (event.type == sf::Event::Closed)
              window.close();
      }

      window.clear();
      window.draw(tree);
      window.display();
  }
return 1;
}
```

Output from running code:


Activities
Evan Mitchell Pythagorean Tree
Tue 01:40
Terminal
File Edit View Search Terminal Help
Evan Mitchell Pythagorean Tree

```
emitchel@dan416-14:~/ps2$ make
g++ -std=c++11 -c -Wall -c pTree.cpp -o pTree.o -lsfml-graphics -lsfml-window -lsfml-system
g++  pTree.o -o tree -lsfml-graphics -lsfml-window -lsfml-system
emitchel@dan416-14:~/ps2$ make clean
rm pTree.o pTree.cpp~ Makefile~ tree ps2-readme.txt~
emitchel@dan416-14:~/ps2$ ls
Makefile  ps2-readme.txt  pTree  pTree.cpp
emitchel@dan416-14:~/ps2$ emacs Makefile
emitchel@dan416-14:~/ps2$ rm pTree
emitchel@dan416-14:~/ps2$ ls
Makefile  ps2-readme.txt  pTree.cpp
emitchel@dan416-14:~/ps2$ make
g++ -std=c++11 -c -Wall -c pTree.cpp -o pTree.o -l
g++  pTree.o -o tree -lsfml-graphics -lsfml-window
emitchel@dan416-14:~/ps2$ make clean
rm pTree.o pTree.cpp~ Makefile~ tree ps2-readme.tx
rm: cannot remove 'pTree.cpp~': No such file or di
rm: cannot remove 'Makefile~': No such file or dir
rm: cannot remove 'ps2-readme.txt~': No such file
Makefile:18: recipe for target 'clean' failed
make: *** [clean] Error 1
emitchel@dan416-14:~/ps2$ ls
Makefile  ps2-readme.txt  pTree.cpp
emitchel@dan416-14:~/ps2$ make
g++ -std=c++11 -c -Wall -c pTree.cpp -o pTree.o -l
g++  pTree.o -o tree -lsfml-graphics -lsfml-window
emitchel@dan416-14:~/ps2$ ls
Makefile  ps2-readme.txt  pTree.cpp  pTree.o  tree
emitchel@dan416-14:~/ps2$ ./tree 200 20
```

PS3 N Body Simulation

Part A

Discussion

For PS3, we created an N-Body simulation, which reads n bodies in from a txt file and creates a simulation based on their gravitational pull. For Part A, we were tasked with creating a static simulation, which took a file in as a command line argument and created N objects that represented the bodies. For this assignment, we stored all the internal data from the file and displayed a static image, but did not make them move based on their velocity, mass and other forces.

I created a Body class, which stores all the internal data. It has a default constructor, and an overloaded istream operator. The istream operator is designed to take an istream as the left hand operand, and a body object as the right hand operand. This is the tool I used to read in the data from the file. Internal data includes x position, y position, y velocity, x velocity, mass, and filename. The body class inherits from sf::Drawable, so that it is in line with the functionality of other sf::Drawable classes. I had to overload the virtual draw function inherited from drawable so that I could use the window.draw(body) syntax that other sf::drawable objects use.

This assignment was very useful in learning more about designing my own classes and using them. In addition, I was able to practice inheritance and operator overloading.

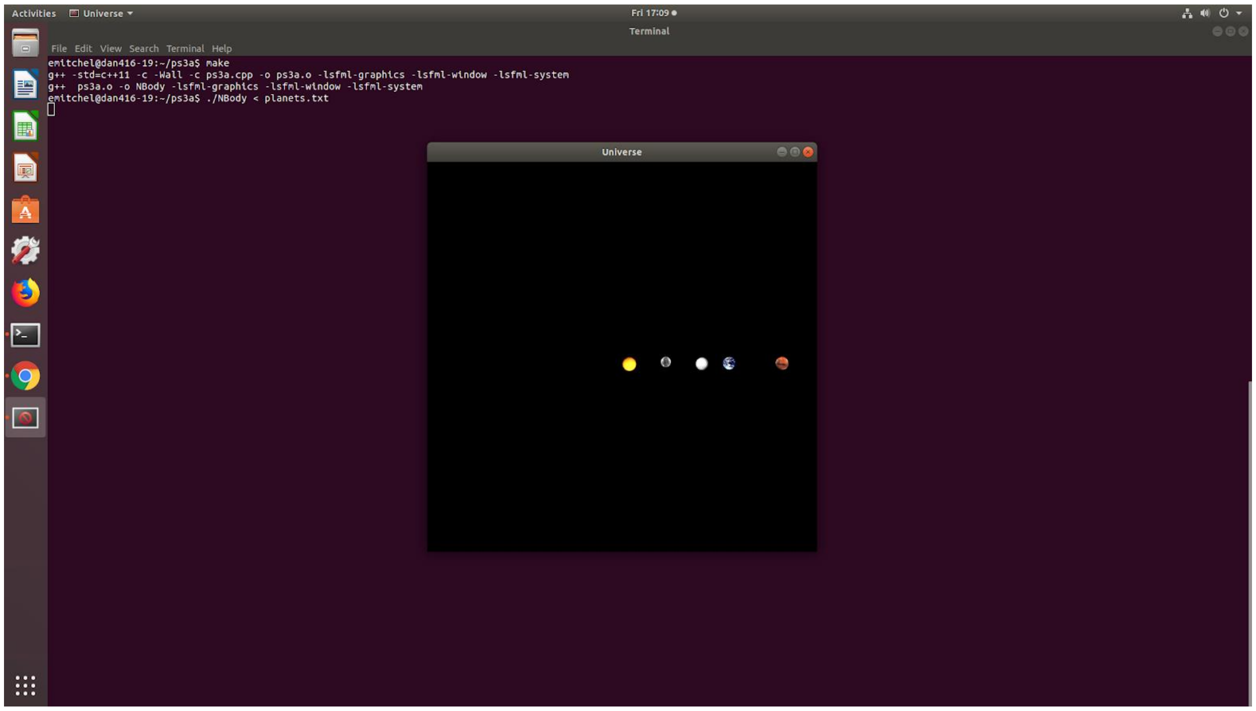
Makefile:

```
CC = g++
CFLAGS= -std=c++11 -c -Wall
LDFLAGS=
SOURCES= ps3a.cpp
OBJECTS=$(SOURCES:.cpp=.o)
LIBS= -lsfml-graphics -lsfml-window -lsfml-system
EXECUTABLE = NBody

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
	$(CC) $(LDFLAGS) $(OBJECTS) -o $@ $(LIBS)

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

clean:
	rm ps3a.o ps3a~

```

Ps3a.cpp Source Code:

```cpp
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include <stdio.h>
#include <string.h>

class Body : public sf::Drawable
{
public:
  //constructors
  Body(double x, double y, double velocity_x, double velocity_y, double mass, std::string file_name) : x(x), y(y),
  velocity_x(velocity_x), velocity_y(velocity_y), mass(mass), file_name(file_name) { }

  Body() : x(0), y(0), velocity_x(x), velocity_y(0), mass(0)
  {
    universe_radius = 0;
    file_name = "";
  }
  //overloaded istream
  friend std::istream &operator>>( std::istream  &input, Body& b )
  {
    std::string line;
    std::vector<std::string> numbers;

    std::getline(std::cin, line);
    char * c_line = strdup(line.c_str());
    char * token;
    token = strtok(c_line, " ");
    while(token != NULL)
      {
    numbers.push_back(token);
    token = strtok(NULL, " ");
      }
    free(c_line);
    free(token);

   std::string::size_type sz;
      b.x = std::stod(numbers[0], &sz);
      b.x *= 1.03626943e-9;
      b.y = std::stod(numbers[1], &sz);
      b.velocity_x = std::stod(numbers[2], &sz);
      b.velocity_y = std::stod(numbers[3], &sz);
      b.mass =std:: stod(numbers[4], &sz);
      b.file_name = numbers[5];

      return input;

  }

  void set_sprite()
  {
    if(!texture.loadFromFile(file_name))
      {
          std::cout << "Failed to load image. Exiting." << std::endl;
          exit(1);
      }
      sprite.setTexture(texture);
      sprite.setPosition(sf::Vector2f((universe_radius / 2) + x, (universe_radius / 2) + y));
  }
  void set_universe_radius(double radius)
  {
    universe_radius = radius;
  }

private:

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

  double universe_radius;
  long int x, y, velocity_x, velocity_y, mass;
  std::string file_name;
  sf::Sprite sprite;
   sf::Texture texture;
};

int main()
{
  double  number_of_planets, universe_radius;
  std::string str_number_of_planets, str_universe_radius;

   std::string::size_type sz;

  std::getline(std::cin, str_number_of_planets);
  number_of_planets = stod(str_number_of_planets, &sz);

  std::getline(std::cin, str_universe_radius);
  universe_radius = stod(str_universe_radius, &sz);
  universe_radius *= 2.4e-9;

  std::vector<Body*> universe;
  //set up vector
   for(int i = 0; i < number_of_planets; i++)
    {
      universe.push_back(new Body());
      universe[i]->set_universe_radius(universe_radius);
      std::cin >> *universe[i];
      universe[i]->set_sprite();
    }
  //display planets
  sf::RenderWindow window(sf::VideoMode(universe_radius, universe_radius), "Universe");
  while (window.isOpen())
  {
    sf::Event event;
    while (window.pollEvent(event))
    {
      if (event.type == sf::Event::Closed)
        window.close();
    }
    window.clear();
   for(int i = 0; i < number_of_planets; i++)
  {
   universe.push_back(new Body());
   universe[i]->set_universe_radius(universe_radius);
   std::cin >> *universe[i];
   universe[i]->set_sprite();
  }
 //display planets
 sf::RenderWindow window(sf::VideoMode(universe_radius, universe_radius), "Universe");
  while (window.isOpen())
  {
    sf::Event event;
    while (window.pollEvent(event))
    {
      if (event.type == sf::Event::Closed)
        window.close();
    }
    window.clear();
   for(int i = 0; i < number_of_planets; i++)
  {
    window.draw(*universe[i]);
  }
   window.display();
  }
}
```

Displaying a Static Solar System:



```
emitchel@dan416-19:~/ps3a$ make
g++ -std=c++11 -c -Wall -c ps3a.cpp -o ps3a.o -lsfml-graphics -lsfml-window -lsfml-system
g++  ps3a.o -o NBody -lsfml-graphics -lsfml-window -lsfml-system
emitchel@dan416-19:~/ps3a$ ./NBody < planets.txt
```

Part B

Discussion

For Part B of this assignment we were tasked with changing the static universe that we created in part a into a dynamic one. We continued using the body class for the implementation.

Since it is an N body simulation, not a solar system simulation, it must work for a generic number of bodies. To do this, I created a vector of pointers to bodies, which I referred to as the universe, and then stored all of the internal data (mass, velocity, etc.), in each body. Then, I used a for loop to go through each body in the vector and calculate the net force acting on it based on all the other bodies. The positions were then updated and the universe was displayed.

How often the universe was displayed relative to the movements of the bodies was based on a value delta t. Large values of delta t made the universe be updated less frequently, while smaller values made the simulation more accurate by updating more frequently. The time elapsed in seconds is shown on the top left as well.

This assignment was very useful for getting more experiencing using and designing classes, since the SFML API relies heavily on them.

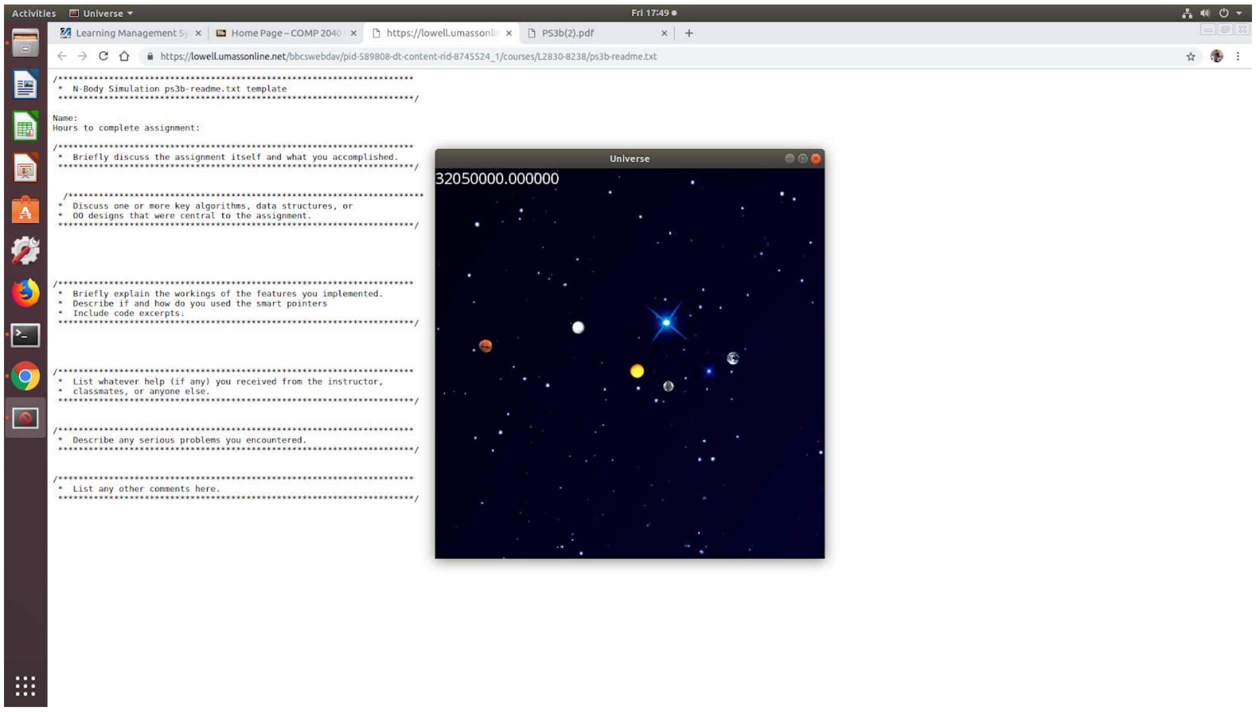
Ps3b.cpp Source Code:

```
//Evan Mitchell

#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include <stdio.h>
#include <string.h>
#include <cmath>
#include <thread>
#include <chrono>

const double G_CONST = 6.67e-11;
const double SCALE_UP = 1.00396943e9;
const double SCALE_DOWN = 1.00396943e-9;

class Body : public sf::Drawable
{
public:
  //constructor

  Body() : x(0), y(0), velocity_x(x), velocity_y(0), mass(0),
acceleration_x(0), acceleration_y(0), net_force_x(0), net_force_y(0)
  {
    universe_radius = 0;
    file_name = "";
  }
  //overloaded istream
  friend std::istream &operator>>( std::istream  &input, Body& b )
  {
    std::string line;
    std::vector<std::string> numbers;

    std::getline(std::cin, line);
    char * c_line = strdup(line.c_str());
    char * token;
    token = strtok(c_line, " ");
    while(token != NULL)
   {
    numbers.push_back(token);
    token = strtok(NULL, " ");
      }
    free(c_line);
    free(token);

      std::string::size_type sz;
      b.x = std::stod(numbers[0], &sz);
      b.x *= SCALE_DOWN;
      b.x = b.x + (b.universe_radius / 2);
```

```cpp

      b.y = std::stod(numbers[1], &sz);
      b.y *= SCALE_DOWN;
      b.y = b.y + (b.universe_radius / 2);

      b.velocity_x = std::stod(numbers[2], &sz);
      b.velocity_y = std::stod(numbers[3], &sz);
      b.mass =std:: stod(numbers[4], &sz);
      b.file_name = numbers[5];

      return input;
  }

  void set_sprite()
  {
    if(!texture.loadFromFile(file_name))
      {
          std::cout << "Failed to load image." << std::endl;
      }
      sprite.setTexture(texture);
      sprite.setPosition(sf::Vector2f(x, y));
  }

  void set_universe_radius(double radius){ universe_radius = radius; }

  void set_d_time(double time) {D_TIME = time;}

  void adjust_net_force(const Body& other_body) //adjust net force in the body based on the other body that is acting on it
  {
    double length_apart, x_apart, y_apart;
    x_apart = ((other_body.get_x() * SCALE_UP) - (x * SCALE_UP));
    y_apart = ((other_body.get_y() * SCALE_UP) - (y * SCALE_UP));

  length_apart = sqrt(((x_apart * x_apart) + (y_apart * y_apart)));

  double force = (mass * other_body.get_mass() * G_CONST) / (length_apart * length_apart);

  net_force_x = (x_apart * force) / length_apart;
  net_force_y = (y_apart * force) / length_apart;


  }

   void adjust_acceleration()
  {
    acceleration_x = net_force_x / mass;
    acceleration_y = net_force_y / mass;
  }

   void adjust_velocity()
   {
     velocity_x = velocity_x + (D_TIME * acceleration_x);
     velocity_y = velocity_y + (D_TIME * acceleration_y);
   }

```

```cpp
void adjust_position()
{
  x = x + (D_TIME * velocity_x * SCALE_DOWN);
  y = y + (D_TIME * velocity_y * SCALE_DOWN);
  sprite.setPosition(sf::Vector2f(x, y));
}

  double get_net_force_x() const {return net_force_x;}
  double get_net_force_y() const {return net_force_y;}
  double get_velocity_x() const {return velocity_x;}
  double get_velocity_y() const {return velocity_y;}
  double get_x() const {return x;}
  double get_y() const {return y;}
  double get_mass() const {return mass;}
  std::string get_file_name() const {return file_name;}

private:

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

  double D_TIME;
  double universe_radius;
  double x, y, velocity_x, velocity_y, mass;
  std::string file_name;
  double acceleration_x, acceleration_y;
  double net_force_x;
  double net_force_y;
  sf::Sprite sprite;
  sf::Texture texture;
};

void step(double& seconds, std::vector<Body*>& universe, double D_TIME)
  {
    for(int i = 0; i < universe.size(); i++)//calculate net force on each body based on other bodies in the simulation
      {
        for(int j = i + 1; j < universe.size(); j++) //above the current body...
         {
          universe[i]->adjust_net_force(*universe[j]);
          universe[i]->adjust_acceleration();
          universe[i]->adjust_velocity();
        }
        for(int k = 0; k < i; k++) //... and below
        {
          universe[i]->adjust_net_force(*universe[k]);
          universe[i]->adjust_acceleration();
          universe[i]->adjust_velocity();
        }
      }

    for(int i = 0; i < universe.size(); i++)
      {
        universe[i]->adjust_position();
      }

    seconds = seconds + D_TIME;
  }

int main(int argc, char* argv[])
{
  std::string::size_type sz;

  const double TIME = std::stod(argv[1], &sz);
  const double D_TIME = std::stod(argv[2], &sz);

  double  number_of_planets, universe_radius;
  std::string str_number_of_planets, str_universe_radius;

  std::getline(std::cin, str_number_of_planets);
  number_of_planets = stod(str_number_of_planets, &sz);

  std::getline(std::cin, str_universe_radius);
  universe_radius = stod(str_universe_radius, &sz);
  universe_radius *= 2.4e-9;

  std::vector<Body*> universe;
  //set up vector
  for(int i = 0; i < number_of_planets; i++)
    {
      universe.push_back(new Body());
      universe[i]->set_universe_radius(universe_radius);
      universe[i]->set_d_time(D_TIME);
      std::cin >> *universe[i];
      universe[i]->set_sprite();
    }

  //create font
  sf::Font font;
   if (!font.loadFromFile("OpenSans-Regular.ttf"))
   {
     std::cout << "Could not load font" << std::endl;
   }
   sf::Text text;
   text.setFont(font);
   text.setCharacterSize((static_cast<int>(universe_radius)) / 25);
   text.setFillColor(sf::Color::White);

   //create background
   sf::Texture t_background;
   if(!t_background.loadFromFile("background_universe.png"))
     {
       std::cout << "Could not load background image";
     }
   sf::Sprite s_background(t_background);

  //display planets
  sf::RenderWindow window(sf::VideoMode(universe_radius, universe_radius),
"Universe");
   double time = 0;
    while (window.isOpen())
    {
  sf::Event event;

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

      while(time < TIME)
        {

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

          text.setString(std::to_string(time));

 window.clear();
        window.draw(s_background);
        window.draw(text);
        for(int i = 0; i < number_of_planets; i++)
              {
                  window.draw(*universe[i]);
              }
               window.display();
               step(time, universe, D_TIME);
               std::this_thread::sleep_for(std::chrono::milliseconds(25));
        }
      window.clear();
      window.close();
    }

    std::cout << number_of_planets << std::endl;
    std::cout << universe_radius * 2.4e9 << std::endl;

    for(int i = 0; i < universe.size(); i++)
      {
        std::cout << universe[i]->get_x() << "   " << universe[i]->get_y() << "   " << universe[i]->get_velocity_x() << "   ";

        std::cout << universe[i]->get_velocity_y() << "   " << universe[i]->get_mass() << "   " << universe[i]->get_file_name();

        std::cout << std::endl;
      }

  return 1;
}

```

Simulation Running with a Solar System Theme:


Activities
Universe
Fri 17:49
Learning Management Sy
Home Page – COMP 2040
https://lowell.umassonlin
PS3b(2).pdf
https://lowell.umassonline.net/bbcswebdav/pid-589808-dt-content-rid-8745524_1/courses/L2830-8238/ps3b-readme.txt
Universe
32050000.000000

```
/**********************************************************************
 *  N-Body Simulation ps3b-readme.txt template
 **********************************************************************/

Name:
Hours to complete assignment:

/**********************************************************************
 *  Briefly discuss the assignment itself and what you accomplished.
 **********************************************************************/


  /**********************************************************************
 *  Discuss one or more key algorithms, data structures, or
 *  OO designs that were central to the assignment.
 **********************************************************************/



/**********************************************************************
 *  Briefly explain the workings of the features you implemented.
 *  Describe if and how do you used the smart pointers
 *  Include code excerpts.
 **********************************************************************/



/**********************************************************************
 *  List whatever help (if any) you received from the instructor,
 *  classmates, or anyone else.
 **********************************************************************/

/**********************************************************************
 *  Describe any serious problems you encountered.
 **********************************************************************/


/**********************************************************************
 *  List any other comments here.
 **********************************************************************/
```

PS4 DNA Sequence Alignment

Discussion

For this assignment, we created a class that would use the Needleman-Wunsch algorithm to calculate the edit distance to match two character sequences. In addition, we calculated the time it took to match two DNA strands and analyzed the time and space performance.

The actual implementation used a matrix, which I simulated using a vector of vectors. I found the vector to be more useful than an array since I can use the member functions of a vector, whereas with an array I would be more limited. The algorithm finds close to an optimal solution using gaps where needed in matching.

The algorithm uses a 2d grid of length m by n, and the entire grid is filled out by the algorithm. This means that the space complexity is O(mn). Since filling out each cell is O(1), the time complexity is O(mn). The output of the program shows the matching results and the cost associated with the algorithm. The cost associated with matching two that are the same is 0, different is 1, and adding a gap is 2. The algorithm can match 2 strands of 2500 nucleotides in ecoli in 0.197733 seconds.

This assignment was useful for learning about implementing algorithms, as well as memory checking since I used valgrind, a memory analysis tool, to help me debug memory leaks and segmentation faults.

Matching part of two strands of ecoli:

```
GG0
AA0
TT0
GG0
TT0
TT0
GG0
CC0
C-2
G-2
C-2
T-2
T-2
T-2
T-2
```

Makefile:

```
CC = g++
CFLAGS= -std=c++11 -c -Wall
LDFLAGS=
SOURCES= ps4.cpp
OBJECTS=$(SOURCES:.cpp=.o)
LIBS= -lsfml-graphics -lsfml-window -lsfml-system
EXECUTABLE = ED

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
	$(CC) $(LDFLAGS) $(OBJECTS) -o $@ $(LIBS)

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

clean:
	rm ps4.o ps4.cpp~ Makefile~ ED

```

Ps4.cpp Source Code:

```
//Evan Mitchell

#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>

class ED
{
public:
  ED(std::string s1, std::string s2)
  {
    sequence_x = s1;
    sequence_y = s2;
  }

  int penalty(char a, char b) //calculates penalty of aligning a and b
  {
    if(a == b)
      {
        return 0;
      }
    return 1;
  }

  int min(int a, int b, int c)//return min of a, b, c
  {
    if(a <= b && a <= c)
      {
        return a;
      }
    if(b <= c && b <= a)
      {
        return b;
      }
    if(c <= a && c <= b)
      {
        return c;
      }
  }

    int OptDistance()
    {
      for(int i = 0; i < sequence_x.size() + 1; i++) //set up matrix
        {
          matrix.push_back(std::vector<int>());
```

```cpp
     matrix[i].resize(sequence_y.size() + 1);
    }

   matrix[matrix.size() - 1][matrix[0].size() - 1] = 0; //place zero at bottom right

   for(int i = matrix[matrix.size() - 1].size() - 2; i >= 0;  i--) //fill up bottom row
    {
     matrix[matrix.size() - 1][i] = matrix[matrix.size() - 1][i + 1] + 2;
    }

   for(int i = matrix.size() - 2; i >= 0; i--) //fill up rightmost column
    {
     matrix[i][matrix[i].size() - 1] = matrix[i + 1][matrix[i].size() - 1] + 2;
    }

   for(int i = matrix.size() - 2; i >= 0; i--) //fill up matrix
    {
     for(int j = matrix[i].size() - 2; j >= 0; j--)
      {
       matrix[i][j] = min(matrix[i + 1][j + 1] + penalty(sequence_x[i], sequence_y[j]), matrix[i + 1][j] + 2, matrix[i][j + 1] + 2);
      }
    }
   edit_distance = matrix[0][0];
   return matrix[0][0]; //return cost
  }

  std::string Alignment()
  {
   std::stringstream ss;
   int i = 0;
   int j = 0;
   while(i <= matrix.size() - 1 || j <= matrix[0].size() - 1) //while not at bottom right
    {
     if(i == matrix.size() - 1 && j == matrix[i].size() - 1) //case where bottom right
      {
       return ss.str();
      }
     else if(i == matrix.size() - 1) //case where bottom row is reached
      {
       while(j < matrix[i].size() - 1)
        {
         opt_x.push_back('-'); //y matched up with -
         opt_y.push_back(sequence_y[j]);

         ss << opt_x.back() << opt_y.back() << '2' << std::endl;
         j++;
        }
```

```cpp
            }
          else if(j == matrix[i].size() - 1)//case where rightmost column is reached
            {
              while(i < matrix.size() - 1)
                {
                  opt_x.push_back(sequence_x[i]); //x matched up with -
                  opt_y.push_back('-');

                  ss << opt_x.back() << opt_y.back() << '2' << std::endl;

                  i++;
                }
            }
          else if((matrix[i][j] == matrix[i+1][j+1]) && (sequence_x[i] == sequence_y[j])) //case where letters match
        {
          opt_x.push_back(sequence_x[i]);//matches the letters up
          opt_y.push_back(sequence_y[j]);

          ss << opt_x.back() << opt_y.back() << '0' << std::endl;
          i++;
          j++;
        }
          else if(matrix[i][j] == matrix[i+1][j+1] + 1)
            {
          opt_x.push_back(sequence_x[i]);//matches the letters up
          opt_y.push_back(sequence_y[j]);

          ss << opt_x.back() << opt_y.back() << '1' << std::endl;
          i++;
          j++;
            }
      else if(matrix[i][j] == matrix[i + 1][j] + 2)
        {
          opt_x.push_back(sequence_x[i]); //x matched up with -
          opt_y.push_back('-');

          ss << opt_x.back() << opt_y.back() << '2' << std::endl;

          i++;
        }
      else if(matrix[i][j] == matrix[i][j + 1] + 2)
        {
          opt_x.push_back('-'); //y matched up with -
          opt_y.push_back(sequence_y[j]);

          ss << opt_x.back() << opt_y.back() << '2' << std::endl;

          j++;
        }

        }
      return ss.str();
    }

private:
  std::string sequence_x, sequence_y, opt_x, opt_y; //sequence x is horizontal, y is vertical
  std::vector<std::vector<int>> matrix;
  int edit_distance;
};

int main()
{
  std::string s1;
  std::string s2;

  std::getline(std::cin, s1);
  std::getline(std::cin, s2);

  sf::Clock clock;
  sf::Time t;

  ED align(s1, s2);

  std::cout << "Edit distance = " << align.OptDistance() << std::endl << align.Alignment();

  t = clock.getElapsedTime();
  std::cout << "Execution time is " << t.asSeconds() << " seconds \n";

  return 1;
}

```

PS5 Ring Buffer and Guitar Hero

Part A

Discussion

For part a of this assignment, we were tasked with creating a ring buffer class which holds integer values. The class member functions are fully utilized in part b, and design is geared towards the class working for that purpose.

The purpose of the ring buffer is to hold different values that together represent a sound buffer. This assignment was useful in further solidifying my knowledge of object oriented design, in addition to using unit testing frameworks. I used the boost libraries testing framework to test my class.

Makefile:

```
CC = g++
CFLAGS= -std=c++11 -c -Wall
LDFLAGS=
SOURCES= RingBuffer.hpp RingBuffer.cpp test.cpp
OBJECTS=$(SOURCES:.cpp=.o)
LIBS= -lsfml-graphics -lsfml-window -lsfml-system -lboost_unit_test_framework
EXECUTABLE = ps5a

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
	$(CC) $(LDFLAGS) $(OBJECTS) -o $@ $(LIBS)

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

clean:
	rm ps5a RingBuffer.cpp~ RingBuffer.hpp~ RingBuffer.o test.cpp~ test.o
```

Test.cpp Source Code (Unit Testing):

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

#include <stdint.h>
#include <iostream>
#include <string>
#include <stdexcept>
#include <exception>
#include "RingBuffer.hpp"

BOOST_AUTO_TEST_CASE(RingBufferSizeTest)
{
  RingBuffer b(5); //ring buffer with capacity 5
  b.enqueue(1);
  b.enqueue(1);
  b.enqueue(1);

  BOOST_REQUIRE(b.size() == 3); //testing to make sure size works correctly
}

BOOST_AUTO_TEST_CASE(RingBufferPeekTest)
{
  RingBuffer b(3);
  BOOST_REQUIRE_THROW(b.peek(), std::runtime_error); //can't peek from empty buffer
  b.enqueue(4);
  BOOST_REQUIRE(b.peek() == 4); //front must be 4
}

 BOOST_AUTO_TEST_CASE(RingBufferIsEmptyandIsFullTest)
{
  RingBuffer b(2);
  BOOST_REQUIRE(b.isEmpty());
  b.enqueue(7);
  b.enqueue(8);
  BOOST_REQUIRE(b.isFull()); //testing for empty and full at appropriate times
}

BOOST_AUTO_TEST_CASE(RingBufferEnqueueandDequeueTest)
{
        RingBuffer b(10);
        b.enqueue(5);
        b.enqueue(3);
        BOOST_REQUIRE(b.dequeue() == 5);
        BOOST_REQUIRE(b.dequeue() == 3); //checking enqueue and dequeue
        BOOST_REQUIRE_THROW(b.dequeue(), std::runtime_error); //making sure it throws an err when empty
}

BOOST_AUTO_TEST_CASE(RingBufferConstructortTest)
{
 BOOST_REQUIRE_THROW(RingBuffer(0), std::exception);  //making sure it throws an exception for
 BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument); //invalid arguments
 BOOST_REQUIRE_NO_THROW(RingBuffer(50)); //constructor for 50 capacity
}

```

RingBuffer.hpp Source Code:

```
#include <iostream>
#include <stdint.h>
#include <vector>
using namespace std;

//RingBuffer class
class RingBuffer
{
public:
  RingBuffer(int capacity);
  int size();
  bool isEmpty();
  bool isFull();
  void enqueue(int16_t x);
  int16_t dequeue();
  int16_t peek();
private:
  vector<int16_t> buffer;
};
```

RingBuffer.cpp Source Code:

```
#include "RingBuffer.hpp"
#include <vector>
#include <iostream>
```

```cpp
#include <stdint.h>
#include <stdexcept>
using namespace std;

RingBuffer::RingBuffer(int capacity)
{
  if(capacity <= 0)
    {
      throw invalid_argument("RB constructor: capacity must be greater than zero");
    }
  buffer.reserve(capacity);
}

int RingBuffer::size()
{
  return buffer.size();
}

bool RingBuffer::isEmpty()
{
  if(buffer.empty())
    {
      return true;
    }
  else
    {
      return false;
    }
}

bool RingBuffer::isFull()
{
  if(buffer.size() == buffer.capacity())
    {
      return true;
    }
  else
    {
      return false;
    }
}

void RingBuffer::enqueue(int16_t x)
{
  if(this->isFull())
    {
      throw runtime_error("enqueue: can't enqueue to a full ring.");
    }
  buffer.push_back(x);
}

int16_t RingBuffer::dequeue()
{
  if(this->isEmpty())
    {
      throw runtime_error("dequeue: can't dequeue from an empty buffer.");
    }
  int16_t front = buffer.front();
  buffer.erase(buffer.begin());
  return front;
}

int16_t RingBuffer::peek()
{
  if(this->isEmpty())
    {
      throw runtime_error("peek: can't peek from an empty buffer.");
    }
  return buffer.front();
}
```

Part B

Discussion

For Part b of the assignment we used the ring buffer class to create a guitar simulation.

37 keys on the keyboard represent guitar strings with different frequencys. We utilized the Karplus-Strong algorithm to mimic the sound of a real life guitar string.

The way to create a sound with SFML is to use a sound buffer, which uses a vector of std::int16_ts to mimic a sound. I used a function which took a guitar string object as its parameter and returned a std::vector<std::int16_t>.

I implemented a guitar string class which used a pointer to a ring buffer as a private data member. The guitar string class has several member functions that encapsulate the behavior of a real life guitar string. Internal ring buffers are set up based on the frequency the object is supposed to represent. Since each key has an associated frequency, I used a map to map each key to its frequency, then I mapped each key to its corresponding sound. I used the map to play each sound when the associated key was pressed.

My knowledge of the copy constructor was useful for this assignment, since I used a function which took a guitar string object as a copy, and initially I did not know where the bug was coming from. Eventually I found out that the problem was shared data. I was able to fix the problem by passing the object by reference instead, since it was not used after the function call so it did not matter if it was changed.

This assignment was very useful for getting more experience with the copy constructor, designing and using my own classes, using maps, and implementing algorithms.

Makefile:

```
CC = g++
CFLAGS= -std=c++11 -c -Wall
LDFLAGS=
SOURCES= RingBuffer.hpp RingBuffer.cpp GuitarHero.cpp GuitarString.cpp GuitarString.hpp
OBJECTS=$(SOURCES:.cpp=.o)
LIBS= -lsfml-graphics -lsfml-window -lsfml-audio -lsfml-system -lboost_unit_test_framework
EXECUTABLE = GuitarHero

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
	$(CC) $(LDFLAGS) $(OBJECTS) -o $@ $(LIBS)

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

clean:
	rm GuitarHero RingBuffer.cpp~ RingBuffer.hpp~ RingBuffer.o GuitarHero.cpp~ GuitarHero.o ps5b-readme.txt~ GuitarString.o GuitarString.cpp~ GuitarString.hpp~ Makefile~

```

GuitarHero.cpp Source Code:

```cpp
#include <exception>
#include <stdexcept>
#include <stdint.h>
#include <time.h>

#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Audio.hpp>
#include <SFML/Window.hpp>
#include <vector>
#include <map>
#include <string>
#include <math.h>

#include "GuitarString.hpp"

const double CONC_A = 220.0;
const double SAMPLES_PER_SECOND = 44100;

std::vector<std::int16_t> make_samples(GuitarString& g)
{
  std::vector<std::int16_t> samples;
  g.pluck();
  for(int i = 0; i <  44100 * 8; i++)
    {
      g.tic();
      samples.push_back(g.sample());
    }
  return samples;
}

```

```
int main()
{
  srand(time(NULL));

  sf::RenderWindow window(sf::VideoMode(300, 200), "Guitar Hero");
  sf::Event event;

  std::string key_list("q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ");
  std::map <char, double> key_frequency_pairs;
  std::map <char, sf::Sound> key_sound_pairs;

  for(int i = 0; i < 37; i++)
    {
      key_frequency_pairs.insert(pair<char, double>(key_list[i], (CONC_A *
2) *
pow(2, (i - 24) / 12)));
    }

   std::vector<std::int16_t> samples;

   vector<sf::SoundBuffer*> buffer_ps;

   for(map<char, double>::iterator it = key_frequency_pairs.begin(); it !=
key_f
requency_pairs.end(); ++it)
    {
      GuitarString gs  = GuitarString(it->second);
      sf::Sound sound;
      sf::SoundBuffer sbuffer;
      samples = make_samples(gs);
      if(!sbuffer.loadFromSamples(&samples[0], samples.size(), 2,
SAMPLES_PER_SE
COND))
         throw std::runtime_error("sf::SoundBuffer: failed to load from
samples.
");
      buffer_ps.push_back(new sf::SoundBuffer(sbuffer));
      sound.setBuffer(*(buffer_ps.back()));
      key_sound_pairs.insert(pair<char, sf::Sound>(it->first, sound));
    }

    while (window.isOpen())
      {
    while (window.pollEvent(event))
      {
      switch (event.type)
        {
        case sf::Event::Closed:
          {
            window.close();
            break;
          }
        case sf::Event::TextEntered:
          {
                for(int i = 0; i < 37; i++)
                  {
                    if(static_cast<char>(event.text.unicode) ==
key_list[i])
                      {
key_sound_pairs.lower_bound(static_cast<char>(event.text
.unicode))->second.play();
                      }
                  }

            break;
          }

           default:
           break;

      }

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

  return 1;
  }


```

GuitarString.hpp:

```
#include "RingBuffer.hpp"
#include <stdint.h>
#include <vector>

class GuitarString
{
public:
  GuitarString(double frequency);
  GuitarString(std::vector<std::int16_t> init);
  void pluck();
  void tic();
  std::int16_t sample();
  int time();
  ~GuitarString();
  //GuitarString(const GuitarString& old_gs);
private:
  RingBuffer* buffer;
  int _time;
};

```

GuitarString.cpp:

```
#include "GuitarString.hpp"
#include <stdint.h>
```

```
#include <vector>
#include <math.h>

const double DECAY_FACTOR = .996;

  GuitarString::GuitarString(double frequency)
  {
    int N = ceil(44100 / frequency);
     buffer = new RingBuffer(N);
    _time = 0;
  }

/*GuitarString::GuitarString(const GuitarString& old_gs)
{
  _time = old_gs._time;
  buffer = new RingBuffer(old_gs.buffer);
}*/

GuitarString::GuitarString(std::vector<std::int16_t> init)
  {
    buffer = new RingBuffer(init.size());
    for(int i = 0; i < init.size(); i++)
      {
        buffer->enqueue(init[i]);
      }
  }
  void  GuitarString::pluck()
  {

    while(!buffer->isEmpty())
      {
        buffer->dequeue();
      }

    for(int i = 0; i < buffer->capacity(); i++)
      {
        std::int16_t rand_val = static_cast<std::int16_t>(rand() % 32766);
        if(static_cast<int>(rand()) % 2)
          {
            rand_val *= -1;
          }

        buffer->enqueue(rand_val);

      }

  }
  void GuitarString::tic()
  {
    std::int16_t first = buffer->dequeue();
    std::int16_t second = buffer->peek();
    buffer->enqueue(((first + second)/2) * DECAY_FACTOR);
    _time++;
  }

  std::int16_t GuitarString::sample()
  {
```

```
  return buffer->peek();
}

int GuitarString::time()
{
  return _time;
}

GuitarString::~GuitarString()
{
  delete buffer;
}

```

PS6 Airport

Discussion

For this assignment we coordinated airplanes landing on airport runways with mutexes and conditional variables. Certain runways could not be used at the same time as other runways, so when one plane lands on a runway, all the other planes must know not to land on that runway until the other plane takes off. Each airplane was represented by a thread.

Unfortunately, I was not able to complete this assignment. It is the only of the seven assignment that is not working. When the program runs, it results in two planes landing on the same runway and the program terminating. Despite not being able to get it working, I learned a lot about conditional variables and mutexes.

My idea was to have a Boolean for each runway being ready, initially set to true. In addition, there would be a conditional variable for each runway. When a plane requests to land on a runway, if that runway is not ready based on the Boolean, it would wait on a conditional variable that the plane on that runway (or on a runway that blocks that one) will notify.

Makefile:

```
CC = g++
CFLAGS = -c -g -Og -std=c++11
OBJ = Airplane.o Airport.o AirportRunways.o AirportServer.o
DEPS =
LIBS = -pthread
EXE = Airport-NoSync

```

```
all: $(OBJ)
	$(CC) $(OBJ) -o $(EXE) $(LIBS)

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

clean:
	rm -f $(OBJ) $(EXE)
```

Airport.cpp

```
/**
*  Airport driver program
*/

#include <iostream>
#include <thread>
#include <vector>

#include "AirportServer.h"
#include "AirportRunways.h"
#include "Airplane.h"

using namespace std;


void run(Airplane* ap)
{
	ap->land();

} // end run


int main(void)
{
	AirportServer as;

	vector<thread> apths; // Airplane threads

	                                                    // Create and launch the
individual
      Airplane threads
	for (int i = 1; i <= AirportRunways::NUM_AIRPLANES; i++)
	{
		Airplane* ap = new Airplane(i, &as);

		apths.push_back(thread(&run, ap));
	}

	// Wait for all Airplane threads to terminate (shouldn't happen!)
	for (auto& th : apths)
	{
		th.join();
	}

```

```
        return 0;

} // end main

```

AirportServer.cpp:

```
#include <iostream>
#include <thread>
#include <condition_variable>

#include "AirportServer.h"


/**
*  Called by an Airplane when it wishes to land on a runway
*/
void AirportServer::reserveRunway(int airplaneNum,
AirportRunways::RunwayNumber
runway)
{
        // Acquire runway(s)
        {  // Begin critical region

                unique_lock<mutex> runwaysLock(runwaysMutex);

                {
                        lock_guard<mutex> lk(AirportRunways::checkMutex);

                        cout << "Airplane #" << airplaneNum << " is
acquiring an
y needed runway(s) for landing on Runway "
                                 << AirportRunways::runwayName(runway) <<
endl;
                }

                /**
                 *  ***** Add your synchronization here! *****
                 */

                switch(runway)
                  {
                    case AirportRunways::RUNWAY_4L:
                    {
                      while(!r_4l_rdy) r_4l_cv.wait(r_4l_lck);
                      r_4l_rdy = false;
                      r_15l_rdy = false;
                      r_15r_rdy = false;
                            break;
                    }
                     case AirportRunways::RUNWAY_4R:
                    {

                      while(!r_4r_rdy) r_4r_cv.wait(r_4r_lck);
                      r_4r_rdy = false;
```

```
                         r_15l_rdy = false;
                         r_9_rdy = false;
                         r_15r_rdy = false;
                            break;
                       }
                        case AirportRunways::RUNWAY_9:
                       {

                          if(!r_9_rdy) r_9_cv.wait(r_9_lck);
                         r_9_rdy = false;
                         r_4r_rdy = false;
                         r_15r_rdy = false;
                            break;
                       }
                        case AirportRunways::RUNWAY_14:
                       {

                         while(!r_14_rdy) r_14_cv.wait(r_14_lck);
                         r_14_rdy = false;
                            break;
                       }
                        case AirportRunways::RUNWAY_15L:
                       {

                         while(!r_15l_rdy) r_15l_cv.wait(r_15l_lck);
                         r_15l_rdy = false;
                         r_4r_rdy = false;
                         r_4l_rdy = false;
                            break;
                       }
                     case AirportRunways::RUNWAY_15R:
                       {

                         while(!r_15r_rdy) r_15r_cv.wait(r_15r_lck);
                         r_15r_rdy = false;
                         r_9_rdy = false;
                         r_4l_rdy = false;
                         r_4r_rdy = false;
                            break;
                       }
                     default:
                       cout << "ERROR IN SWITCH..." << endl;
                       exit(1);

                     }

                   // Check status of the airport for any rule violations
                   AirportRunways::checkAirportStatus(runway);

                   runwaysLock.unlock();

           } // End critical region

           // obtain a seed from the system clock:
           unsigned seed =
std::chrono::system_clock::now().time_since_epoch().coun
t();
```

```
        std::default_random_engine generator(seed);

        // Taxi for a random number of milliseconds
        std::uniform_int_distribution<int> taxiTimeDistribution(1,
MAX_TAXI_TIME
);
        int taxiTime = taxiTimeDistribution(generator);

        {
                lock_guard<mutex> lk(AirportRunways::checkMutex);

                cout << "Airplane #" << airplaneNum << " is taxiing on
Runway "
<< AirportRunways::runwayName(runway)
                                << " for " << taxiTime << " milliseconds\n";
        }

        std::this_thread::sleep_for(std::chrono::milliseconds(taxiTime));

} // end AirportServer::reserveRunway()


/**
 *  Called by an Airplane when it is finished landing
 */
void AirportServer::releaseRunway(int airplaneNum,
AirportRunways::RunwayNumber
runway)
{
        // Release the landing runway and any other needed runways
        { // Begin critical region

                unique_lock<mutex> runwaysLock(runwaysMutex);

                {
                        lock_guard<mutex> lk(AirportRunways::checkMutex);

                        cout << "Airplane #" << airplaneNum << " is
releasing an
y needed runway(s) after landing on Runway "
                                 << AirportRunways::runwayName(runway) <<
endl;
                }

                /**
                *  ***** Add your synchronization here! *****
                */

                  switch(runway)
                  {
                    case AirportRunways::RUNWAY_4L:
                    {
                      r_4l_rdy = true;
                      r_15l_rdy = true;
                      r_15r_rdy = true;
                        AirportRunways::finishedWithRunway(runway);
                        //runwaysLock.unlock();
```

```
          r_4l_cv.notify_all();
          r_15l_cv.notify_all();
           r_15r_cv.notify_all();
            break;
       }
        case AirportRunways::RUNWAY_4R:
       {

         r_4r_rdy = true;
         r_15l_rdy = true;
         r_9_rdy = true;
         r_15r_rdy = true;
           AirportRunways::finishedWithRunway(runway);
           //runwaysLock.unlock();
          r_4r_cv.notify_all();
           r_15l_cv.notify_all();
            r_9_cv.notify_all();
             r_15r_cv.notify_all();
            break;
       }
        case AirportRunways::RUNWAY_9:
       {

         r_4r_rdy = true;
         r_15r_rdy = true;
         r_9_rdy = true;
           AirportRunways::finishedWithRunway(runway);
           //runwaysLock.unlock();
          r_9_cv.notify_all();
           r_15r_cv.notify_all();
            r_4r_cv.notify_all();
            break;
       }
        case AirportRunways::RUNWAY_14:
       {

        r_14_rdy = true;
           AirportRunways::finishedWithRunway(runway);
           //runwaysLock.unlock();
         r_14_cv.notify_all();
            break;
       }
        case AirportRunways::RUNWAY_15L:
       {

         r_15l_rdy = true;
         r_4r_rdy = true;
         r_4l_rdy = true;

           AirportRunways::finishedWithRunway(runway);

           //runwaysLock.unlock();
           r_4r_cv.notify_all();
           r_4l_cv.notify_all();
         r_15l_cv.notify_all();
            break;
       }
```

```
                     case AirportRunways::RUNWAY_15R:
                    {
                      //unique_lock<mutex> r_15r_lck(r_15r_mtx);
                      r_15r_rdy = true;
                      r_9_rdy = true;
                      r_4l_rdy = true;
                      r_4r_rdy = true;

                        AirportRunways::finishedWithRunway(runway);

                      r_9_cv.notify_one();
                       r_4l_cv.notify_one();
                        r_4r_cv.notify_one();
                      r_15r_cv.notify_one();
                         break;
                    }
                     default:
                    cout << "ERROR IN SWITCH..." << endl;
                    exit(1);

                  }



                // Update the status of the airport to indicate that the
landingis complete


        } // End critical region

        // obtain a seed from the system clock:
        unsigned seed =
std::chrono::system_clock::now().time_since_epoch().count();
        std::default_random_engine generator(seed);

        // Wait for a random number of milliseconds before requesting the
next l
anding for this Airplane
        std::uniform_int_distribution<int> waitTimeDistribution(1,
MAX_WAIT_TIME
);
        int waitTime = waitTimeDistribution(generator);

        {
                lock_guard<mutex> lk(AirportRunways::checkMutex);

                cout << "Airplane #" << airplaneNum << " is waiting for "
<< wai
tTime << " milliseconds before landing again\n";
        }

        std::this_thread::sleep_for(std::chrono::milliseconds(waitTime));

} // end AirportServer::releaseRunway()

```

PS7 Kronos Intouch Parsing

Discussion

For the final programming assignment of the semester, we used regular expressions to parse a file and calculate boot up time of a Kronos Intouch system. The assignment also made use of boost libraries regular expression functionalities. We read input in from five Intouch log files and determined whether or not the boot up was successful, and if it was how long the bitt up time was based on the timestamp.

Every time the system began a boot up, the message: "(log.c.166) server started" was written to the log, and when the bootup was finished a message was written as well. The finish bootup message was: "oejs.AbstractConnector:Started SelectChannelConnector". I used a regex object and the boost::regex_match() function to determine the starting and finishing lines for a boot up. If two boot up messages were received in a row, the result is a failed boot up since it attempted a new boot up before finishing.

In order to match any line containing those expressions, the regular expressions had to have .* before and after to match 0 or more of any characters except for new line. Since I was parsing the file using getline(), the fact that it did not match a new line did not matter. In addition, I had to use put the parentheses for the start up message regex in square brackets so that they would be taken as the actual character and not their special meaning. So the final regular expressions were as follows:

Boot up begin: .*[(]log.c.166[)] server started.*

Boot up end: .*oejs.AbstractConnector:Started SelectChannelConnector.*

The time calculation was achieved using a function I wrote that took two strings and returned a string. The return value was the time in the second string minus the time in the first string, resulting in time elapsed between the two strings. The timestamps were at the beginning of each line each, so I passed the two strings for the boot up and end boot up messages once each boot up was complete. Originally I wrote a driver program to test my function, then once I had it working I used it in my implementation.

Prior to this assignment, I hadn't had much experience with regular expressions, so this helped me get acquainted with how the work. It also helped me get more experience with working with files, and also program testing and design.

Makefile:

```
CC = g++
CFLAGS= -std=c++11 -c -Wall
```

```
LDFLAGS=
SOURCES= ps7.cpp
OBJECTS=$(SOURCES:.cpp=.o)
LIBS= -lboost_regex
EXECUTABLE = ps7

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
        $(CC) $(LDFLAGS) $(OBJECTS) -o $@ $(LIBS)

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

clean:
        rm ps7 Makefile~ ps7.o ps7.cpp~
```

Ps7.cpp Source Code:

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

using namespace std;
using namespace boost;

string timecalc(string time1, string time2); //take time1 and subtract time2 time format: 2013-11-01 12:28:32

int main(int arg, char* argv[])
{
  string inputname(argv[1]);
  string outputname(argv[1]);
  outputname.append(".rpt");
  ifstream input(inputname);
  ofstream output(outputname);
  string line;
  regex bootup(".*[(]log.c.166[)] server started.*");
  regex endbootup(".*oejs.AbstractConnector:Started SelectChannelConnector.*");
  bool lastfoundbootup = false;
  string bootupline;
  if(input.is_open())
    {
      output  << "Device Boot Report\n\n" << "InTouch log file: " << inputname << "\n\n";
      while(getline(input, line)) //reading in lines from input log
        {
          //cout << line << endl;
          if(regex_match(line, bootup) && !lastfoundbootup) //if match and not two bootups in a row
```

```cpp
            {
              output << "==Device boot==\n";
              //cout << "==Device boot==\n";
              output << line << " Boot Started\n";
              bootupline = line;
               lastfoundbootup = true;
            }
           if(regex_match(line, bootup) && lastfoundbootup)
            {
              output << "**** Incomplete boot ****\n\n";
              //cout << "**** Incomplete boot ****\n\n\n\n";
              output << "==Device boot==\n";
              output << line << " Boot Started\n";
              lastfoundbootup = true;
              bootupline = line;
            }
           if(regex_match(line, endbootup) && lastfoundbootup)
            {
              output << line << " Boot ended\n";
              //cout << line << " Boot ended\n";
              lastfoundbootup = false;
              //cout << bootupline << endl << line << endl << endl;
              output << "Time elapsed: "<< timecalc(bootupline, line ) << "\n\n"
 ;

              }
        }
    }
  cout << "start up diagnostics now in file named: " << outputname << endl;
  return 1;
}

string timecalc(string time1, string time2) //take time1 and subtract time2 time
 format: 2013-11-01 12:28:32
{
  string year1;
  year1.push_back(time1[0]);
  year1.push_back(time1[1]);
  year1.push_back(time1[2]);
  year1.push_back(time1[3]);
  string day1;
  day1.push_back(time1[5]);
  day1.push_back(time1[6]);
  string month1;
  month1.push_back(time1[8]);
  month1.push_back(time1[9]);

  string hour1;
  hour1.push_back(time1[11]);
  hour1.push_back(time1[12]);
  string minute1;
  minute1.push_back(time1[14]);
  minute1.push_back(time1[15]);
  string second1;
  second1.push_back(time1[17]);
```

```
    second1.push_back(time1[18]);

    gregorian::date d1(stoi(year1), stoi(day1), stoi(month1));
    posix_time::ptime t1(d1, posix_time::hours(stoi(hour1)) +
posix_time::minutes(
 stoi(minute1)) + posix_time::seconds(stoi(second1)));


    string year2;
    year2.push_back(time2[0]);
    year2.push_back(time2[1]);
    year2.push_back(time2[2]);
    year2.push_back(time2[3]);
    string day2;
    day2.push_back(time2[5]);
    day2.push_back(time2[6]);
    string month2;
    month2.push_back(time2[8]);
    month2.push_back(time2[9]);

    string hour2;
    hour2.push_back(time2[11]);
    hour2.push_back(time2[12]);
    string minute2;
    minute2.push_back(time2[14]);
    minute2.push_back(time2[15]);
    string second2;
    second2.push_back(time2[17]);
    second2.push_back(time2[18]);

    gregorian::date d2(stoi(year2), stoi(day2), stoi(month2));
    posix_time::ptime t2(d2, posix_time::hours(stoi(hour2)) +
posix_time::minutes(
stoi(minute2)) + posix_time::seconds(stoi(second2)));

    posix_time::time_duration td = t2 - t1;

    return posix_time::to_simple_string(td);
}

```

Sample of resulting file from running code on an Intouch log:

Device Boot Report

InTouch log file: device1_intouch.log

==Device boot==

```
2014-03-25 19:11:59: (log.c.166) server started  Boot Started

**** Incomplete boot ****


==Device boot==

2014-03-25 19:11:59: (log.c.166) server started  Boot Started

2014-03-25 19:15:02.369:INFO:oejs.AbstractConnector:Started SelectChannelConnect

or@0.0.0.0:9080 Boot ended

Time elapsed: 00:03:03


==Device boot==

2014-03-25 19:29:59: (log.c.166) server started  Boot Started

**** Incomplete boot ****


==Device boot==

2014-03-25 19:29:59: (log.c.166) server started  Boot Started

2014-03-25 19:32:44.036:INFO:oejs.AbstractConnector:Started SelectChannelConnect

or@0.0.0.0:9080 Boot ended

Time elapsed: 00:02:45
```