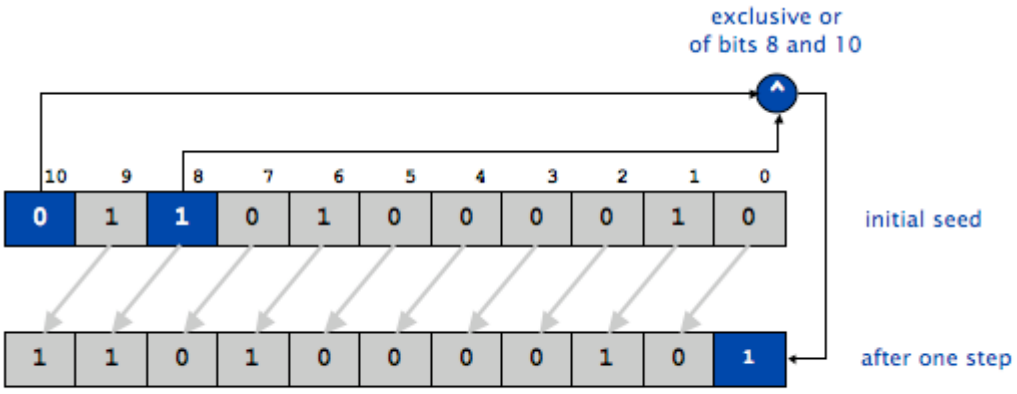
PS2a – Linear Feedback Shift and Unit Testing

The purpose of this assignment was to produce pseudo-random bits by simulating a linear feedback shift register (or LFSR) and then use it to implement a form of encryption for digital pictures (which will be done in PS2b).

PS2a focused on implementing the LFSR class and implementing unit test using the Boost test framework.

Implementing the LFSR class involved knowing what strings are. They are an array of characters so the first step I took towards implementing LFSR was to imagine a character array and what was happening to that array.



One step of an 11-bit LFSR with initial seed 01101000010 and tap at position 8

LFSR from reviewing the image is:

- Shifting the bits one position to the left and
- Replacing the vacated bit by the exclusive or of the bit shifted off and the bit previously at a given tap position in the register

So with this image, it just took messing with certain characters in the array. It took a little trail and error before I got the tap in the right place. I also didn't know how to turn a integer into a string so I did some research and found that the function:

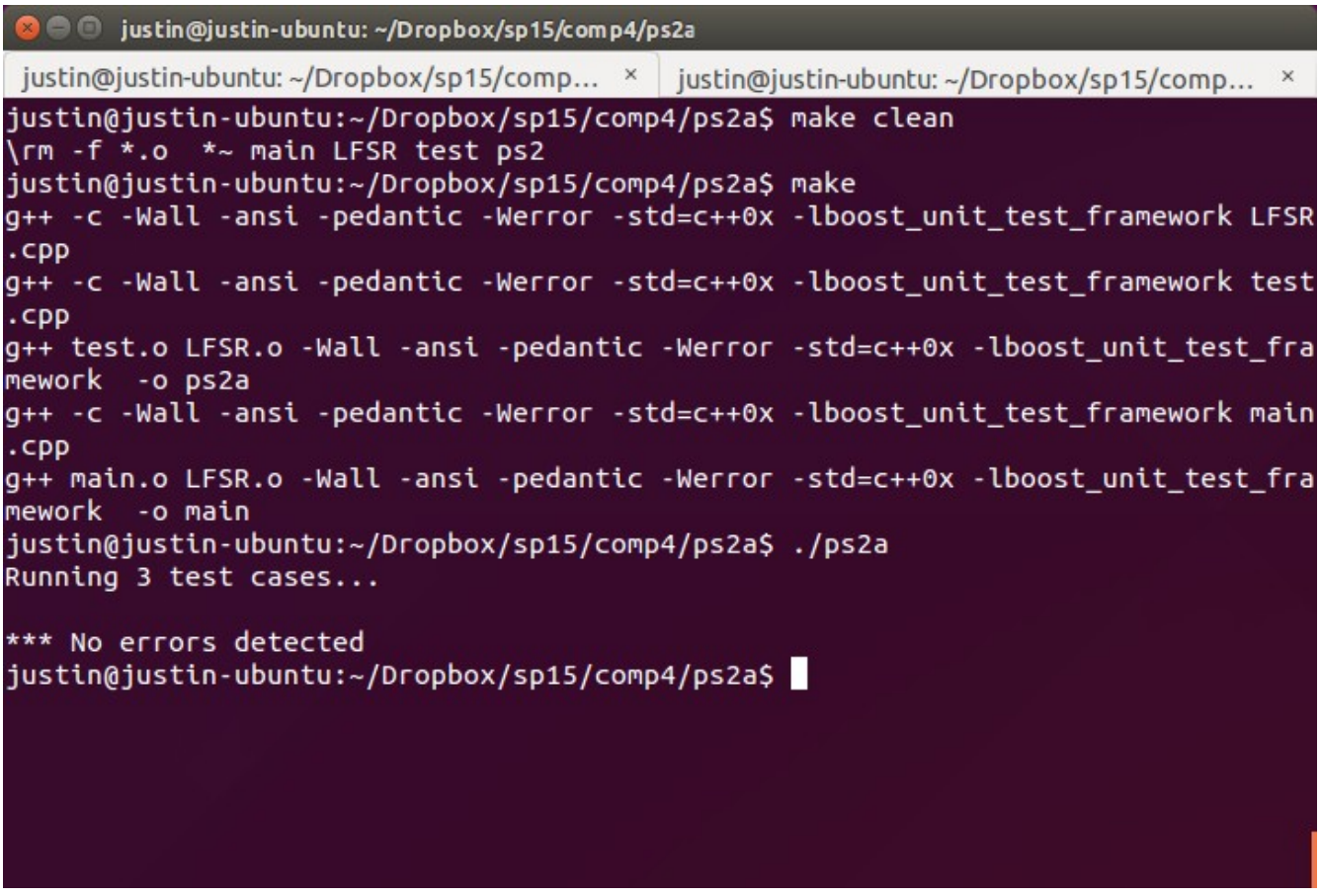
```
std::to_string(int integer);
```

worked perfectly as seen at line 19.

The matter next was unit testing my implementation using Boost. Before knowing what unit testing was, I used to debug my program by having simple test cases set throughout the program. I found out quickly, that unit testing was a much faster way of of debugging my program since I could test multiple cases at once and fix my program from there.

```
11 BOOST_AUTO_TEST_CASE(fiveBitsTapAtTwo) {
12   LFSR l("00111", 2);
13   BOOST_REQUIRE(l.step() == 1);
14   BOOST_REQUIRE(l.step() == 1);
15   BOOST_REQUIRE(l.step() == 0);
16   BOOST_REQUIRE(l.step() == 0);
17   BOOST_REQUIRE(l.step() == 0);
18   BOOST_REQUIRE(l.step() == 1);
19   BOOST_REQUIRE(l.step() == 1);
20   BOOST_REQUIRE(l.step() == 0);
21
22   LFSR l2("00111", 2);
23   BOOST_REQUIRE(l2.generate(8) == 198);
24 }
```

Unit test from test.cpp testing the constructor and the functions step() and generate(int k). The rest of the tests can be seen in the test.cpp code.



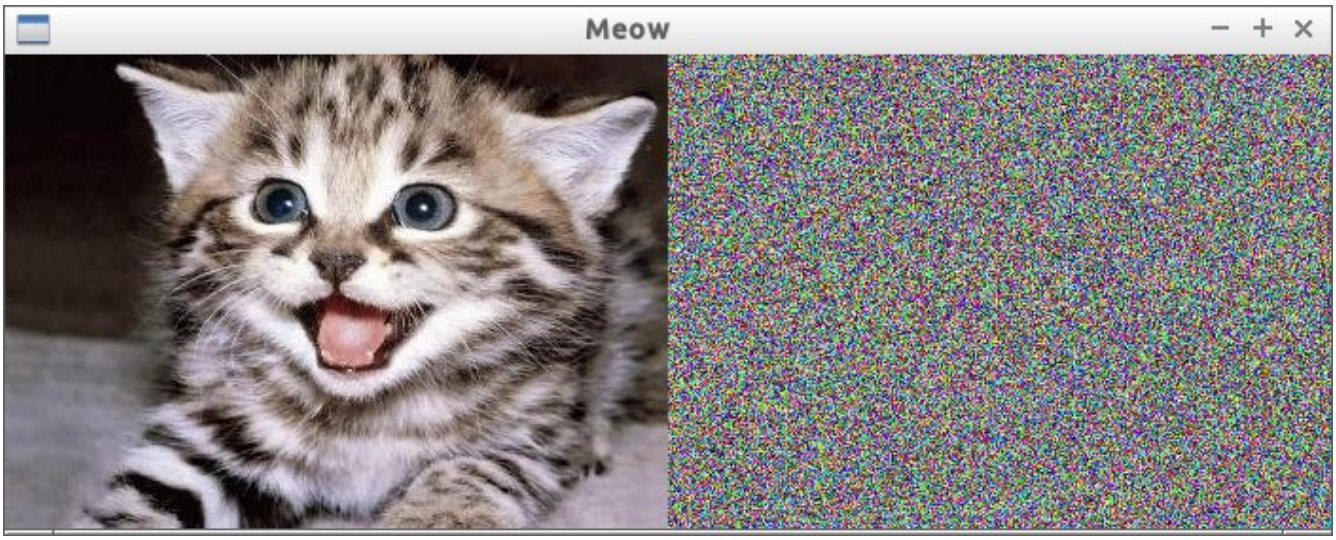
When the executable is linked with the unit tests and ran, it should detect no errors if my code worked correctly.

PS2b – Encoding and Decoding Images with LFSR

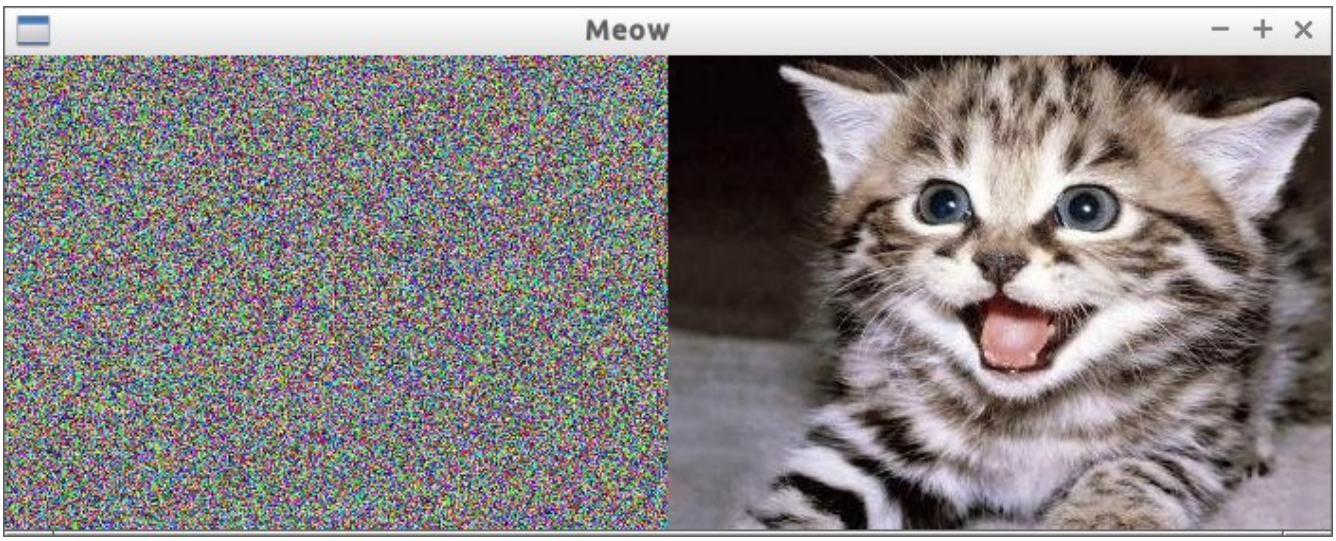
In this part of the assignment, the goal was to use the LFSR program to encrypt and decrypt pictures. The program was to have a transform() function that takes in a picture and an LFSR as arguments and return a new picture that extracted the red, green and blue components of the colors and xor the colors into a newly-generated 8-bit integer.

```
11  sf::Image transform(sf::Image picture, LFSR lfsr) {
12    sf::Vector2u size = picture.getSize();
13    sf::Color p2;
14    for (size_t x = 0; x < size.x; x++) {
15      for (size_t y = 0; y < size.y; y++) {
16        p2 = picture.getPixel(x, y);
17        p2.r = p2.r ^ lfsr.generate(8);
18        p2.g = p2.g ^ lfsr.generate(8);
19        p2.b = p2.b ^ lfsr.generate(8);
20        picture.setPixel(x, y, p2);
21      }
22    }
23    return picture;
24  }
```

The transform() function that PS2b focused on.



When the program is ran on an image output is produced on the left.
The interesting thing about this program is that it can decode the same image into the original!



```
    1: CC = g++
    2: OFLAGS = -c -Wall -ansi -pedantic -Werror -std=c++0x -lboost_unit_test_frame
work
    3: CFLAGS = -Wall -ansi -pedantic -Werror -std=c++0x -lboost_unit_test_framewor
k
    4: LFLAGS = -lsfml-window -lsfml-graphics -lsfml-system
    5:
    6: all:    ps2b
    7:
    8: ps2b: LFSR.o PhotoMagic.o
    9:         $(CC) PhotoMagic.o LFSR.o $(CFLAGS) $(LFLAGS)  -o PhotoMagic
   10:
   11:
   12: PhotoMagic.o: PhotoMagic.cpp
   13:         $(CC) $(OFLAGS) $(LFLAGS) PhotoMagic.cpp
   14:
   15: LFSR.o: LFSR.cpp LFSR.hpp
   16:         $(CC) $(OFLAGS) LFSR.cpp
   17:
   18:
   19: clean:
   20:         \rm -f *.o  *~ PhotoMagic ps2b
```

```
 1: // Copyright [2015] Justin Nguyen
 2:
 3: #include <SFML/System.hpp>
 4: #include <SFML/Window.hpp>
 5: #include <SFML/Graphics.hpp>
 6: #include <iostream>
 7: #include <string>
 8: #include <algorithm>
 9: #include "LFSR.hpp"
10:
11: sf::Image transform(sf::Image picture, LFSR lfsr) {
12:   sf::Vector2u size = picture.getSize();
13:   sf::Color p2;
14:   for (size_t x = 0; x < size.x; x++) {
15:     for (size_t y = 0; y < size.y; y++) {
16:       p2 = picture.getPixel(x, y);
17:       p2.r = p2.r ^ lfsr.generate(8);
18:       p2.g = p2.g ^ lfsr.generate(8);
19:       p2.b = p2.b ^ lfsr.generate(8);
20:       picture.setPixel(x, y, p2);
21:     }
22:   }
23:   return picture;
24: }
25:
26: int main(int argc, char *argv[]) {
27:   std::string fi;
28:   if (argc < 3) {
29:     std::cout << "Usage: " << argv[0] << "inputfile LFSR bit" << std::endl;
30:     exit(0);
31:   } else {
32:     fi = argv[1];
33:   }
34:   sf::Image image;
35:   if (!image.loadFromFile(fi))
36:     return -1;
37:   sf::Image imageOut;
38:   if (!imageOut.loadFromFile(fi))
39:     return -1;
40:   LFSR lfsr(argv[2], atoi(argv[3]));
41:   imageOut = transform(imageOut, lfsr);
42:   sf::Vector2u size = image.getSize();
43:   sf::RenderWindow window(sf::VideoMode(size.x * 2, size.y), "Meow");
44:   if (!imageOut.saveToFile(fi))
45:     return -1;
46:   sf::Texture texture;
47:   texture.loadFromImage(image);
48:   sf::Sprite sprite;
49:   sprite.setTexture(texture);
50:   sf::Texture textureOut;
51:   textureOut.loadFromImage(imageOut);
52:   sf::Sprite spriteOut;
53:   spriteOut.setTexture(textureOut);
54:   spriteOut.setPosition(size.x, 0);
55:   while (window.isOpen()) {
56:     sf::Event event;
57:     while (window.pollEvent(event)) {
58:       if (event.type == sf::Event::Closed)
59:         window.close();
60:     }
61:     window.clear(sf::Color::White);
```

```
   62:     window.draw(sprite);
   63:     window.draw(spriteOut);
   64:     window.display();
   65:   }
   66:   if (!imageOut.saveToFile("cat-out.bmp"))
   67:     return -1;
   68:   return 0;
   69: }
```

```
 1: // Copyright [2015] Justin Nguyen
 2:
 3: #ifndef LFSR_HPP_
 4: #define LFSR_HPP_
 5:
 6: #include <iostream>
 7: #include <string>
 8:
 9: class LFSR {
10:  private:
11:         int length;
12:         int tap;
13:         std::string _data;
14:
15:  public:
16:         LFSR(std::string seed, int  t);
17:         int step(); // ensures either 0 or 1.  //steps the register once.
18:         int generate(int k); // steps the register k times
19:         friend std::ostream& operator << (std::ostream& out, LFSR& lfsr);
20: };
21:
22: #endif
```

```
 1: // Copyright [2015] Justin Nguyen
 2:
 3: #include <iostream>
 4: #include <cmath>
 5: #include "LFSR.hpp"
 6:
 7: LFSR::LFSR(std::string seed, int t) {
 8:   _data = seed;
 9:   length =  _data.size();
10:   tap = t;
11: }
12:
13: int LFSR::step() {
14:   int bit;
15:   int length = _data.size();
16:   std::string s_bit;
17:   bit = _data.front() ^ _data[length -  tap - 1];
18:   s_bit = std::to_string(bit);
19:   _data.erase(0, 1);
20:   _data = _data + s_bit;
21:   return bit;
22: }
23:
24: int LFSR::generate(int k) {
25:   int count = 0;
26:   for (int i = k - 1; i >= 0; i--) {
27:     if (step() == 1)
28:       count +=  pow(2, i);
29:   }
30:   return count;
31: }
32:
33: std::ostream& operator << (std::ostream& out, LFSR& lfsr) {
34:   out << lfsr._data;
35:   return out;
36: }
```