

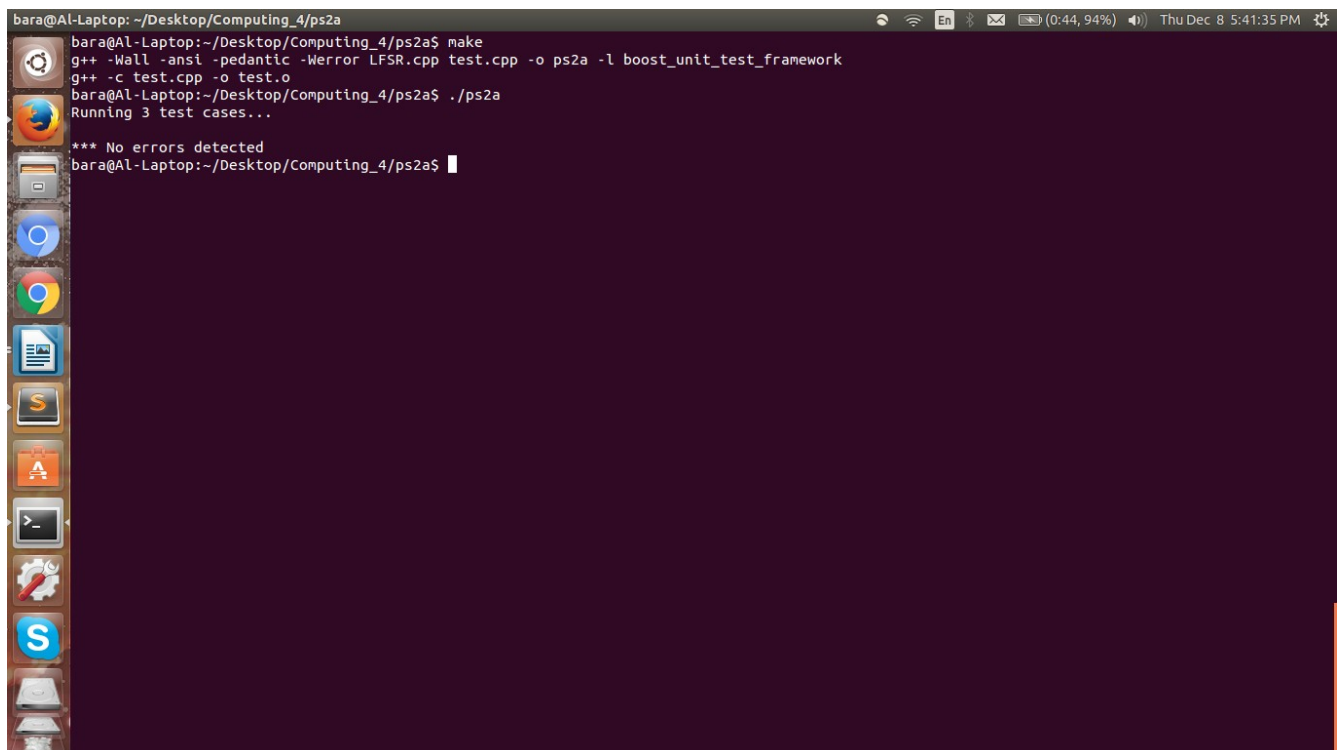
PS2: Linear Feedback Shift Register and Image Encoding

PS2a:

This assignment was to learn to register bits that shifts bits one position to the left and replaces the vacated bit by the exclusive or of the bit shifted off and the bit previously at a given tap position in the register. To test our LFSR is working, we needed to use software called boost.

I used this a class that accepts the seed and number of taps for the constructor. The step function would LFSR only once and would return the bit to test in my boost file. The generate function was essentially call the step function but would also give a condition.

I learned about what LFSR was. One big thing I learned is boost. I learned how create test cases having different conditions to test out the step and generate functions. Though, I currently can't find any use for it in my own projects.



```
bara@Al-Laptop: ~/Desktop/Computing_4/ps2a
bara@Al-Laptop:~/Desktop/Computing_4/ps2a$ make
g++ -Wall -ansi -pedantic -Werror LFSR.cpp test.cpp -o ps2a -l boost_unit_test_framework
g++ -c test.cpp -o test.o
bara@Al-Laptop:~/Desktop/Computing_4/ps2a$ ./ps2a
Running 3 test cases...

*** No errors detected
bara@Al-Laptop:~/Desktop/Computing_4/ps2a$
```

The image shows a terminal window on a Linux system. The user is in the directory ~/Desktop/Computing_4/ps2a. They run 'make' to compile the code, which produces 'ps2a' and 'test.o'. Then they run './ps2a', which outputs 'Running 3 test cases...' and '*** No errors detected'. The terminal has a dark purple background and a light blue prompt. The window title is 'bara@Al-Laptop: ~/Desktop/Computing_4/ps2a'. The system tray at the top right shows the date and time as 'Thu Dec 8 5:41:35 PM' and a battery level of 94%.

```
1: cc = g++
2: #all
3:
4: all : LFSR test
5:
6: LFSR : LFSR.o test.o
7:      $(cc) -Wall -ansi -pedantic -Werror LFSR.cpp test.cpp -o ps2a -l boo
st_unit_test_framework
8:
9: LFSR.o : LFSR.hpp
10:      $(cc) -c LFSR.cpp -o LFSR.o
11:
12: test : test.cpp LFSR.hpp
13:      $(cc) -c test.cpp -o test.o
14:
15:
16: clean:
17:      rm *.o ps2a
18:
19: run:
20:      ./ps2a
21:
22: debug: cc += -g
23: debug: ps2a
24:
```

```
1:
2: #include "LFSR.hpp"
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(fiveBitsTapAtTwo) {
9:     //steps through the function 8 times and checks to see the return bit is 1
or 0.
10:    LFSR l("00111", 2);
11:    BOOST_REQUIRE(l.step() == 1);
12:    BOOST_REQUIRE(l.step() == 1);
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:
20:    //Doing the math, it turns out to 198
21:    LFSR l2("00111", 2);
22:    BOOST_REQUIRE(l2.generate(8) == 198);
23:
24: }
25:
26: BOOST_AUTO_TEST_CASE(Test1){
27:     // I stepped 5 times by doing the xor to the 2nd tap.
28:    LFSR l3("0110110", 2);
29:    BOOST_REQUIRE(l3.step() == 1);
30:    BOOST_REQUIRE(l3.step() == 0);
31:    BOOST_REQUIRE(l3.step() == 1);
32:    BOOST_REQUIRE(l3.step() == 1);
33:    BOOST_REQUIRE(l3.step() == 1);
34:
35: }
36: BOOST_AUTO_TEST_CASE(Test2){
37:     //goes through step 3 times and by doing it out by hand, the answer is 7
38:     //It steps 3 times.
39:    LFSR l4("0110001", 3);
40:    BOOST_REQUIRE(l4.generate(4) == 7);
41: }
42:
43:
```

```
1:
2:
3: #include <iostream>
4: #include <string>
5: #include <cmath>
6:
7:
8: class LFSR {
9:
10: public:
11:     LFSR(std::string seed_, int tap_);
12:     ~LFSR();
13:     int step();
14:     int generate(int k);
15:
16:     friend std::ostream& operator<< (std::ostream &out, LFSR &lfsr);
17:
18: private:
19:     std::string seed;
20:     int tap;
21: };
22:
```

```
1: /*
2: Name: ALbara Mehene
3: Date: 9/25/2016
4: Computing IV
5:
6: */
7:
8:
9: #include <iostream>
10: #include <string>
11: #include <cmath>
12: #include "LFSR.hpp"
13:
14: //constructor
15: LFSR::LFSR(std::string seed_, int tap_){
16:
17:     seed = seed_;
18:     tap = tap_;
19:
20: }
21:
22:
23: int LFSR::step(){
24:     int bit;
25:     int size;
26:
27:     size = seed.length();//stored the amount of elements
28:
29:     bit = seed.at(0) ^ seed[size - tap - 1]; //Took the total elements
and subtracted by the tap and by 1
30:
31:
32:     seed.erase(0, 1); // erased the front element
33:
34:     if(bit == 1){//condition if its 1, it would return the chracter 1
35:         seed.push_back('1');
36:     }
37:     else{//returns 0 if its anything else
38:         seed.push_back('0');
39:     }
40:     //returns bit to test the test.cpp
41:     return bit;
42: }
43:
44: int LFSR::generate(int k){
45:     int temp = 0;
46:
47:     //Condition to test the generate function in test.cpp
48:     for(int i = k - 1; i >= 0; i--){
49:         if(step() == 1){
50:             temp += pow(2,i);
51:         }
52:     }
53:     return temp;
54:
55: }
56: //prints out the string if I were to use the a main. It was not required in
this assigment
57: std::ostream& operator<< (std::ostream &out, LFSR &lfsr){
58:     out << lfsr.seed;
59:     return out;
```

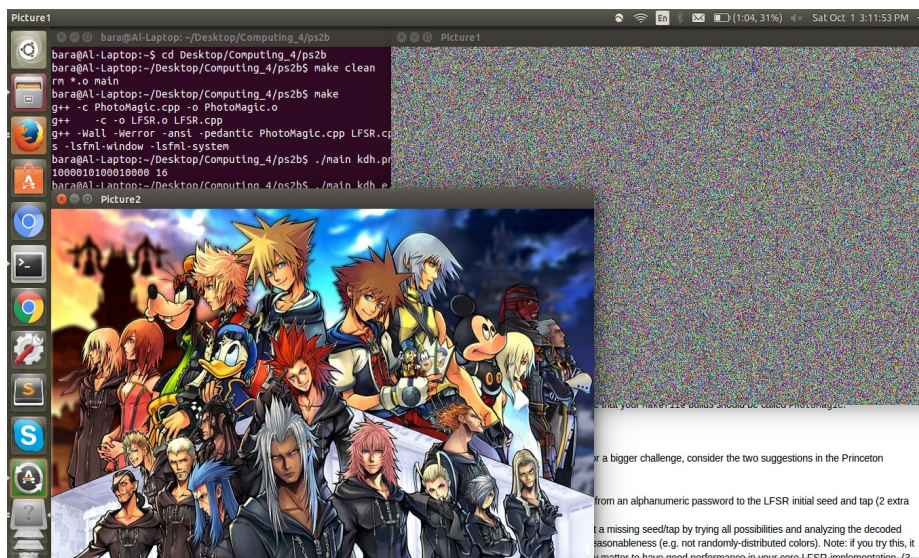
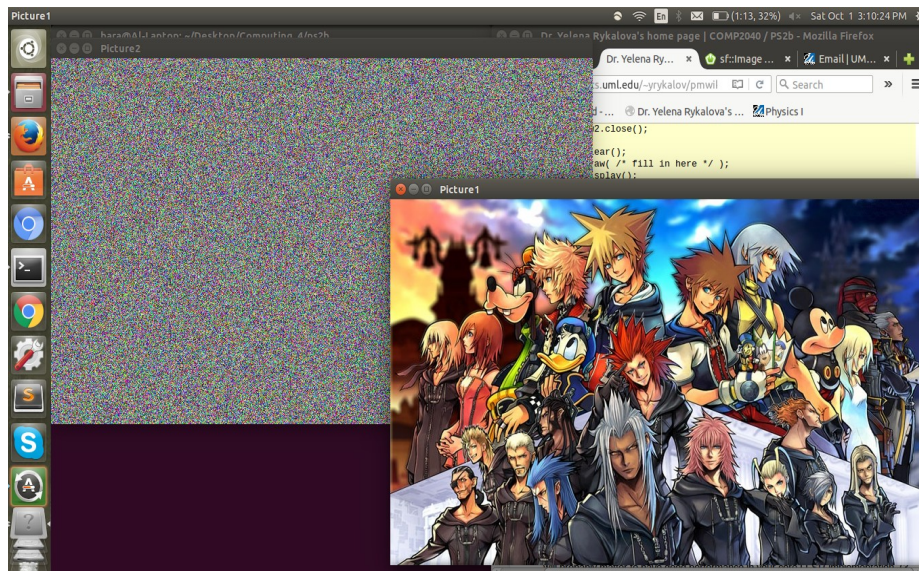
```
60: }  
61:  
62:  
63:  
64: LFSR::~LFSR() {  
65:  
66: }  
67:  
68:  
69:  
70:
```

PS2b:

Ps2b assignment was make a main client that uses LFSR to encrypt and decrypt pictures. To accomplish this task, we needed to pass a image. Each pixel would be passed go through LFSR.

To accomplish this task, I created a main and created a object of class LFSR. The accepted input of the image name would then be stored. I then used a function that would call the generate function and XOR the green, blue, and red. This would be done using the SFML Color type. Then create the output file and insert the new encrypted picture.

Learning about the amount of pixels in different types pictures was helpful to know in the future. XOR was nice to know what type of algorithm it is. Loading images and creating new image files was another skill I learned from this project.



Makefile **Sat Oct 01 15:26:16 2016** **1**

```
1: cc = g++
2: #all
3:
4: all : PhotoMagic
5:
6: PhotoMagic : PhotoMagic.o LFSR.o
7:             $(cc) -Wall -Werror -ansi -pedantic PhotoMagic.cpp LFSR.cpp -o Photo
Magic -lsfml-graphics -lsfml-window -lsfml-system
8:
9: PhotoMagic.o : LFSR.hpp
10:             $(cc) -c PhotoMagic.cpp -o PhotoMagic.o
11:
12: LFSR : LFSR.cpp LFSR.hpp
13:             $(cc) -c LFSR.cpp -o LFSR.o
14:
15: clean:
16:         rm *.o main
17:
```



```
1: /*
2: Name: Albara Mehene
3: Date: 10/1/2016
4: Computing IV
5:
6: */
7:
8: #include <SFML/System.hpp>
9: #include <SFML/Window.hpp>
10: #include <SFML/Graphics.hpp>
11:
12: #include "LFSR.hpp"
13:
14: sf::Image transform(sf::Image picture, LFSR lfsr);
15:
16: int main(int argc, char* argv[])
17: {
18:     if(argc < 5){
19:
20:         std::cout << "input-file.png, output-file.png, seed,
tap" << std::endl;
21:         return -1;
22:     }
23:
24:     std::string input = argv[1];
25:     std::string output = argv[2];
26:     std::string i_seed = argv[3];
27:     int i_tap = atoi(argv[4]);
28:
29:     sf::Image image;
30:     if (!image.loadFromFile(input))
31:         return -1;
32:
33:     sf::Image image_e = image;
34:     sf::Image temp_e;
35:
36:     //Pass the seed and tap function
37:     LFSR l(i_seed, i_tap);
38:     temp_e = transform(image_e, l);
39:
40:
41:     sf::Vector2u size = image.getSize();
42:     sf::Vector2u size2 = temp_e.getSize();
43:     sf::RenderWindow window(sf::VideoMode(size.x, size.y), "Picture1");
44:     sf::RenderWindow windowl(sf::VideoMode(size2.x, size2.y), "Picture2"
);
45:
46:
47:
48:
49:     sf::Texture texture;
50:     texture.loadFromImage(image);
51:
52:     sf::Texture texture_e;
53:     texture_e.loadFromImage(temp_e);
54:
55:     sf::Sprite sprite;
56:     sprite.setTexture(texture);
57:
58:     sf::Sprite sprite_e;
59:     sprite_e.setTexture(texture_e);
```

```
60:
61:     while (window.isOpen() && window1.isOpen())
62:     {
63:         sf::Event event;
64:         while (window.pollEvent(event))
65:         {
66:             if (event.type == sf::Event::Closed)
67:                 window.close();
68:         }
69:         while (window1.pollEvent(event))
70:         {
71:             if (event.type == sf::Event::Closed)
72:                 window1.close();
73:         }
74:
75:         window.clear(sf::Color::White);
76:         window1.clear(sf::Color::White);
77:         window.draw(sprite);
78:         window1.draw(sprite_e);
79:         window.display();
80:         window1.display();
81:     }
82: }
83:
84: // fredm: saving a PNG segfaults for me, though it does properly
85: // write the file
86: if (!temp_e.saveToFile(output))
87:     return -1;
88:
89: return 0;
90: }
91:
92: sf::Image transform(sf::Image picture, LFSR lfsr){
93:     // p is a pixel
94:     sf::Color p;
95:     int temp;
96:     sf::Vector2u size = picture.getSize();
97:
98:     // create photographic negative image of upper-left 200 px square
99:     for (unsigned int x= 0; x < size.x; x++) {
100:         for (unsigned int y = 0; y < size.y; y++) {
101:             p = picture.getPixel(x, y);
102:
103:             temp = lfsr.generate(8);
104:             p.r = p.r ^ temp;
105:
106:             temp = lfsr.generate(8);
107:             p.g = p.g ^ temp;
108:
109:             temp = lfsr.generate(8);
110:             p.b = p.b ^ temp;
111:
112:             picture.setPixel(x, y, p);
113:         }
114:     }
115:     return picture;
116: }
117:
118:
119:
120:
```

121:

```
1:
2:
3: #include <iostream>
4: #include <string>
5: #include <cmath>
6:
7:
8: class LFSR {
9:
10: public:
11:     LFSR(std::string seed_, int tap_);
12:     ~LFSR();
13:     int step();
14:     int generate(int k);
15:
16:     friend std::ostream& operator<< (std::ostream &out, LFSR &lfsr);
17:
18: private:
19:     std::string seed;
20:     int tap;
21: };
22:
```

```
1: /*
2: Name: ALbara Mehene
3: Date: 9/25/2016
4: Computing IV
5:
6: */
7:
8:
9: #include <iostream>
10: #include <string>
11: #include <cmath>
12: #include "LFSR.hpp"
13:
14: //constructor
15: LFSR::LFSR(std::string seed_, int tap_){
16:
17:     seed = seed_;
18:     tap = tap_;
19:
20: }
21:
22:
23: int LFSR::step(){
24:     int bit;
25:     int size;
26:
27:     size = seed.length();//stored the amount of elements
28:
29:     bit = seed.at(0) ^ seed[size - tap - 1]; //Took the total elements
and subtracted by the tap and by 1
30:
31:
32:     seed.erase(0, 1); // erased the front element
33:
34:     if(bit == 1){//condition if its 1, it would return the chracter 1
35:         seed.push_back('1');
36:     }
37:     else{//returns 0 if its anything else
38:         seed.push_back('0');
39:     }
40:     //returns bit to test the test.cpp
41:     return bit;
42: }
43:
44: int LFSR::generate(int k){
45:     int temp = 0;
46:
47:     //Condition to test the generate function in test.cpp
48:     for(int i = k - 1; i >= 0; i--){
49:         if(step() == 1){
50:             temp += pow(2,i);
51:         }
52:     }
53:     return temp;
54:
55: }
56: //prints out the string if I were to use the a main. It was not required in
this assigment
57: std::ostream& operator<< (std::ostream &out, LFSR &lfsr){
58:     out << lfsr.seed;
59:     return out;
```

```
60: }  
61:  
62:  
63:  
64: LFSR::~LFSR() {  
65:  
66: }  
67:  
68:  
69:  
70:
```