PS1 Linear Feedback Shift Register and Image Encoding

What is the assignment?

For part A of the assignment, it was pretty much about building a LFSR class to enable left shifting a binary string easily without doing any complicated math by using its member functions and that based on the tap positions, you would XOR the first bit of the binary string, determining if the first bit should be a 1 or 0 according to the tap positions and last bit of the binary string.

For part B of the assignment, it is pretty much basically decoding an image via the use of the LFSR class to generate pseudorandom numbers to determine the pixel color of the encrypted image and encrypt that image into another file. Then decrypt that same image using the same LFSR bit string to make it look like the same image from the original.

What I learned from this assignment:

1) I learned how to use C++ built in reverse function that uses iterators, making it easy for me to implement the step() function so that it works as intended by the time I start testing the member functions of the LFSR class.
2) I also learn how to utilize the Boost Unit Testing Framework or Library so I can test each individual aspect and edge cases of each member function of the LFSR class.
3) I learned how to use the bitset algorithm that is already built in C++ and utilize it for this assignment.
4) I also learned how alphanumeric conversion works to some extent.

What I accomplished in the assignment:

1) I successfully implemented the LSFR class where the step() function XORs the first bit of the LFSR accordingly based on the tap positions bit values and the last bit of the LFSR then left shifts. I did so by reversing the string and obtaining their appropriate tap position bits and then determining

the first bit by counting the amount of 1s are in the tap positions. If they're an odd amount, the 1st bit becomes a 1 and a 0 if it's not.

2) I also managed to get my generate function working as intended where it basically class step() multiple times based on what its parameter k is, where it will call step() k number of times and then read in the first k number of bits in the LFSR after calling step() k number of times to get the decimal result after calculating the bits and translating them from binary to decimal.
3) I managed to encrypt and decrypt an image using the transform function which I pretty much did so by calling generate(8) inside the transform function so that we will only generate random numbers from 0-255 so that we set the rgb colors after XORing it in order for no one to know what the image is except for the recipient.
4) I also attempted to do the extra credit for this assignment. So how I tried to convert an alphanumeric password using my alphanum_conversion function and what it does is that reads each individual character and converting them into their ascii values first, then add them together before converting the resulting value into binary. Which I achieved by using the bitset class to make it easy to convert an int to binary, then I used generate(8) to produce a random binary string result and converted the return value of generate by using bitset to_string() function and then finally returning the binary string. The output when using the alphanum_conversion() is just like when I normally run it with the input given in the pdf and when I decode the image as well.

Issues that came up in the assignment:

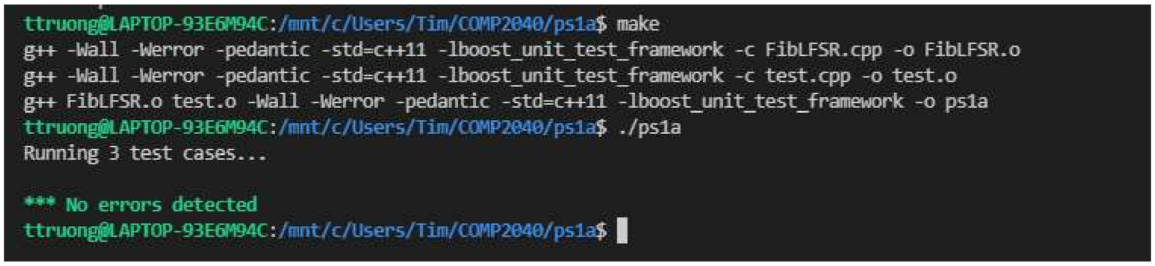
None, no issues came up during the process of building up the assignment.

Key Algorithms, Data Structures, or OO Designs used for this assignment:

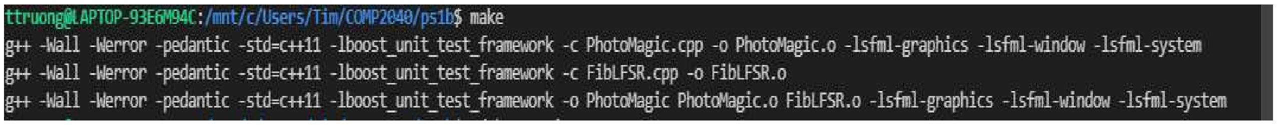
For part A, I did not utilize any data structures or any built-in algorithms within the STL library or <algorithm> library other than the string library. It would have made it easier for me now that I am thinking back to it, but I went along with my initial idea of manipulating the bit string inside a string. So how I tackled this problem is by reversing the given bit string so that when the tap positions (which are 13, 12, and 10 respectively) are at the correct positions of the bit string since a bit string position goes from left to right instead of right to left, we can XOR all the bits in the tap positions and last bit position to determine if the 1st bit of the bit string should be a 1 or a 0. How I determine if the 1st bit of the bit string should be a 1 or 0 is by checking if there an odd amount of 1s or an even amount of 1s. Because 0 ^ 0 ^ 1 ^ 1 results in being 0, while 0 ^ 0 ^ 0 ^ 1 results in the number 1. The next thing I did was reverse the string back to how it was originally and shift all the elements in the string by using the [] operator and a for loop with int i incrementing so that the last bit will have the previous bit. Finally, I determine if the 1st bit should be a 0 or 1 based on the outcome of the XORs and return the value of the 1st bit. And to reiterate, I selected this method or algorithm mainly because it's probably the easiest way to tackle this problem, at least for me personally, and it was the first one I could come up with.

For part B, I utilized the bitset algorithm to help me go through the extra credit much easier as it helps convert the alphanumeric characters that I turned into decimal values to binary much more efficiently.

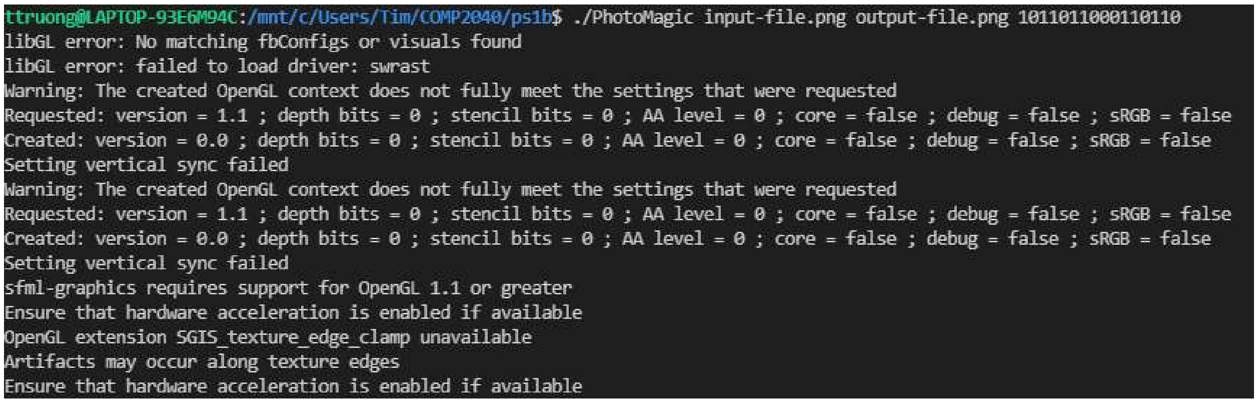
Screenshot(s):



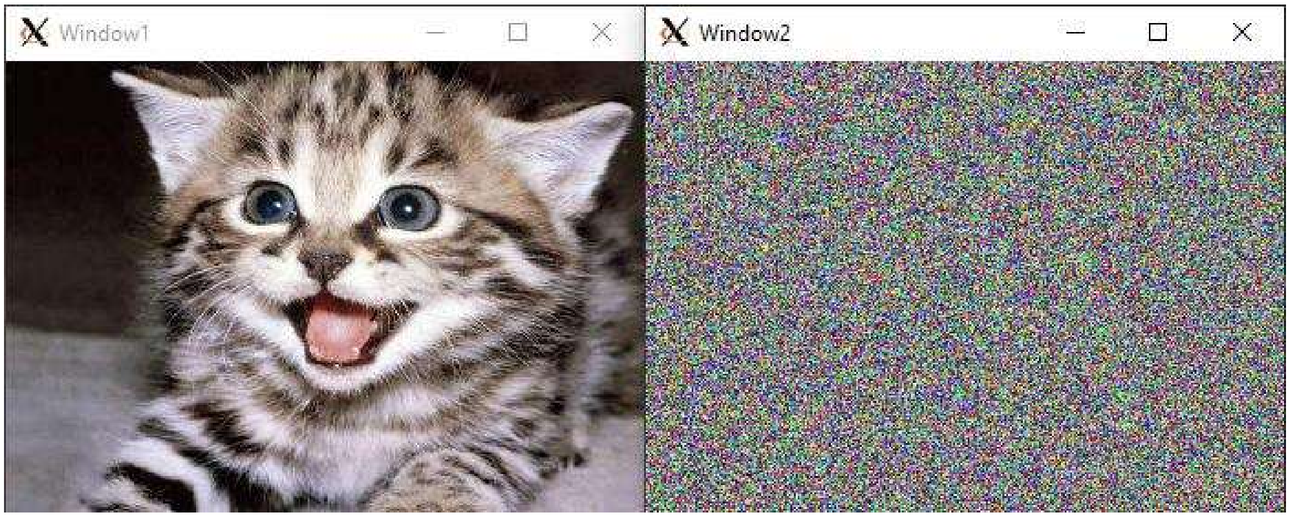
```
ttruong@LAPTOP-93E6M94C:/mnt/c/Users/Tim/COMP2040/ps1a$ make
g++ -Wall -Werror -pedantic -std=c++11 -lboost_unit_test_framework -c FibLFSR.cpp -o FibLFSR.o
g++ -Wall -Werror -pedantic -std=c++11 -lboost_unit_test_framework -c test.cpp -o test.o
g++ FibLFSR.o test.o -Wall -Werror -pedantic -std=c++11 -lboost_unit_test_framework -o ps1a
ttruong@LAPTOP-93E6M94C:/mnt/c/Users/Tim/COMP2040/ps1a$ ./ps1a
Running 3 test cases...

*** No errors detected
ttruong@LAPTOP-93E6M94C:/mnt/c/Users/Tim/COMP2040/ps1a$
```

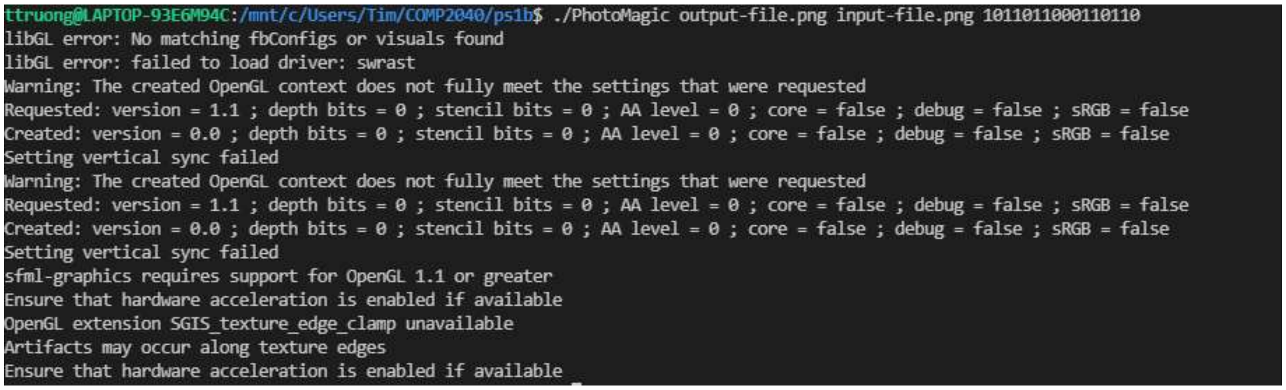


```
ttruong@LAPTOP-93E6M94C:/mnt/c/Users/Tim/COMP2040/ps1b$ make
g++ -Wall -Werror -pedantic -std=c++11 -lboost_unit_test_framework -c PhotoMagic.cpp -o PhotoMagic.o -lsfml-graphics -lsfml-window -lsfml-system
g++ -Wall -Werror -pedantic -std=c++11 -lboost_unit_test_framework -c FibLFSR.cpp -o FibLFSR.o
g++ -Wall -Werror -pedantic -std=c++11 -lboost_unit_test_framework -o PhotoMagic PhotoMagic.o FibLFSR.o -lsfml-graphics -lsfml-window -lsfml-system
```

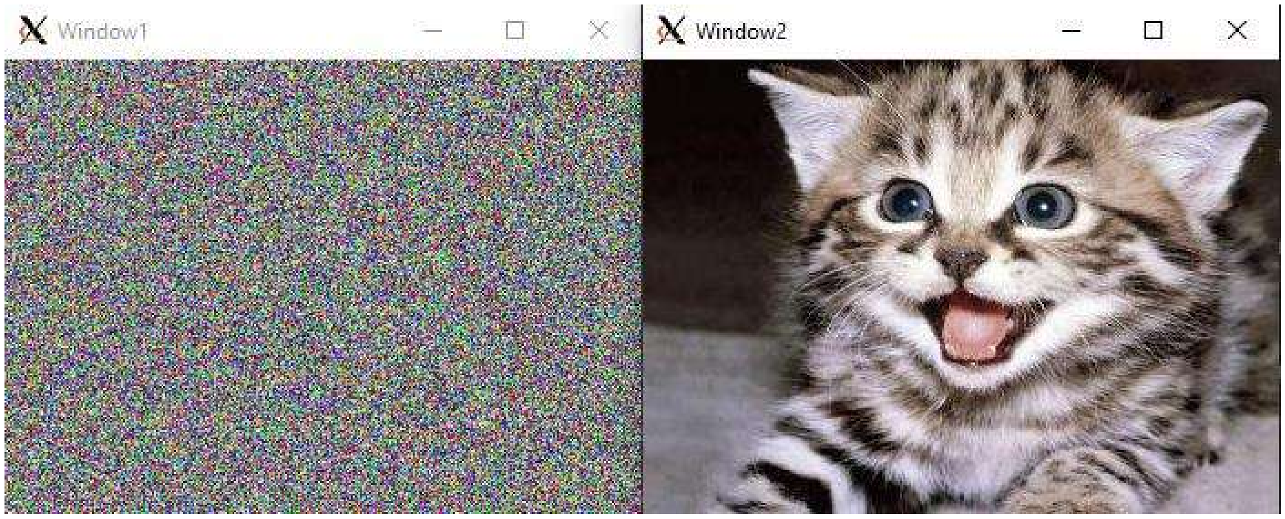


```
ttruong@LAPTOP-93E6M94C:/mnt/c/Users/Tim/COMP2040/ps1b$ ./PhotoMagic input-file.png output-file.png 1011011000110110
libGL error: No matching fbConfigs or visuals found
libGL error: failed to load driver: swrast
Warning: The created OpenGL context does not fully meet the settings that were requested
Requested: version = 1.1 ; depth bits = 0 ; stencil bits = 0 ; AA level = 0 ; core = false ; debug = false ; sRGB = false
Created: version = 0.0 ; depth bits = 0 ; stencil bits = 0 ; AA level = 0 ; core = false ; debug = false ; sRGB = false
Setting vertical sync failed
Warning: The created OpenGL context does not fully meet the settings that were requested
Requested: version = 1.1 ; depth bits = 0 ; stencil bits = 0 ; AA level = 0 ; core = false ; debug = false ; sRGB = false
Created: version = 0.0 ; depth bits = 0 ; stencil bits = 0 ; AA level = 0 ; core = false ; debug = false ; sRGB = false
Setting vertical sync failed
sfml-graphics requires support for OpenGL 1.1 or greater
Ensure that hardware acceleration is enabled if available
OpenGL extension SGIS_texture_edge_clamp unavailable
Artifacts may occur along texture edges
Ensure that hardware acceleration is enabled if available
```





```
ttruong@LAPTOP-93E6M94C:/mnt/c/Users/Tim/COMP2040/ps1b$ ./PhotoMagic output-file.png input-file.png 1011011000110110
libGL error: No matching fbConfigs or visuals found
libGL error: failed to load driver: swrast
Warning: The created OpenGL context does not fully meet the settings that were requested
Requested: version = 1.1 ; depth bits = 0 ; stencil bits = 0 ; AA level = 0 ; core = false ; debug = false ; sRGB = false
Created: version = 0.0 ; depth bits = 0 ; stencil bits = 0 ; AA level = 0 ; core = false ; debug = false ; sRGB = false
Setting vertical sync failed
Warning: The created OpenGL context does not fully meet the settings that were requested
Requested: version = 1.1 ; depth bits = 0 ; stencil bits = 0 ; AA level = 0 ; core = false ; debug = false ; sRGB = false
Created: version = 0.0 ; depth bits = 0 ; stencil bits = 0 ; AA level = 0 ; core = false ; debug = false ; sRGB = false
Setting vertical sync failed
sfml-graphics requires support for OpenGL 1.1 or greater
Ensure that hardware acceleration is enabled if available
OpenGL extension SGIS_texture_edge_clamp unavailable
Artifacts may occur along texture edges
Ensure that hardware acceleration is enabled if available
```


Window1
Window2

```
CC = g++
CFLAGS = -Wall -Werror -pedantic -std=c++11 -lboost_unit_test_framework
CFLAGSX = -Wall -Werror -pedantic -std=c++11
OBJECTS = PhotoMagic.o FibLFSR.o
TOBJECTS = test.o
LIBS = -lboost_unit_test_framework
SFML = -lsfml-graphics -lsfml-window -lsfml-system

all: PhotoMagic ps1a
PhotoMagic: $(OBJECTS)
	$(CC) $(CFLAGS) -o PhotoMagic $(OBJECTS) $(SFML)
ps1a: FibLFSR.o $(TOBJECTS)
	$(CC) FibLFSR.o $(TOBJECTS) $(CFLAGSX) $(LIBS) -o ps1a
PhotoMagic.o: PhotoMagic.cpp
	$(CC) $(CFLAGS) -c PhotoMagic.cpp -o PhotoMagic.o $(SFML)
FibLFSR.o: FibLFSR.cpp FibLFSR.h
	$(CC) $(CFLAGS) -c FibLFSR.cpp -o FibLFSR.o
test.o: test.cpp
	$(CC) $(CFLAGSX) $(LIBS) -c test.cpp -o test.o
clean:
	rm PhotoMagic.o FibLFSR.o test.o PhotoMagic ps1a
```

```
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include "FibLFSR.h"
#include <bitset>

/*************************************************************
 *Name: Tim Truong
 *Course name: COMP.2040
 *Assignment: PS1b - PhotoMagic
 *Instructor’s name: Dr. James Daly
 *Date: 2/7/22
 *Sources Of Help: SFML Tutorial Site, StackOverflow
*************************************************************/

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

std::string alphanum_conversion(std::string ascii);

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

        if ((argc < 4) || (argc > 4)) {
                exit(0);
        }
        else {
                // rbit = alphanum_conversion(argv[3]);
                rbit = (argv[3]);
        }

        FibLFSR L(rbit);
        sf::Image image;
        sf::Image image2;

        if (!image.loadFromFile(argv[1])) {
                return -1;
        }
        if (!image2.loadFromFile(argv[1])) {
                return -1;
        }

        transform(image2, &L);

        sf::Texture texture;
        sf::Texture texture2;
        sf::Sprite sprite;
        sf::Sprite sprite2;

        sf::Vector2u size = image.getSize();
        unsigned int width = size.x;
        unsigned int height = size.y;

        sf::RenderWindow window1(sf::VideoMode(width, height),
        "Window1");
        sf::RenderWindow window2(sf::VideoMode(width, height),
        "Window2");

        window1.setPosition(sf::Vector2i(100, 100));
        window2.setPosition(sf::Vector2i(450, 100));

        if (!image2.saveToFile(argv[2])) {
                return -1;
        }

        texture.loadFromImage(image);
```

```cpp
        sprite.setTexture(texture);

        texture2.loadFromImage(image2);
        sprite2.setTexture(texture2);

        while (window1.isOpen() && window2.isOpen()) {
                sf::Event event;
                while (window1.pollEvent(event)) {
                        if (event.type == sf::Event::Closed) {
                                window1.close();
                        }

                        if(sf::Keyboard::isKeyPressed
                        (sf::Keyboard::Escape)) {
                                window1.close();
                        }
                }
                while (window2.pollEvent(event)) {
                        if (event.type == sf::Event::Closed) {
                                window2.close();
                        }

                        if(sf::Keyboard::isKeyPressed
                        (sf::Keyboard::Escape)) {
                                window2.close();
                        }
                }
                window1.clear();
                window1.draw(sprite);
                window1.display();
                window2.clear();
                window2.draw(sprite2);
                window2.display();
        }

        return 0;
}

void transform(sf::Image& pic, FibLFSR* l) {
        sf::Color p;
        sf::Vector2u size = pic.getSize();
        unsigned int width = size.x;
        unsigned int height = size.y;

        for (unsigned int x = 0; x < width; x++) {
                for (unsigned int y = 0; y < height; y++) {
                        p = pic.getPixel(x, y);
                        p.r ^= l->generate(8);
                        p.g ^= l->generate(8);
                        p.b ^= l->generate(8);
                        pic.setPixel(x, y, p);
                }
        }
        return;
}

std::string alphanum_conversion(std::string ascii) {
        std::string fin_binary;
        int num = 0, tmp = 0;
        for (size_t i = 0; i < ascii.size(); i++) {
                num = static_cast<int>(ascii[i]);
                tmp += num;
        }

        std::bitset<16> bset1(tmp);

        FibLFSR L(bset1.to_string());
        num = L.generate(8);
        bset1 = num;
        fin_binary = bset1.to_string();

        return fin_binary;
}
```

```
#ifndef LFSR_H
#define LFSR_H

#include <iostream>
#include <string>
#include <algorithm>
#include <sstream>
#include <cmath>

class FibLFSR {
public:
    FibLFSR(std::string seed);
    int step(void);
    int generate(int k);
    friend std::ostream & operator << (std::ostream& out,
    const FibLFSR& L);
private:
    std::string new_seed;
    int seed_length;
};

#endif
```

```
#include "FibLFSR.h"

/**********************************************************
 *Name: Tim Truong
 *Course name: COMP.2040
 *Assignment: PS1b - PhotoMagic
 *Instructor's name: Dr. James Daly
 *Date: 2/7/22
 *Sources Of Help: SFML Tutorial Site, StackOverflow
**********************************************************/

FibLFSR::FibLFSR(std::string seed) {
    seed_length = seed.length();
    new_seed = seed;

    if(seed_length > 32 || seed_length < 15) {
        throw std::exception{};
    }
}

int FibLFSR::step(void) {
    std::string temp = new_seed;
    int check = 0, tap_pos = 13, tap_pos2 = 10, tap_pos3 = 12;

    reverse(temp.begin(), temp.end());

    for (int i = 0; i < seed_length; i++) {
        if ((i == tap_pos) && (temp[tap_pos] == '1')) {
            check++;
        }
        else if ((i == tap_pos2) && (temp[tap_pos2] == '1')) {
            check++;
        }
        else if ((i == tap_pos3) && (temp[tap_pos3] == '1')) {
            check++;
        }
        else if ((i == seed_length - 1) &&
        (temp[seed_length - 1] == '1')) {
            check++;
        }

    }

    if (check % 2 == 1) {
        check = 1;
    }
    else {
        check = 0;
    }

    reverse(temp.begin(), temp.end());

    for (int i = 0; i < seed_length; i++) {
        if (i == seed_length - 1) {
            if (check == 0) {
                temp[seed_length - 1] = '0';
                break;
            }
            else if (check == 1) {
                temp[seed_length - 1] = '1';
                break;
            }
        }

        temp[i] = temp[i + 1];
    }

    new_seed = temp;

    return new_seed[seed_length - 1] - '0';
}

int FibLFSR::generate(int k) {

    int decimal = 0, base = 0;
    for (int i = 0; i <= k - 1; i++) {
        step();
    }

    for (int i = seed_length - 1; i >= seed_length - k; i--) {
        if (new_seed[i] == '1') {
            decimal += pow(2, base);
        }
        base++;
    }

    return decimal;

}

std::ostream & operator << (std::ostream& out, const FibLFSR& L) {
    out << L.new_seed;

    return out;
}



```

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

BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {

  FibLFSR l("1011011000110110");
  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);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 1);

  FibLFSR l2("1011011000110110");
  BOOST_REQUIRE(l2.generate(9) == 51);

}

BOOST_AUTO_TEST_CASE(SeventeenBits_Three_Taps) {

  FibLFSR l("10010010101111110");

  std::ostringstream oss;
  oss << l;
  std::string test = oss.str();

  BOOST_REQUIRE(test == "10010010101111110");
  BOOST_REQUIRE(l.step() == 1);

  oss.str(std::string());
  oss << l;
  test = oss.str();

  BOOST_REQUIRE(test == "00100101011111101");
  BOOST_REQUIRE(l.step() == 0);

  oss.str(std::string());
  oss << l;
  test = oss.str();

  BOOST_REQUIRE(test == "01001010111111010");
  BOOST_REQUIRE(l.step() == 0);

  oss.str(std::string());
  oss << l;
  test = oss.str();

  BOOST_REQUIRE(test == "10010101111110100");
  BOOST_REQUIRE(l.step() == 0);

  oss.str(std::string());
  oss << l;
  test = oss.str();

  BOOST_REQUIRE(test == "00101011111101000");
  BOOST_REQUIRE(l.step() == 0);

  oss.str(std::string());
  oss << l;
  test = oss.str();

```

```cpp
  BOOST_REQUIRE(test == "0101011111010000");
  BOOST_REQUIRE(l.step() == 0);

  oss.str(std::string());
  oss << l;
  test = oss.str();

  BOOST_REQUIRE(test == "1010111110100000");
  BOOST_REQUIRE(l.step() == 1);

  oss.str(std::string());
  oss << l;
  test = oss.str();

  BOOST_REQUIRE(test == "0101111101000001");
  BOOST_REQUIRE(l.step() == 1);

  oss.str(std::string());
  oss << l;
  test = oss.str();

  BOOST_REQUIRE(test == "1011111010000011");

  FibLFSR l2("1101010010011001 0");
  BOOST_REQUIRE(l2.generate(10) == 126);
  BOOST_CHECK_EQUAL(l2.generate(9), 19);

  BOOST_REQUIRE_THROW(FibLFSR l3("11111111111111111111111111111111"),
  std::exception);
  BOOST_REQUIRE_NO_THROW(FibLFSR l4("10101100000011001"));
  BOOST_REQUIRE_THROW(FibLFSR l5("1001"), std::exception);
  BOOST_REQUIRE_THROW(FibLFSR l6("10010010011110"), std::exception);
  BOOST_REQUIRE_NO_THROW(FibLFSR l7("100100100111101"));
}

BOOST_AUTO_TEST_CASE(Fifteen_Bits_Three_Taps) {

  FibLFSR l("001010010101110");
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 1);

  FibLFSR l2("110100110010110");
  BOOST_REQUIRE(l2.generate(10) == 44);
  BOOST_CHECK_EQUAL(l2.generate(9), 40);

  FibLFSR l3("100100101011111 0");

  std::ostringstream ss;
  ss << l3;
  std::string test = ss.str();

  BOOST_REQUIRE(test == "1001001010111110");
  BOOST_REQUIRE(l3.step() == 0);

  ss.str(std::string());
  ss << l3;
  test = ss.str();

  BOOST_REQUIRE(test == "0010010101111100");
  BOOST_REQUIRE(l3.step() == 0);

  ss.str(std::string());
  ss << l3;
  test = ss.str();

  BOOST_REQUIRE(test == "0100101011111000");
  BOOST_REQUIRE(l3.step() == 0);

  ss.str(std::string());
  ss << l3;
  test = ss.str();

  BOOST_REQUIRE(test == "1001010111110000");
  BOOST_REQUIRE(l3.step() == 1);

  ss.str(std::string());
  ss << l3;
  test = ss.str();

  BOOST_REQUIRE(test == "0010101111100001");
  BOOST_REQUIRE(l3.step() == 1);

  ss.str(std::string());
  ss << l3;
  test = ss.str();

  BOOST_REQUIRE(test == "0101011111000011");
  BOOST_REQUIRE(l3.step() == 0);

  ss.str(std::string());
  ss << l3;
  test = ss.str();

  BOOST_REQUIRE(test == "1010111110000110");
  BOOST_REQUIRE(l3.step() == 1);

  ss.str(std::string());
  ss << l3;
  test = ss.str();

  BOOST_REQUIRE(test == "0101111100001101");
  BOOST_REQUIRE(l3.step() == 0);

  ss.str(std::string());
  ss << l3;
  test = ss.str();

  BOOST_REQUIRE(test == "1011111000011010");

}
```