```
 1: CC = g++
 2: CFLAGS = -Wall -Werror -pedantic --std=c++14
 3: LIBS = -lboost_unit_test_framework
 4: DEPS = FibLFSR.h
 5: SFMLFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
 6:
 7: %.o: %.cpp $(DEPS)
 8:         $(CC) $(CFLAGS) -c $<
 9:
10: all: PhotoMagic
11:
12: PhotoMagic: photoMagic.o FibLFSR.o
13:         $(CC) $(CFLAGS) -o PhotoMagic $^ $(LIBS) $(SFMLFLAGS)
14:
15: clean:
16:         rm *.o PhotoMagic
```

```
    1: /**
    2: * photoMagic.cpp - Essentially to encode and decode an image using the Fi
bLFSR
    3: * that was programmed back in ps1a as an assignment
    4: *
    5: * Date 2/1/22 - 2/7/22
    6: *
    7: * Created by: Anson Cheang
    8: *
    9: */
   10:
   11: #include <SFML/System.hpp>
   12: #include <SFML/Window.hpp>
   13: #include <SFML/Graphics.hpp>
   14: #include "FibLFSR.h"
   15:
   16: // transforms image using FibLFSR
   17: void transform( sf::Image&, FibLFSR*);
   18:
   19: // display an encrypted copy of the picture, using the LFSR
   20: // to do the encryption
   21: int main(int argc, char* argv[])
   22: {
   23:     sf::Image image1;
   24:     FibLFSR encryptionCode(argv[3]);
   25:         if (!image1.loadFromFile(argv[1]))
   26:     {
   27:                 return -1;
   28:     }
   29:
   30:         sf::Vector2u size = image1.getSize();
   31:         sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "Input");
   32:
   33:         sf::Texture texture;
   34:         texture.loadFromImage(image1);
   35:
   36:         sf::Sprite sprite;
   37:         sprite.setTexture(texture);
   38:
   39:     transform(image1, &encryptionCode);
   40:
   41:     sf::Vector2u size2 = image1.getSize();
   42:         sf::RenderWindow window2(sf::VideoMode(size2.x, size2.y), "Output
");
   43:
   44:         sf::Texture texture2;
   45:         texture2.loadFromImage(image1);
   46:
   47:         sf::Sprite sprite2;
   48:         sprite2.setTexture(texture2);
   49:
   50:     while (window1.isOpen() && window2.isOpen())
   51:     {
   52:         sf::Event event;
   53:         while (window1.pollEvent(event)) {
   54:             if (event.type == sf::Event::Closed)
   55:             {
   56:                 window1.close();
   57:             }
   58:         }
   59:         while (window2.pollEvent(event))
   60:         {
   61:             if (event.type == sf::Event::Closed)
   62:             {
   63:                 window2.close();
```

```
64:             }
65:         }
66:         window1.clear();
67:         window1.draw(sprite);
68:         window1.display();
69:         window2.clear();
70:         window2.draw(sprite2);
71:         window2.display();
72:     }
73:
74:     if (!image1.saveToFile(argv[2]))
75:         {
76:         return -1;
77:     }
78:
79:     return 0;
80: }
81:
82: void transform( sf::Image& image, FibLFSR* encryptionCode)
83: {
84:     sf::Color p;
85:         sf::Vector2u size = image.getSize();
86:
87:         // create photographic negative image of upper-left 200 px square
88:         for (int x = 0; x < static_cast<int>(size.x); x++) {
89:                 for (int y = 0; y < static_cast<int>(size.y); y++) {
90:                         p = image.getPixel(x, y);
91:                         p.r = p.r ^ encryptionCode->generate(8);
92:                         p.g = p.g ^ encryptionCode->generate(8);
93:                         p.b = p.b ^ encryptionCode->generate(8);
94:                         image.setPixel(x, y, p);
95:                 }
96:         }
97: }
```

```
 1: #ifndef FibLFSR_H_
 2: #define FibLFSR_H_
 3: #include <string>
 4: #include <vector>
 5: #include <iostream>
 6:
 7: using namespace std;
 8:
 9: class FibLFSR {
10: public:
11: FibLFSR(string seed);   // constructor to create LFSR with
12:                         // the given initial seed
13: int step();             // simulate one step and return the
14:                         // new bit as 0 or 1
15: int generate(int k);    // simulate k steps and return
16:                         // k-bit integer
17: friend ostream& operator<<(ostream& out, FibLFSR CurrentBits);
18: private:
19: vector<int> list;
20: };
21:
22: #endif
```

```
    1: /**
    2: * FibLFSR.cpp - To run the FubLFSR class, in which they constructor, step
(),
    3: * and generator(), essentially using Linear feedback shift register
    4: *
    5: * Date 1/24/22 - 1/31/22
    6: *
    7: * Created by: Anson Cheang
    8: *
    9: */
   10:
   11: #include "FibLFSR.h"
   12: #include <vector>
   13: #include <string>
   14: #include <cmath>
   15:
   16: using namespace std;
   17:
   18: FibLFSR::FibLFSR(string seed)
   19: {
   20:     string Pbit;
   21:     for(int i = 0; i < static_cast <int> (seed.length()); i++)
   22:     {
   23:         Pbit = seed[i];
   24:         list.push_back(stoi(Pbit, 0, 10));
   25:
   26:     }
   27: }
   28:
   29: int FibLFSR::step()
   30: {
   31:     int size = static_cast <int> (list.size());
   32:     int RBit = list[0] ^ list[2] ^ list[3] ^ list[5];
   33:     for(int i = 0; i < size - 1; i++)
   34:     {
   35:         list[i] = list[i+1];
   36:     }
   37:
   38:     list[size - 1] = RBit;
   39:
   40:     return RBit;
   41: }
   42:
   43: int FibLFSR::generate(int k)
   44: {
   45:     if(k > 32 || k < 0)
   46:     {
   47:         throw out_of_range("The value inputted isn't within the range of
0 - 32");
   48:     }
   49:     int RBit;
   50:     int total = 0;
   51:     for(int i = 0; i < k; i++)
   52:     {
   53:         RBit = step();
   54:         total = total + pow(2, k-i-1) * RBit;
   55:     }
   56:
   57:     return total;
   58: }
   59:
   60: ostream& operator<<(ostream& out, FibLFSR CurrentBits)
   61: {
   62:     vector<int>::iterator it;
   63:     for(it = CurrentBits.list.begin(); it != CurrentBits.list.end(); it++
```

```
)
    64:     {
    65:         out << *it;
    66:     }
    67:     return out;
    68: }
```