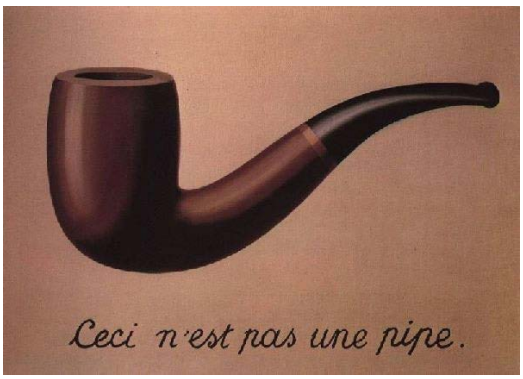
PS2: Linear Feedback Shift Register (part B)

For this portion of the assignment, you will:

- Write a C++ program to read three arguments from the command line: source image filename, output image filename, and `FibLFSR` seed.
- Use SFML to load the source image from disk and display it in its own window.
- Use your debugged `FibLFSR` class to encode (or decode) the image.
- Display the encoded/decoded image in its own window.
- Save the new image to disk.

Detals



Your task is to write a program PhotoMagic.cpp that can encrypt and decrypt pictures, by implementing the following API:

Transform function.

The `transform()` function takes an image and an `FibLFSR` as arguments. It transforms image using the linear feedback shift register as follows:

For each pixel (x, y), in row major order — (0, 0), (0, 1), (0, 2), ... — extract the red, green, and blue components of the color (each component is an integer between 0 and 255). Then, XOR the red component with a newly-generated 8-bit integer. Do the same for the green (using another new 8-bit integer) and, finally, the blue. Create a new color using the result of the XOR operations, and set the pixel in the new picture to that color.

```
// transforms image using FibLFSR
void transform( sf::Image&, FibLFSR*);
```

Main function.

The `main()` function takes three command-line arguments: an image source filename, an encrypted image filename, the initial `FibLFSR` seed. It should display the transformed picture on the screen.

Your main code should be in a file named `PhotoMagic.cpp` and should accept command line arguments as follows (e.g.):

```
% PhotoMagic input-file.png output-file.png 1011011000110110
```

which should take the input file and encrypt it using the method described above, with `FibLFSR` seed `1011011000110110`.

Your program should display the source file and encrypted file, and write out the encrypted file to `output-file.png`. Note: If you save as a JPG, you won't be able to restore the file properly. (Why not?)

Then, if you re-run your program on the encrypted file, and give it the same LFSR seed and tap, it should produce the original input file! (Make sure you understand why.)

Make sure to transform the whole image.

Note: to work with two SFML windows, create two window objects (e.g., `window1` and `window2`), and use this as your event loop:

```
while (window1.isOpen() && window2.isOpen()) {
  sf::Event event;
  while (window1.pollEvent(event)) {
    if (event.type == sf::Event::Closed)
      window1.close();
  }
  while (window2.pollEvent(event)) {
    if (event.type == sf::Event::Closed)
      window2.close();
  }
  window1.clear();
  window1.draw( /* fill in here */ );
  window1.display();
  window2.clear();
  window2.draw(/* fill in here */ );
  window2.display();
}
```

What to turn in

- code files `PhotoMagic.cpp`, `FibLFSR.cpp`, and `FibLFSR.hpp` plus your `Makefile`
- two screenshots: one showing the encryption process (`encode.png`) and the other showing decryption (`decode.png`), and
- a `ps2b-readme.txt` with: name, statement of the functionality of your program (e.g., fully works, or explanation of partial functionality). Optional: any other notes
- The executable file that your `Makefile` builds should be called `PhotoMagic`.

How to turn it in

Submit on Blackboard.

Grading rubric

Feature	Value	Comment
core implementation	8	full & correct implementation = 8 pts; nearly complete = 6pts; part way = 4 pts; started = 2 pt must parse command line arguments properly (2 pts)
screenshots	2	must show both original and encrypted whole image and then encrypted and decrypted image
Makefile	2	Makefile included targets `all` and `clean` must exist `all` should build `PhotoMagic` must have dependencies correct
ps2b-readme.txt	2	
Total	**14**	

Extra credit

If you're looking for a bigger challenge, consider the two suggestions:

1. Converting from an alphanumeric password to the `FibLFSR` initial seed (2 extra points)
2. Figuring out a missing seed by trying all possibilities and analyzing the decoded image for reasonableness (e.g. not randomly-distributed colors). Note: if you try this, it will probably matter to have good performance in your core `FibLFSR` implementation. (3 extra points)

If you do any of the extra credit work, make sure to describe exactly what you did in `ps2b-readme.txt`.