PS2: Linear Feedback Shift Register (part B)

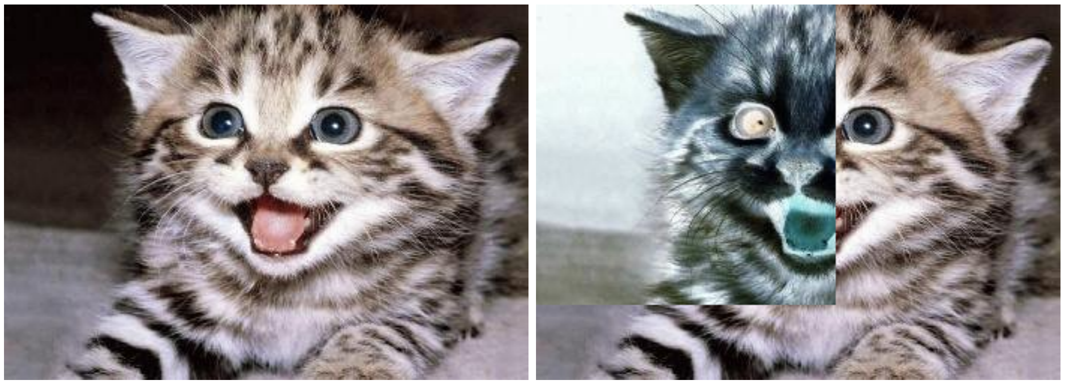
For this portion of the assignment, you will:

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

Details

Please see starter code `pixels.cpp` for using SFML to load, manipulate, display, and save an image.

The code reads in a file and then uses individual pixel access to photographically negate an upper 200 px square, like this:



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 01101000010100010000 16
```

which should take the input file and encrypt it using the method described in the Princeton assignment, with LFSR seed 01101000010100010000 and tap position 16.

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, not just the upper-left 200x200 pixel square from the demo code above.

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`, `LFSR.cpp`, and `LFSR.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, such as extra credit attempts (see below).
- The executable file that your `Makefile` builds should be called `PhotoMagic`.

How to turn it in

- Submit a "tarball" via the PS2b assignment page on Blackboard. The filename of your tarball file should include your own name; e.g., `Tom_Wilkes_ps2b.tar.gz` .

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 LFSR initial seed and tap (2 extra points)
2. Figuring out a missing seed/tap 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 LFSR 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`.