Concepts of Parallel and Distributed Systems (CSCI-251)
Project 3: Stream Cipher-Linear Feedback Shift Registers (LFSR)

I. Goals

Linear Feedback Shift Registers (LFSR) are often used for generating random numbers as well as for error checking and correction. In this project, we will focus on using the LFSR as a major component in a stream cipher, to generate some pseudorandom sequence of bits, known as a keystream. We want to also use this algorithm for the encryption and decryption of simple messages and images.

II. Overview

Write a command line application that encrypts and decrypts messages and images with a stream cipher that uses the LFSR key generator. This program will use .NET version 8 and C# programming language.

III. Stream Cipher-Linear Feedback Shift Registers

Recall from your lecture notes that a stream cipher is a symmetric key cipher that encrypts plain text one digit at a time. A stream cipher with LFSR has the following operations:

- **Key Initialization**: The LFSR is initialized with an initial seed value (secret key)
- **LFSR keystream Generation**: The LFSR generates a pseudorandom sequence of bits (keystream) by shifting the bits one position to the left; replacing the vacated bit by performing an *Exclusive Or*(*XOR*) operation between the bit shifted off and the bit previously at a **given *tap* position** in the register.
- **Encryption**: The keystream generated by the LFSR is combined with the plaintext using a bitwise XOR operation. Each bit of the plaintext is XORed with a corresponding bit from the pseudorandom sequence.
- **Decryption**: To decrypt the ciphertext, a similar keystream is generated using the LFSR initialized with the seed value and tap position. The keystream is XORed with the ciphertext to recover the original plaintext.

IV. Requirements

Your program must be a .net core command line program:

dotnet run <option> <other arguments>

For “option”, your program will accept any of the following:

1. Cipher
2. GenerateKeystream
3. Encrypt
4. Decrypt
5. MultipleBits
6. EncryptImage
7. DecryptImage

Each of these options will accomplish a basic task, as detailed below (along with the extra command line options):

1. **Cipher** *seed tap*: This option takes the initial seed and a tap position from the user and simulates one step of the LFSR cipher. This returns and prints the new seed and the recent rightmost bit (which may be 0 or 1).
2. **GenerateKeystream** *seed tap step*: This option will accept a seed, tap position, and the number of steps (let's say n: a positive integer). For each step, the LFSR cipher simulation prints the new seed and the rightmost bit (see sample run). At the end of the iteration, the keystream (a string consisting of "n" rightmost bits) is saved in a file in the same directory.
3. **Encrypt** *plaintext:* This option will accept plaintext in bits; perform an XOR operation with the retrieved keystream from the file; and return a set of encrypted bits (ciphertext).
4. **Decrypt** *ciphertext:* This option will accept ciphertext in bits; perform an XOR operation with the retrieved keystream from the file; and return a set of decrypted bits (plaintext).
5. **MultipleBit** *seed tap step iteration*: This option will accept an initial *seed, tap*, *step* - a positive integer (let's say p) and perform 'p' steps of the LFSR cipher simulation. It will also accept *iteration*- a positive integer (let's say w). After each iteration *i (0 ≤ i <w)*, it returns a new seed, and accumulated integer value.
   At each step, it performs the cipher simulation; gets the recent rightmost bit and performs an arithmetic with this bit value.

   <u>How to get the accumulated Integer Value for each Iteration</u>

   At each iteration, this method **prints only the seed at the end of the last step and prints an accumulated integer value obtained after the arithmetic operation is done at the last step.**

   To get the accumulated integer value for each iteration: Initialize a variable to zero and perform the following 'p' times (p= step):
   a. double the value of the variable.
   b. Add the recent rightmost bit returned by each step to the variable to get a new variable value.

   For instance, if p =5, and we have 1, 1, 0, 0, 1; each number being the rightmost bit value for all the 5 steps; the variable takes on the values 1, 3, 6, 12, and finally 25, ending with the binary representation of the bit sequence. The accumulated integer value is 25 for the first iteration in this instance.

6. **EncyrptImage** *imagefile seed tap:* Given an image with a seed and a tap position , generate a row-encrypted image. You should install a nugget package called **SkiaSharp.Views** and include the namespace: "SkiaSharp" in your program. **Do not use any other nugget packages or libraries for images.** You will need this namespace to convert the image found in the path to a bitmap image. You will use one of its classes - "SkiColor" to access each pixel in the bitmap image. https://learn.microsoft.com/en-us/dotnet/api/skiasharp.skcolor?view=skiasharp-2.88.

   To encrypt an image, follow the steps below:

   a. Generate a new-seed using the seed and tap. Do this using the cipher option.
   b. Generate a row-encrypted bitmap image.

      For each pixel in each row of the original bitmap, do the following for its red, blue and green color components:

      - Use the new-seed to generate a random unsigned 8-bit integer
      - Perform an XOR on the random 8-bit integer and the value of the color component of the pixel; the result of the XOR will be the new value to replace the color component.

      Create a new color object using the new red, new green, and new blue components. Then, set the new color to the pixel.

   c. Encode the row-encrypted bitmap image to an image file and save it in the same directory where your program is. Name the image: "Original_File_NameENCRYPTED"
7. **DecyrptImage** *imagefile tap seed.* Given an encrypted image, a seed and tap position, generate the original image and save it with a different name in the same directory. Name the image: "Original_File_NameNEW".
8. If the user specifies invalid user arguments for any of the options, your program should provide an appropriate error message indicating the problem.

V. Design

- The program must be command-line-driven.
- The output (minus error messages) must match the writeup.
- Command line help must be provided.
- The program must be designed using object-oriented design principles as appropriate.
- The program must make use of reusable software components as appropriate.
- Each class and interface must include a comment describing the overall class or interface.

VI. Important Notes

- Please include the nugget packages "**SkiaSharp.NativeAssets.Linux"** for Linux and "**SkiaSharp.NativeAssets.macOS**" for Mac.
- Note that the shift of the bits is one position from right to left.

- After each LFSR simulation, the new rightmost bit is the pseudorandom bit for that step/iteration.

VII. Grading

Your project will be graded over 100 points. I will grade your project on the following criteria:

- **Cipher** (10 points) - Generating a new seed and pseudorandom bit based on the LFSR algorithm.
- **GenerateKeystream** (5 points) - Generating each seed and a (new rightmost)pseudorandom bit after each simulation and a keystream (a sequence of pseudorandom bits).
- **Encrypt** (12 points) - generating a ciphertext in bits from a given plaintext (in bits) using the keystream in 3 test cases (see sample run):
    * Plaintext length less than the length of the keystream
    * Plaintext length greater than the length of the keystream
    * Plaintext length the same as the length of the keystream
- **Decrypt** (3 points) - generating a plaintext in bits from a given ciphertext (in bits) using the keystream with the above test cases.
- **MultipleBits** (20 points) – generating each seed at the end of the last step, and an accumulated integer value gotten from the arithmetic operation per iteration.
- **EncryptImage** (30 points)-generating an encrypted image from an image file
- **DecryptImage** (20 points)-generating the original image from an encrypted image

VIII. Submission Requirements

Zip up your solution in a file called project3.zip. The zip file should contain at least the following files, with the csproj being in the root of the zip.

- Program.cs
- Lfsr.csproj
- Text file if you used generative AI

I will be testing your program on either a Mac or Windows machine, with secondary tests being run on Linux. You should ensure your program works on multiple platforms.

IX. Sample Runs (for Output Formatting)

- **Cipher**:

```
dotnet run cipher 01101000010 9
01101000010 – seed
11010000101  1
```

- **GenerateKeystream***:*

```
dotnet run GenerateKeystream 01101000010 9 10
01101000010 – seed
11010000101  1
10100001011  1
01000010110  0
10000101100  0
00001011001  1
00010110010  0
00101100100  0
01011001001  1
10110010010  0
01100100100  0
The Keystream: 1100100100
```

- **Encrypt** *:* This includes the 3 test cases for encryption.

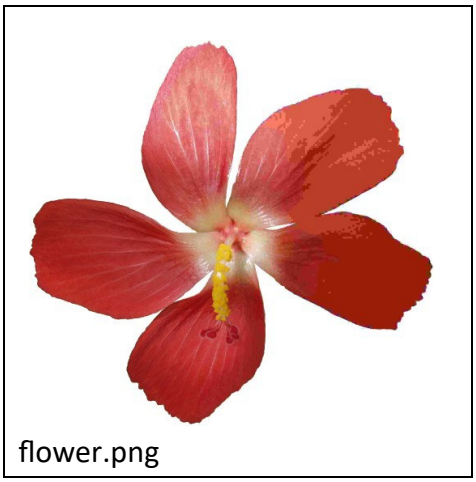
Case 1	dotnet run Encrypt 0100010000 The ciphertext is: 1000110100
Case 2	dotnet run Encrypt 0110010010001011 The ciphertext is: 0110011110101111
Case 3	dotnet run Encrypt 1110101 The ciphertext is: 1101010001

- **Decrypt**: It can be generated using the same test cases. However, we will have "dotnet run Decrypt". Descriptive comments should be different from the above. e.g. "The ciphertext is " should be replaced with "The plaintext is".

- **MutipleBits**: if p =5 and w =10; we have:

```
dotnet run MultipleBits 01101000010 9 5 10
01101000010 - seed
00001011001   25
01100100100   4
10010011110   30
01111011011   27
01101110010   18
11001011010   26
01101011100   28
01110011000   24
01100010111   23
01011111101   29
```

- **EncryptImage:**
  Although the aspect ratio in these results is not the same here due to “copy and paste” and the inclusion of text in each box; **Please make sure that the aspect ratio of all your images/results is the same.**



flower.png

dotnet run EncryptImage "flower.png" 01101000010100010000 15

flowerENCRYPTED.png

- **DecryptImage**:

dotnet run DecryptImage "flowerEncrypted.png" 01101000010100010000 15

flowerNEW.png