CS 241 Data Organization using C
Lab 7: LCG and Cipher

April 7 2020 (due on April 20 2020)

1 Project Overview

In this assignment, you will create an implementation of a Linear Congruential Generator to that can be used to create a sequence of "random" numbers. You will then use your LCG as part of a cipher algorithm to encrypt and decrypt text.

This assignment will give you some practice using structs, pointers, header files, and makefiles.

2 Linear Congruential Generator

I am giving you a header file `lcg.h` which defines a structure for an LCG and functions to manipulate it. You must implement these functions in `lcg.c`

2.1 LCG Initialization

Given the values

- m : a 64-bit positive integer used as an LCG modulus,
- c : a 64-bit positive integer used as an LCG increment.

we can define a Linear Congruential Generator:

$$X_{n+1} = (aX_n + c) \mod m$$

Where

- $X_0 = c$
- $a = 1 + 2p$, if 4 is a factor of m, otherwise, $a = 1 + p$.
- $p =$ (product of m's unique prime factors).
- $0 < m$
- $0 < a < m$
- $0 \leq c < m$

If the values given result in an invalid LCG, set all the fields to zero to indicate an error.

2.2 Finding Unique Prime Factors

1. Let *n* be the number of which to find the prime factors.
2. Start with 2 as a test divisor.
3. If the test divisor squared is greater than the current *n*, then the current *n* is either 1 or prime. Save it if prime and return.
4. If the remainder of *n* divided by the test divisor is zero, then:
   (a) The test divisor is a prime factor of *n*. Save it.
   (b) Replace *n* with *n* divided by the test divisor.
   (c) Repeat 4b until test divisor is no longer a factor of *n*.
5. If, in step 4, the remainder of *n* divided by the test divisor was not zero, then increment the test divisor.
6. Loop to step 3.

3 Cipher Program

Write a C program, `cipher.c`, that:

1. Reads characters from the standard input stream using the standard library function `getchar()`.
2. Determines whether each line encodes a valid cipher direction, key pair and data.
3. For each valid line use the described cipher algorithm either encrypt or decrypt the data. Send the result to the standard output stream.
4. If the line contains an error, then skip to the end of the line, print an error message, and read the next line.
5. *Output will be checked by an automated script and must adhere to a strict format.*
6. Your `cipher.c` should include `lcg.h` and use your LCG implementation to generate the pseudorandom numbers used in the cipher algorithm.

3.1 Cipher Data

The cipher algorithm used in this project:

- Encrypts plain text messages consisting solely of the printable ASCII characters.
- Decrypts ciphertext consisting solely of the printable ASCII characters.
- The *printable ASCII characters* are 8-bit codes in the range of values from 32 to 126 (00100000 through 01111110). See `http://en.wikipedia.org/wiki/ASCII`
- Any 8-bit sequences outside of this range constitutes invalid data.

3.2 Cipher Record Format

Each cipher record must be of the form:

Action	lcg_m	,	lcg_c	,	Data	\n
1 char	1-20 char	1 char	1-20 char	1 char	any number of char	1 char

Action: Must be either 'e' or 'd', specifying encryption or decryption respectively.

lcg_m: Specifies a 64-bit positive integer used for m of a Linear Congruential Generator. Must be decimal digits.

lcg_c: Specifies a 64-bit positive integer used for c of a Linear Congruential Generator. Must be decimal digits.

Data: Printable ASCII character data to be encrypted or decrypted. Can be empty or arbitrarily long. Note: with the given algorithm there will be no need to keep the full line of data in memory.

3.3 Algorithm Summary

1. Determine the Linear Congruential Generator specified by the given key. This is done only once per line of input.
2. Read 1 byte of data b
3. Generate random value x
4. Compute encrypted byte as b XOR (x mod 128)
5. Deal with any non-printable ASCII characters.
6. Print the resulting character(s) to the standard output stream.
7. Return to step 2 and continue until the end of the line

Note: (A XOR B) XOR B = A, so we are able to decrypt our message reapplying the same XOR operation. (When decrypting, you will deal with special characters first, then XOR to decrypt back to the original plaintext character.)

3.4 Initialization

Given a 128-bit symmetric key consisting of:

- m : a 64-bit positive integer used as an LCG modulus,
- c : a 64-bit positive integer used as an LCG increment.

Define a Linear Congruential Generator as described in section 2.1.

3.5 Reading Data Bytes

- Within each record, the data portion is the set of characters between the second comma and `'\n'`.
- When encrypting: Use `getchar()` to read 1 byte of data to encrypt.
- When decrypting: Reading in an encrypted byte may require reading 2 bytes from standard input since some character codes have 2-byte ciphertext representations.
- Any character in the data segment not in the range of printable ASCII characters (32 to 126) is an error.

3.6 Non-printable ASCII characters

This cipher algorithm can generate target bytes that are outside the range of printable ASCII.

- When encrypting, if a generated byte `e` is:

< 32	Replace with two bytes: `'*'` and `'?'+e`.
$= 127$	Replace with two bytes: `'*'` and `'!'`.
`='*'`	Replace with two bytes: `'*'` and `'*'`.

- When decrypting, if a generated byte `p` is a non-printing ASCII character, then print the specified error message and read to the end of line.

3.7 Output Format

- For every line of input, one line of output must be sent to the standard output stream.
- If the input line is invalid, then the output has the form:
  `("%5d) %s Error\n", inputLineNumber, trash)`
  where `trash` is any string no longer than twice the length any data part of the line.
- If the input line is valid, then the output has the form:
  `("%5d) %s\n", inputLineNumber, outStr)`
  where `outStr` is the encrypted or decrypted data.

Grading Rubric (total of 60 points)

-5 point : The programs do not start with a comment stating the students first and last name and/or the source files are not named correctly.

-5 points : Programs compile with warnings on moons.cs.unm.edu using `/usr/bin/gcc` with the `-Wall -ansi -pedantic` options (and options for any standard libraries you may want to use such as `-lm` for `math.h`). If you do use a standard library that requires a linking option, please include this as a note when you submit in Learn.

10 points : Output of running `lcgtest` matches `lcgtestout.txt`.

30 Points: The given test file: `cipherdata.in` contains 30 tests. For each failed test, one point is lost. Every line reported by `/usr/bin/diff your.output cipherdata.out` is a failed test.

5 Points: Must decrypt `novel.crypt`, a file containing an encrypted novel in html format. Success is easily checked by opening the output in an html viewer. If it looks good to the eye at the start, somewhere in the middle and end, it passes.

15 Points: Follows CS-241 Coding Standard: including quality, quantity and neatness of comments, no dead code, and Best Practices (functions not being too long, nestings not needlessly deep, and avoidance of duplicate code).