CS 241 Data Organization using C
Project 2: Cipher

Fall 2014

Project Overview

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 *lejo cipher algorithm*[1] to 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.*

Lejo Algorithm

Cipher Data

Lejo, the cipher 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.

[1]You won't find this algorithm on the internet.

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.

Algorithm Summary

1. Determine the Linear Congruential Generator specified by the given key. This is done only once per line of input.
2. Build $f : i \in [0, 27] \to k \in [0, 27]$, a one-to-one onto function.
3. Read 4 bytes of data.
4. Use f to copy the lower 7 bits from 4 source bytes, $s_0 \ldots s_{27}$ to the lower 7 bits of 4 target bytes $t_0 \ldots t_{27}$.
5. Deal with any non-printable ASCII characters.
6. Print the resulting characters to the standard output stream in the order: $T_1T_2T_3T_4$ where T_1 is one or two characters resulting from $t_0 \ldots t_6$, T_2 from $t_7, \ldots t_{13}$, etc.
7. Return to step 2 and *build a new map* for the next 4 bytes.

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:

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

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).

Bit Mapping

- The bit mapping is a one-to-one onto function

$$f : i \in [0, 27] \to k \in [0, 27]$$

- A new map is created for each 4-byte block using the specified LCG:
  - The first map uses $N_0 = c$ (the increment of the LCG) as the seed. The first map is built using N_0 through N_{27}.
  - The second map is built using N_{28} through N_{55}, etc.
- The map is used to copy the lower 7 bits from each of 4 source bytes to the lower 7 bits of each of 4 target bytes.
  - If encrypting, use the mapping: $f(i) = k$ to map $p_i \to e_k$.
  - If decrypting, use the inverse: $f^{-1}(k) = i$ to map $e_k \to p_i$.

There are 28 ASCII bits in each 4-byte block. The LCG is used to generate a different mapping for each block: $f : i \in [0, 27] \to k \in [0, 27]$.

- Lejo uses a LCG to build each mapping:

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

- The first plaintext bit, p_0, can be mapped to any random ciphertext bit $p_{f(0)}$, where $f(0) \in [0, 27] : f(0) = (X_n \mod 28)$.
- The second plaintext bit, p_1, can be mapped to any of the 28 cipher bits *except* $f(0)$.
- One way to exclude $f(0)$ from a random pick is:
  - Let $g(1) = (X_{n+1} \mod (28 - 1))$.
  - Find $f(1)$: Start at the first cipher bit, v_0 and count each unmapped cipher bit $e_0, e_1, \ldots e_k$ until a total of $g(1) + 1$ unmapped cipher bits have been counted. Then, $f(1) = k$.
  - More generally, $g(i) = (X_{n+i} \mod (28 - i))$.

Reading Data Blocks

- Within each record, the data portion is the set of characters between the second comma and `'\n'`.
- When encrypting:
  - Use `getchar()` to read 4 bytes of data or until `'\n'`.
  - If less than 4 bytes (and more than zero bytes) are read when `'\n'` is reached, then pad with `'\0'`.
- When decrypting:
  - Always populate an array of source characters with four character codes. This may require reading as many as 8 characters 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.

Non-printable ASCII characters

The bit mapping of the lejo 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:
  - =`'\0'` The byte had been generated during encryption as padding. Drop it from the plaintext output.
  - = any other non-printing ASCII character, then print the specified error message and read to the end of line.

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

54 Points: The given test file: `cipherdata.in` contains 27 tests. For each failed test, two points are lost. Every line reported by `/usr/bin/diff your.output cipherdata.out` is a failed test.

10 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.

36 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).