CSCI 330 Spring 2017 Assignment 3

Purpose

This program is meant to give students practice working on projects that contain multiple files of source code. It will require the students to construct a Makefile and appropriate header files. It will also include the use of `getopt` to handle optional command line parameters. Manipulating data read in from a file will also be introduced.

The following functions will be of use to you (`read` and `write` are mandatory).

- `read(2)`
- `write(2)`
- `getopt(3)`

Description

In this assignment, you will be creating an upgrade to the `cat` command you implemented in assignment 2. Let's call it `dog` because canines are so obviously better than felines. The features that will be added to this program include:

- The ability to specify a command line parameter (`-b x`) to change the size of the **b**uffer used with `read` and `write` to *x*.
- The ability to specify a command line parameter (`-n x`) to change the **n**umber of bytes read from each file to *ex*. (Instead of reading the entire file.)
- The ability to specify a command line parameter (`-c x`) to change the shift to *x* when applying a simple Caesar cipher to text within the file (described below).
- The ability to specify a command line parameter (`-r x`) to specify the shift to *x* when applying a similar **r**otation to binary data within the file.
- The ability to specify a command line parameter (`-x`) to output the data in the file as hexadecimal numbers.

Requirements

- The main loop should be in your `main` function in one source code file. The functions that perform the encryption should be found in another file. Their declarations will need to be in a header file so that they can be called from `main` even though their implementations are in another file.
- You must make a `Makefile` that has rules sufficient to compile your program when a user types `make` in the same directory as your source code. Only files that have changed or have had their dependencies change should be recompiled.
- You must use the system calls `read` and `write` to do the input and output for the program.
- Obviously, your source code must be well documented. If it is not, there will be a large point penalty. This requirement holds for all assignments, whether this line is in the writeup or not.

- The features enabled by the command line parameters listed in the description section above must be implemented.
  - Note: If both `-r` and `-c` are specified, the behavior can change depending on which is executed first. Don't allow a user to specify both: give them an error message if they try.
  - The other command line parameters can all be on at the same time without any problem. Make sure that they work together.

Caesar cipher

A Caesar cipher is a very simple method of encrypting data. It involves taking the letters of the input and shifting them upwards alphabetically by a number of letters. This number will be specified as a command line parameter. If the parameter has not been specified, do not perform any shift. If it is, then your output should be shifted by that number of letters. Only letters (uppercase and lowercase) should have any shift applied at all. Leave any other characters alone. You should assume that the files are using the ASCII encoding when writing this (`man 7 ascii`.

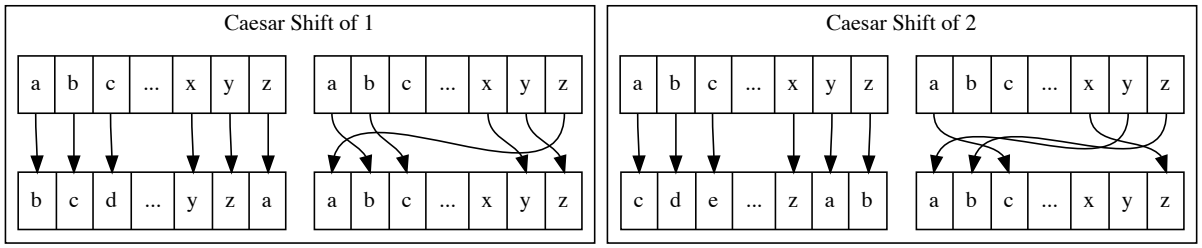


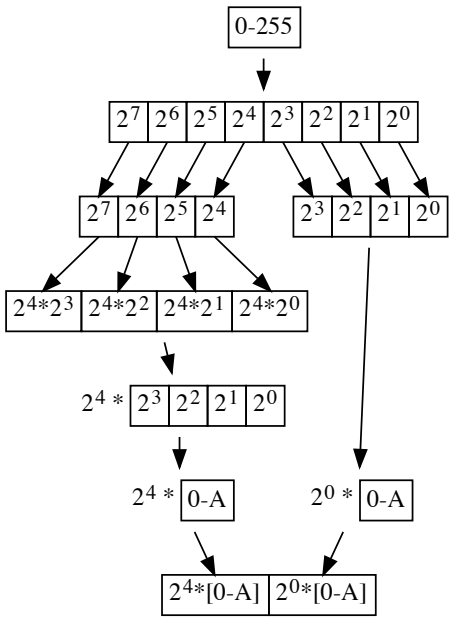
Figure 1: A diagram of the basic Caesar shift cipher for shifts of one and two.

Binary rotation

While the Caesar cipher above was interpreting the bytes that you read as text, changing only the alphabetical charaters, this option should apply to any binary value. The principle is the same, but instead of applying it to A-Z and a-z you apply it to any one-byte binary value from 0-255.

Hexadecimal Representation

All numbers are stored as binary bits on the computer. Each `char` is one byte, which is 8 bits. When dealing with files that are not meant to be read by a human (binary as opposed to ASCII), it can become convenient to encode the data in a format called hexadecimal. Hexadecimal, the base-16 number system, is useful precisely because it is based on a power of 2 ($2^4 = 16$), and so each hexadecimal digit represents 4 bits. Numbers represented as decimal (base-10) do not line up in the same way. Each byte is 8 bits, so it can be written as 2 hexadecimal digits. Your program should be able to output the data in this format (2 hex digits per input byte) whenever the `-x` command line parameter is enabled. Keep in mind that there will be two characters of output for every character of input in this mode.



Decimal	Hex	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Figure 2: A diagram showing how the bits of a byte can be represented in hexadecimal.

What to turn in?

When turning in the assignment, the following files should be included:

- The C++ source code file containing your `main` function.
- The C++ source code file containing the functions that implement the features enabled by command line parameters.
- A C++ header (`.h`) file containing the declarations for each of the functions implemented.
- A `Makefile` as shown in class. The rules should be set up so that only the files that have changed need to be recompiled.

The submission should be over Blackboard. Keep in mind that there is no credit for late work.