A Simple Text File Crypto System

CS 224

Due 11:59 pm, Friday, April 3, 2015

1 Introduction

For this project you will write a small C program that *encrypts* and *decrypts* text files using a short *key* specified by the user. The goal is to familiarize you with. . .

- performing I/O using `stdin`, `stdout`, and `stderr`,
- reading command line arguments via `argc, argv`, and
- using the shell's *file redirection* facilities.

2 Input/Output

The program takes two *command line arguments.* The first argument must be either an `e` or a `d` which specifies whether the program is *encrypting* or *decrypting.* The second argument is the encryption *key.* The program reads each input character from *standard input* (`stdin`) and writes the encrypted version of each character to *standard output* (`stdout`).

For example, here is how I could encrypt a simple text file `poem.txt` and store the result in `secret.txt`:

```
$ cat poem.txt
There are 10 kinds of people:
  Those that understand binary, and
  those that don't.
$ ./cypher e 'Beware the Jabberwock, my son!' < poem.txt > secret.txt
$ cat secret.txt
7n}tx&"(n&Qz"nltw,0srL1s*1!uI
b&lj#y&5}n"_"xqjx+$xm;%.|*#q";n&ypw
@5}n0^g#wnt-0h{;G#H
```

I can print out the original poem by decrypting using the same key:

```
$ ./cypher d 'Beware the Jabberwock, my son!' < secret.txt
There are 10 kinds of people:
  Those that understand binary, and
  those that don't.
```

If all goes well, the program should *exit* with an *error code* of zero. If there is an error (*i.e.,* the user does not provide the appropriate arguments), then print a "usage message" to *standard error* (`stderr`) and exit with a non-zero error code:

```
$ ./cypher
to encrypt: ./cypher e <key>
to decrypt: ./cypher d <key>
```

```
$ ./cypher x 'abcd'
first argument must be 'e' or 'd'!
to encrypt: ./cypher e <key>
to decrypt: ./cypher d <key>
```

3 Encryption Algorithm

We use a very simple encryption algorithm. First of all, we only alter printable ASCII characters whose encoded values lie in the range 32 to 126. Non-printable ASCII characters are output untouched. We encrypt the ith input character c using a key of length n as follows:

$$e = (c - 32 + \mathrm{key}[i \ \% \ n]) \ \% \ 95 + 32. \quad (1)$$

Note that the encrypted character e will also be a printable ASCII character. To decrypt e to obtain the original character c as follows:

$$c = (e - 32 - \mathrm{key}[i \ \% \ n] + 3 * 95) \ \% \ 95 + 32. \quad (2)$$

For example, consider the following command:

```
$ ./cypher e 'Beware the Jabberwock, my son!' < poem.txt > secret.txt
```

- the *key* is the string `"Beware the Jabberwock, my son!"` which has a length $n = 30$;
- the input characters c are read one at a time from `stdin` which is redirected from the file `poem.txt` in this case;
- the output encrypted characters e are written to `stdout` which is redirected to the file `secret.txt`.

3.1 Implementation Details

Use `c = fgetc(stdin)` to fetch the next input character `c` from `stdin`; When there are no more input characters, the constant `EOF` is returned. Declare `c` as an `int` (not a `char`) since it will need to hold values that can not fit in a single byte. Start the character counter i at zero and increment it for each input character. Use `fputc(c, stdout)` to write each output character.

4 Makefile

You are to include a `Makefile` in your submission that builds your application and also supplies (at least) the following *targets:*

`all:` (default) build `cypher` program.

`test:` runs `test.sh` script (provided by me); dependency `cypher`.

`clean:` removes build riffraff.

`clobber:` removes build riffraff and executable

`cypher:` compiles/links app.

`cypher.tar.gz:` creates compressed tar-ball archive containing: `Makefile, README, cypher.c, test.sh` and any other supporting files.

`achive:` triggers `cypher.tar.gz` creation

Make sure each target has the correct *dependencies.* For example, the `test` target should depend on `cypher` so that a fresh executable is created before the test is run:

```
$ make clobber
rm -rf *.o *~ *.dSYM cypher *.tar.gz
$ make test
gcc -c -g -Wall -std=c99 cypher.c
gcc -g -Wall -std=c99 cypher.o -o cypher
./test.sh
encrypting...
...
```

Please use macros like `CC` and `COPTS` where appropriate so the user can easily change compilers (*e.g.,* between `gcc` and `clang`) and the its options (*e.g.,* between `-g -Wall` for development and `-O3` for the release version).

5 What to submit

You will bundle your source code (name it `cypher.c`), `Makefile`, `test.sh` (test script I provide), and a `README` text file in an archive file and submit it electronically through the class web site. Your `README` file should contain the following information:

- Your name and email address.
- A brief description of what your program does.
- A description of how to build the executable from source.
- A description on how to use the program (perhaps an example or two).
- A list of all files in the archive.

Your program should compile and execute under Linux and OS X on the lab machines. The program is due by midnight on the due date. Have fun.