ENEE 140 Project 2: A Spy's Toolkit

Posted: Tuesday, March 22, 2016
Due: Monday, April 11, 2016 at 11:59 PM

Project objectives

1. Become familiar with basic file operations in C.
2. Learn how to manipulate command-line arguments in C.
3. Cement your understanding of arrays, conditional statements, and loops.
4. Learn how to complete a complex programming project in C by splitting the functionality into several code and header files.
5. Understand basic cryptographic techniques.

Project description

In this project, you will write a program for encrypting and decrypting text files. This program will take the names of the input and output files, the operating mode (encrypt/decrypt and the algorithm to use) and additional parameters as arguments provided on the command line. The program will be launched as follows:

```
./enee140_crypto plaintext_file cyphertext_file mode [additional_arguments]
```

where:

- `plaintext_file` is the name of the unencrypted file.
- `cyphertext_file` is the name of the encrypted file.
- `mode` is an integer number between 0 and 3 that specifies the operation to be performed.
- `additional_arguments` are additional integer arguments passed to the program. These additional arguments are needed only in modes 2 and 3 and they are described below.

Program structure

The functionality of the program will be implemented in three files. `crypto.h` is a header file that declares four functions, each corresponding to one of the four requirements described below. These four functions represent the *advanced programming interface (API)* of your program. In this header file, you may also declare some constants you use in your program. `crypto.c` is a C source file

where you define (implement) these functions, as well as any other helper functions you may find useful. `enee140_crypto.c` is a C source file that includes the `main()` function of your program. In the `main()` function, you will process the command-line arguments and you will invoke (call) the functions defined in `crypto.h`, depending on the mode requested.

Hint: In order to interpret the command-line arguments, you will need to convert the elements of the `char *argv[]` array to integers. You can do this using the `atoi()` function from `<stdlib.h>` or the `sscanf()` function from `<stdio.h>`. The second alternative is preferred because it returns an error when the conversion fails.

Cryptographic functionality

Your crypto program should perform the 4 operations described below. Implement *each operation as a separate C function* that you will invoke from your `main()` function. In some cases, you may find it helpful to implement *helper functions*, which provide functionality that is useful for several tasks.

1. This functionality corresponds to `mode = 0`. Implement a function with the following declaration:

   ```
   int encrypt(FILE *in, FILE *out);
   ```

   This function will encrypt the content of the text file `in` using a *substitution cypher*, and it will write this encrypted content to the text file `out`. A substitution cypher replaces each character with a corresponding character from a substitution table. To implement the substitution table, define three arrays in your program:

   ```
   char lowercase[] = { /* add lowercase letters */ };
   char uppercase[] = { /* add uppercase letters */ };
   char numbers[] = { /* add digits */ };
   ```

   Initialize the `lowercase[]` array with the list of 26 lowercase characters (e.g. `'a', 'b', 'c', ...`), in an order that is *different from their alphabetical order*. Similarly, initialize the `uppercase[]` array with the uppercase characters (e.g. `'A', 'B', 'C', ...`) not ordered alphabetically. Initialize the `numbers[]` array with the 10 digits (`'0', '1', ... '9'`), unordered.

   You will substitute each character in the original file with the character in the corresponding array located on the position that corresponds to the position of the original character in the alphabetical (or numerical) order. For example, you will substitute `'a'` with `lowercase[0]`, because `'a'` is the first lowercase character. Similarly, you will substitute `'b'` with `lowercase[1]`, `'C'` with `uppercase[2]` and `'5'` with `numbers[5]`. All the characters that are not in the substitution table (e.g. whitespace, special characters) should remain unchanged.

   In the output file, start by writing the substitution table. Write two characters per line: the original character and the substituted character. Write the lowercase table first, followed by the uppercase table, followed by the numbers table. For example, this section of the output file could look as follows (parts of this table have been replaced with ... for brevity):

   ```
   a z
   b l
   ```

```
...
z n
A F
B C
...
Z N
0 7
1 2
...
9 8
```

Immediately after the substitution table, write the the encrypted content of the file, generated using the substitution rules described above.

2. This functionality corresponds to `mode = 1`. Implement a function with the following declaration:

```
int decrypt(FILE *in, FILE *out);
```

This function will decrypt the content of the file `in`, which was encrypted using the substitution cypher described in the previous requirement, and it will write this decrypted content to the text file `out`. The function will start by reading the substitution table from the file `in` and will use it to decrypt the rest of the file. For example, assuming that you have read the substitution table into three arrays named `lowercase`, `uppercase` and `numbers`, you will substitute `lowercase[1]` with `'b'`, `uppercase[2]` with `'C'` and `numbers[5]` with `'5'`. Write only the decrypted file content into the output file; do not write the substitution table.

Important: Do not assume that the file uses the same substitution table as the one you have used for encryption. You should read the substitution table from the input file. One of the ways we will test your program for correctness will be by trying to decrypt files created using programs from other students in ENEE 140.

3. This functionality corresponds to `mode = 2`. Implement a function with the following declaration:

```
int encrypt2(FILE *in, FILE *out, int npos, int transposition[]);
```

This function will add a *transposition cypher* to the encryption you have implemented in requirement 1. A transposition cypher changes the order of characters in the output file. A block of `npos` characters is encrypted by moving the character on position `transposition[i]` to position `i`. This operation is repeated for the next block of `npos` characters. If the last block in the file has fewer than `npos` characters, pad the block with spaces and then apply the transposition. For example, if `npos = 5` and `transposition[] = {2, 4, 0, 1, 3}`, the output of the transposition cypher is as follows:

Input: `"Tudor␣D."`
Output: `"drTuo.␣␣D␣"`

You will provide the transposition table on the command line, using the additional arguments of the `enee140_crypto` program:

```
enee140_crypto plaintext_file cyphertext_file mode npos t1 [t2 [...]]
```

After the common command-line arguments, the program will receive an integer `npos`, and then it will receive `npos` more integers that define the transposition table. You may assume that `npos` ≤ 5. The program will print an error message and will exit immediately if the incorrect number of arguments was provided on the command line or if the transposition table is invalid (e.g. it is not a permutation of `{0 .. npos-1}`).

First, encrypt the file using the substitution cypher you have implemented in requirement 1. Then apply the transposition cypher to this content and save the result to the output file. You should apply the transposition cypher to the entire file, including the substitution table.

Hint: Consider saving the output of the substitution cypher to a temporary file and to apply the transposition cypher to the content of this file. You can create a temporary file using the `tmpfile()` function, as follows:

```
// Create a temporary file
if ((tmp = tmpfile()) == NULL)
    /* could not create tmp file => add error handling code here */
```

After you have written this file, you will have to start reading the file from the beginning by invoking `rewind(tmp)`. These two functions are in the `stdio.h` header.

4. This functionality corresponds to `mode = 3`. Implement a function with the following declaration:

   ```
   int decrypt2(FILE *in, FILE *out, int npos, int transposition[]);
   ```

   This function will decrypt the content of the a file `in`, which was encrypted using the method described in requirement 3, and it will write this decrypted content to the text file `out`. The transposition table will be provided on the command line, in the same way it was provided for encryption. First, decrypt the file using the transposition cypher, by moving the character on position `i` in a block to position `transposition[i]`. Then, apply the substitution decryption that you have implemented in requirement 2 and save the result to the output file.

5. **15 bonus points** Implement two more operations that add encryption using Caesar's code to the transposition and substitution cyphers implemented before. `mode = 4` will perform encryption and `mode = 5` will perform decryption. Caesar's code is a simple case of a substitution cypher where each letter is shifted k positions down the alphabet (for example, if $k = 2$, `'A'` becomes `'C'`). When shifting the letters from the end of the alphabet you must wrap around to the beginning (for example, `'z'` becomes `'b'`). While in the lab report from Week 4 we implemented a code with $k = 3$, in this project k should be specified as an additional command line argument.

Project requirements

1. You must program in C and name your program files `crypto.c`, `crypto.h` and `enee140_crypto.c`. Templates for these programs are included at the end of this document (you do not have to use them, but they may provide some hints).
2. Your programs must compile on the GRACE UNIX machines using:

   ```
   gcc -o enee140_crypto  crypto.c enee140_crypto.c
   ```

3. Your programs must implement all the steps described above correctly.
4. Your programs must be readable to other programmers (e.g. the TAs).
5. Before you submit, you must archive all the programs you wrote into a single file, called `enee140_crypto.tar.gz`, using the `tar` command. For example, if all your program files are in the current directory, you can use the following UNIX command:

   ```
   tar cvfz enee140_crypto.tar.gz crypto.c crypto.h enee140_crypto.c
   ```

   Submit the `enee140_crypto.tar.gz` file using the following command:

   ```
   submit 2016 spring enee 140 AAAA 102 enee140_crypto.tar.gz
   ```

   Note: you must replace `AAA` with your own section number (0101, 0102, etc.)

Grading criteria

Correct functionality on correct inputs:	60%
Appropriate error messages on incorrect inputs:	20%
Good coding style and comments:	20%
Late submission penalty:	-40% for the first 24 hours -100% for more than 24 hours
Program that does not compile on GRACE:	-100%
Wrong file names (other than `enee140_crypto.tar.gz`, `crypto.c`, `crypto.h`):	-100%
Bonus (Caesar's code):	15%

Program templates

You can start from the following templates (also available in the GRACE class public directory, at `public/projects/project2`).

Template for `crypto.c`

```c
/*
 * crypto.c
 *
 *  Created on: Mar 29, 2014
 *      Author: tdumitra
 */

#include "crypto.h"
#include <ctype.h>


// Substitution table, for use in encryption
char lowercase[] = { /* add lowercase letters */
};
char uppercase[] = { /* add uppercase letters */
};
char numbers[] ={ /* add digits */
};


int
encrypt(FILE *in, FILE *out)
{
}


int
decrypt(FILE *in, FILE *out)
{
}


int
encrypt2(FILE *in, FILE *out, int npos, int transposition[])
{
}


int
decrypt2(FILE *in, FILE *out, int npos, int transposition[])
{
}
```

Template for `crypto.h`

```
/*
 * crypto.h
 *
 *  Created on: Mar 29, 2014
 *      Author: tdumitra
 */

#ifndef CRYPTO_H_
#define CRYPTO_H_

#include<stdio.h>

#define MAXPOS 5

int encrypt(FILE *in, FILE *out);

int decrypt(FILE *in, FILE *out);

int encrypt2(FILE *in, FILE *out, int npos, int transposition[]);

int decrypt2(FILE *in, FILE *out, int npos, int transposition[]);


#endif /* CRYPTO_H_ */
```

Template for `enee140_crypto.c`

```c
/*
 * enee140_crypto.c
 *
 *  Created on: Mar 29, 2014
 *      Author: tdumitra
 */


#include "crypto.h"
#include <stdio.h>
#include <stdlib.h>


int
main(int argc, char *argv[])
{

        return 0;
}
```