CSCI 151 Fall 2018
Homework 2

For this assignment, you will write a program that can help you encrypt and decrypt files on your computer. Like the previous homework, the program will have a main menu, which will have the following options:

- `e`: encrypt a file

  Ask the user the name of the file to encrypt. If the file doesn't exist, or can't be opened, print an error message, then go back to the menu. Ask the user the name of the new file encrypted file. If it can't be opened, print an error message, and go back to the menu.

  If both files are opened successfully, read the characters from the first, file, encrypt them, and write them into the second file.

- `q`: quit

  When the user quits the program, it should just print a message and exit.

- `d`: decrypt a file

  Ask the user the name of the file to decrypt. If the file doesn't exist, or can't be opened, print an error message, then go back to the menu. Ask the user the name of the new decrypted file. If it can't be opened, print an error message, and go back to the menu.

  If both files are opened successfully, read the characters from the first, file, decrypt them, and write them into the second file.

- If a character other than `e`, `q`, or `d` is entered, the player should get an error message and the menu should be presented again. The user is allowed to enter words that begin with the required letter; for example, `quit` or `qqqq` are both legal entries for `q`, and your program must handle this. However, your program may assume that the user will never enter a single input longer than 256 characters (for a filename, password, or command).

Rules

This programming assignment will be graded on style and structure, as well as correctness. First, the use of `goto` is prohibited. There should be comments and indentation, variables and functions should have meaningful names, and the program should have a clear, simple structure.

The programming assignment must be individual work. Do not discuss it with your classmates. Do not share your solution with anyone, in person or electronically, until the end of the semester. Do not discuss the assignment online, or copy online solutions. If you are having trouble, please ask one of your instructors or TAs.

Your program will be run through an automated plagiarism checker. If you are found to have cheated on any single programming assignment, **you will receive a 0 on ALL the programming assignments, past and future**, so consider this carefully. Late work on this assignment will be accepted, with a penalty of 20% per day.

Encryption mechanism

The cipher I have developed for this homework is reversible, which means the decryption can be done by reversing the steps of the encryption. It is slightly more complex than the *Vigenere* cipher, and much less complex than *RC4*. It involves a key file and a password.

- The keyfile is a file that contains 255 unique characters (all the ascii characters) in some scrambled order. You will be given a keyfile.
- The input file may be any length.
- The password may be any length up to 255.

The encryption process:

- Take the first character from the plain text file
- Add it to the first character from the password
- Take modulo 255 of the sum to get a number between 0 and 254
- Use that number as an index into the keyfile to find the encrypted character.
- Keep going until the end of the plain text file (add second char from file to second char of password, and so on)
- If the end of the password is reached, start over at the first character of the password.

The decryption process is just the opposite:

- Take the first character from the encrypted file.
- Find its index in the keyfile
- Take that index and subtract the first password character from it
- Take modulo 255 of that result to get the plain text character
- Keep going until the end of the encrypted file (using second char of password, and so on)
- If the end of the password is reached, start over at the first character of the password.

For example, suppose the original file contains "`TEST TEXT`" and your password is "`pass`", and the keyfile contains the values as shown below:

Plain text	T (84)	E (69)	S (83)	T (84)	(32)	T (84)	E (69)	X (88)	T (84)
Password (repeated as needed)	p (112)	a (97)	s (115)	s (115)	p (112)	a (97)	s (115)	s (115)	p (112)
(sum of 2 chars) % 255	196	166	198	199	144	181	184	203	196
Encrypted text (from keyfile)	:	X	8	7	N	I	F	3	:

144		166		181	182	183	184		196	197	198	199		203	
...	N	...	X	...	I			F	...	:		8	7	...	3	...

Encrypted text	:	X	8	7	N	I	F	3	:
Index from key file	196	166	198	199	144	181	184	203	196
Password (repeated as needed)	p	a	s	s	p	a	s	s	p
(index – pass char) % 255 -> original	T	E	S	T		T	E	X	T

Programming requirements

You must write functions for doing the encryption and decryption, with the following signatures:

```
void encrypt( FILE *fin, FILE *fout, char subarr[], char pass[] );
void decrypt( FILE *fin, FILE *fout, char subarr[], char pass[] );
```

The input and output files should be opened and checked for validity before calling encrypt or decrypt. You must pass in the character substitution array (keyfile info) and password.

Another function that must be written is a function for reading the keyfile, (hardcode the filename to "`key.255`"). You may choose the details of that function yourself.

Before showing the user the menu, you must ask the user for the password. The same password is used to encrypt/decrypt all the files during one session.

Programming details

This program, like many of the exercises you have done, involves you treating ascii characters as numbers. But we use a keyfile of length 255, and half of those values are considered to be negative numbers (128-254), if not carefully cast. Make sure you consistently use chars as the values you deal with, AND whenever the chars are used as their integer values, cast them as `(unsigned char)` so they are not seen as negative, for example:

```
enc_key[(unsigned char) c] = i;
dec_key[(unsigned char) i] = c;
```

It is not required for you to understand why this happens, but those of you who do want to know more, please see the notes at the end of this file.

The encryption and decryption algorithms require that characters from the keyfile must be looked up in 2 different ways:

- know the index – get the character (for encryption)
- know the character – get the index (for decryption)

It is recommended (but not required) that you store the keyfile data in both ways, in 2 character arrays, each of length 255. Since the keyfile contains 255 distinct values, which are 0 to 254, this is possible. Suppose the keyfile was just 8 characters long, containing the values 0 to 7: `43756201`, you would create 2 arrays, like this:

0	1	2	3	4	5	6	7
4	3	7	5	6	2	0	1

0	1	2	3	4	5	6	7
6	7	5	1	0	3	4	2

Newline characters are even more troublesome for this program than the last one, and you must keep them in mind. A filename cannot have a newline at the end; the password should not include a newline either.

Example User I/O

The following exchange shows correct behavior for the program. The red shows user input. As in the previous homework, please include the word shown in bold caps in your program's output. Each line must end in a newline, including the last line, and do not include any extra lines.

WELCOME to the encryption service.
ENTER your password
abc123
MENU: <e>ncode, <d>ecode, or <q>uit
e
ENTER a file to encrypt
cats.c
ENTER a filename for the encrypted file.
cats.c.enc
MENU: <e>ncode, <d>ecode, or <q>uit
d
ENTER a file to decrypt
cats.c.enc
ENTER a filename for the decrypted file.
cats2.c
MENU: <e>ncode, <d>ecode, or <q>uit
v
UNRECOGNIZED v
MENU: <e>ncode, <d>ecode, or <q>uit
eeeee
ENTER a file to encrypt
nofile.txt
CANNOT open nofile.txt
MENU: <e>ncode, <d>ecode, or <q>uit
quit
BYE!

Extra (not required) information about negative numbers

The `char` type is 1 byte (8 bits) long, so it can represent the numbers 0-255. The simple ascii characters are in the range 0-127 (`00000000-01111111`). We want to use the next set of values, 128-255, or all the values where the most significant bit is 1 (`10000000-11111111`). Doing this gives a wider range of characters used in the encrypted file.

The problem is when a `char`, an 8-bit value, is interpreted as a number. The most significant bit is used to indicate a negative number when it is 1. You may try the following code:

```
char c = 129;
printf( "%c %i %i", c, c, (unsigned char) c );
```

You can see that even though you set the value to 129 (`10000001`), it gets printed as -127. Only if you specifically say `(unsigned char)`, can all 8 bits get included as part of the number. Another question is, if a 1 in the first bit means negative, then why is `10000001` printed as -127 and not -1? Computers use a representation called *two's complement* for negative numbers. You may read about this online.

So anywhere you want to use a `char` variable as a full 8-bit value, you must cast it to `(unsigned char)`.

You might have noticed that the keyfile used in this homework is only 255 (0-254) characters long, rather than 256. This is because the value of the `EOF` constant is 255 (`11111111`), so it has to be excluded from being in the plaintext and encrypted files, because it is returned by the `getc` family of functions. These functions all return a `char`, so a character had to be reserved to indicate the end of file, and 255 was chosen by the developers since it is a special value, and not in the ascii range. If I hadn't left out that character from the keyfile, then this same encryption mechanism could be used for any file, not just ascii text files. Some binary files may include characters with the value 255 in their contents, so `getc` functions are not appropriate for dealing with them. We haven't discussed those issues yet in this class.