Homework #3

Overview

For security, important files and message are often *encrypted* to prevent the bad guys from seeing their contents. Later, a *decryption* algorithm is used to recover the data. For each student we have encrypted a file containing a secret message. Your task is to create a program to decrypt the file and discover the secret message.

We are using a different encryption algorithm for each student so each student will create a unique decryption program. The encryption / decryption algorithm is based on the first 3 letters of your last name. It is discussed below.

The Decryption Algorithm

In this assignment, you are to implement the decryption algorithm associated with your username.

The key to your decryption algorithm is the first three letters of your last name. We are getting your name from the PSU registration system. [Vu, Hanh→vuh]

There are 3 stages to the decryption scheme. The first letter of your key determines the first stage. The second letter of your key determines the second stage. The third letter of your key determines the third stage.

First stage

For every 4 bytes of the file **<c0**,**c1**,**c2**,**c3>**, implement the following based on the first letter of your key:

‘a’ to ‘d’:	Swap bytes **c0** and **c3**.
‘e’ to ‘h’:	Swap bytes **c1** and **c2**.
‘i’ to ‘l’:	Swap bytes **c0** and **c2**.
‘m’ to ‘p’:	Swap bytes **c1** and **c3**.
‘q’ to ‘t’:	Swap bytes **c2** and **c3**.
‘u’ to ‘z’:	Swap bytes **c0** and **c1**.

Second stage

For every byte of the file with bits **<b7**,**b6**,**b5**,**b4**,**b3**,**b2**,**b1**,**b0>**, implement the following based on the second letter of your key:

‘a’ to ‘d’:	Swap bits **b7** and **b1**, Swap bits **b6** and **b0**.
‘e’ to ‘h’:	Swap bits **b5** and **b3**, Swap bits **b4** and **b2**.
‘i’ to ‘l’:	Swap bits **b7** and **b3**, Swap bits **b6** and **b2**.
‘m’ to ‘p’:	Swap bits **b5** and **b1**, Swap bits **b4** and **b0**.
‘q’ to ‘t’:	Swap bits **b7** and **b5**, Swap bits **b6** and **b4**.
‘u’ to ‘z’:	Swap bits **b3** and **b1**, Swap bits **b2** and **b0**.

Third stage

For every 4 bytes of the file **<c0**,**c1**,**c2**,**c3>**, implement the following based on the third letter of your key, <k3>:

‘a’ to ‘d’:	XOR bytes **c0** and **c3** with **k3**.
‘e’ to ‘h’:	XOR bytes **c1** and **c2** with **k3**.
‘i’ to ‘l’:	XOR bytes **c0** and **c2** with **k3**.
‘m’ to ‘p’:	XOR bytes **c1** and **c3** with **k3**.
‘q’ to ‘t’:	XOR bytes **c2** and **c3** with **k3**.
‘u’ to ‘z’:	XOR bytes **c1** and **c0** with **k3**.

Additional Program Specifications

If the file size is not evenly divisible by 4 then, for the bytes at the end of the file that are not in a complete 4 byte block, the first and third stages are not to be applied.

Your program should take exactly one command line argument: the name of the file to decrypt. It should print the decrypted contents to stdout.

Hints and Suggestions

Use **argv** to obtain the filename.

You may use **fopen**, **fseek**, **ftell**, **malloc/calloc**, **fread**, and **fclose** to read the file into a buffer in memory.
Do not use the string libraries since NULL bytes can and will be generated as part of the encryption/decryption process.

The utility command "xxd" will print the contents of a file in hexadecimal. For example:

```
xxd filename
```

To see the file in binary:

```
xxd –b filename
```

(You can also use "od –x" or "hexdump", but watch out; "od" may perform byte reordering.)

For the second stage, the bit shift operator and bit masks are recommended for implementing the swap.

Some Test Data

To help you debug your code, we took a file containing this *plaintext*: "The quick brown fox jumped over the lazy dog!!!\n" and encrypted it with your 3 letter key, giving the encoded *ciphertext* version. You can retrieve this file. You now have a **< plaintext, ciphertext >** pair that you can use to test your algorithm.

[With only one pair such as this, do you think could figure out the key?]

The cipher text for your key is stored in a file whose name is the first three characters in your last name and stored here:

```
cs.pdx.edu/~harry/cs201/hw3/xxx
```

where *xxx* is the file name, i.e., the first 3 characters of your last name (all lower case).

To download a file onto a linuxlab machine, you can use the command **wget**. For example, you can get the ciphertext file discussed in the next sections with this command:

```
wget  http://cs.pdx.edu/~harry/cs201/hw3/por
```

Since the PSU file systems are shared, you should be able to just copy your file directly with:

```
cp  ~harry/public_html/cs201/hw3/xxx  .
```

The Secret Message

There is also a second file for each student containing a secret message. The file name is the same, except with a "2" appended, so you can get it with:

```
      wget http://cs.pdx.edu/~harry/cs201/hw3/xxx2
```

or:

```
      cp  ~harry/public_html/cs201/hw3/xxx2  .
```

Example Usage Session

```
LINUX:/u/harry % wget http://cs.pdx.edu/~harry/cs201/hw3/por
… 'por' saved [48/48]

LINUX:/u/harry % ll por
-rw------- 1 harry faculty 48 Nov 14 15:52 por

LINUX:/u/harry % od -x por
0000000 6145 4a35 1153 5739 007a 0201 617e 770d
0000020 6166 7e28 206a 573d 1d56 4661 6167 5600
0000040 6147 4a35 384e 5208 1502 461d 4b12 1271
0000060

LINUX:/u/harry % hw3 por
The quick brown fox jumped over the lazy dog!!!
```

Trace of Example Usage Session

Here is the ciphertext above dumped out in hex bytes within my program. Note the byte ordering compared to the output of "od -x":

```
45 61 35 4a 53 11 39 57 7a 00 01 02 7e 61 0d 77 66 61 28 7e 6a 20 3d 57 56 1d
61 46 67 61 00 56 47 61 35 4a 4e 38 08 52 02 15 1d 46 12 4b 71 12
```

Because my last name is "`Porter`", in the first stage, I implement the corresponding algorithm for '`p`', which is swap bytes **c1** and **c3** in each 4 byte block. After, the first stage, this is the result in hex bytes:

```
45 4a 35 61 53 57 39 11 7a 02 01 00 7e 77 0d 61 66 7e 28 61 6a 57 3d 20 56 46
61 1d 67 56 00 61 47 4a 35 61 4e 52 08 38 02 46 1d 15 12 12 71 4b
```

For the second stage, I implement the corresponding algorithm for '`o`', which is to swap bits **b5** with **b1** and **b4** with **b0** in each byte. After the second stage, this is the result in hex bytes:

```
54 68 17 52 71 75 1b 11 6b 20 10 00 6f 77 1c 52 66 6f 0a 52 6a 75 1f 02 65 64
52 1d 76 65 00 52 74 68 17 52 6c 61 08 0b 20 64 1d 15 21 21 53 78
```

Finally, for the third stage, I implement the corresponding algorithm for '`r`', which is to XOR bytes **c2** and **c3** with '`r`' (`0x72`) in each 4 byte block. After the third stage, this is the result in hex bytes:

```
54 68 65 20 71 75 69 63 6b 20 62 72 6f 77 6e 20 66 6f 78 20 6a 75 6d 70 65 64
20 6f 76 65 72 20 74 68 65 20 6c 61 7a 79 20 64 6f 67 21 21 21 0a
```

When these bytes are output as a string, it outputs "`The quick brown fox jumped over the lazy dog!!!\n`".

Homework #3

Submission

Send a single email to our grader. Include as a single attachment, the file **hw3.c** with

Subject: CS201, HW3, KEY=<first 3 letters of your last name>, <your name>

Grading

Your program will be compiled with this command on the PSU linux lab machines:

```
gcc -Wall -O1 —m64 hw3.c -o hw3
```

Your program should work correctly. Therefore, your grade will be based on its simplicity, clarity, and efficiency.

Your program should handle all conditions and print reasonable error messages if it is not used correctly. All error output should go to stderr. Nothing should go to stdout except the decrypted output.

It seems likely that you can develop and test the program on your own computer and it will run identically on the PSU Linuxlab machines, but it is up to you to make sure it runs correctly on the PSU Linuxlab machines.