Home Exam 2 for PG3400 - Basic C Programming

Kjetil Raaen &
Vamsidhar Reddy Gaddam

October 2015

Abstract

In the following document, you will find the question, hints and instructions for the second Home Exam for PG3400. This exercise helps you in getting started with libraries, string functions, file operations and other manipulations. For the sake of simplicity we are working with only the 26 English alphabet characters without accents and special characters!

1 Instructions

- You are encouraged to discuss the problem and possible solutions, but your code is expected to be independent or in groups of 2.
- Discussions on the forum might be particularly useful since I follow them too.
- The deadline is 15th of November and it is enforced by the system.
- The style of the code will also be considered in grading.

2 Introduction

This exercise is all about fun and encryption. Let us have a look at some cool things.

In communications and information processing, code is system of rules to convert information - such as a letter, word, sound, image, or gesture - into another, sometimes shortened or secret, form or representation for communication through a channel or storage in a medium [FROM WIKIPEDIA].

Therefore, I introduce two terms to you in the exam. One is *Encoding* and the other is *Decoding*

- The process of encoding converts information from a source into symbols for communication or storage.

- Decoding is the reverse process, converting code symbols back into a form that the recipient understands.
- Source: Wikipedia

However *Cryptography* has taken coding to a whole new level for the sake of hiding information. Traditionally it has been used to send secret messages from a source to a destination. The message can be decoded only with the help of a *key,* which is assumed to be known at both ends. This ensures that even if the coded message is broadcast openly through a channel, only the holders of the key are able to decode the true message.

In modern times, almost all data transferred across is coded in some or the other way to keep the messages safe and secure. In the current exercise, we will have a deep look at how this process works!

In the same exercise, we will try a collaborative approach. The idea is to use your code to create a library and distribute the library to your friends [*NOT THE SOURCE CODE*]. Your friends can try to use your library against their code to verify the standard functionality. In the final submission, I expect you to include the names of the library creators you tested your code against.

3 Library Creation

There are several ways that a developer can help other developers to reuse implementations. Two most common ways of doing this are by providing the entire source code and providing libraries. Distributing libraries implies that the developer must provide interfaces that are linked against the library. You can design the interface for your program in the following way.

You can have a header file named - 'secretCoder.h'. The contents of the header file could be:

```
/* Some documentation regarding inputs to the encode function and what to expect as
 * output. Note that I need two outputs from the encoder - One is the encoded
 * stream, the other is the status if success or fail. Hence, I can use two
 * different forms. I am placing both the forms here.
 *
 */

char *encode(const char *inputMessageFile, const char *keyFile, int *status);
// This form returns char * array of encoded stream and status into the
// reference.

int encode(const char *inputMessageFile, const char *keyFile,
           char *encodedStream);
// This form returns the status and takes the reference to the encoded stream as
// an argument

/* Some documentation regarding decode function and what to expect as output.
 * This functions pretty much the same as encode function
 */

char *decode(const char *inputCodeFile, const char *keyFile, int *status);

int decode(const char *inputCodeFile, const char *keyFile, char *decodedMessage);

// example usage :
//[1] decodedMessage = decode("myCode.txt", "hotelCalifornia.txt", &status);
//[2] status = decode("myCode.txt", "hotelCalifornia.txt", decodedMessage);
```

In addition you can have a source file[or a few] named - 'secretCoder.c' where you implement the functionality. You should be really detailed in your header as to what your implementations expect. Imagine someone who is not looking at your actual code trying to use it. You could also explicitly say that you can output your coded/decoded output to a textfile within your implementation and all you need is a file name. **Just be creative and be precise!**

3.1 static library

You can create a static library from your source code using the following commands.

```
user@machine $ gcc -c -g secretCoder.c -o secretCoder.o
user@machine $ ar rcs libsecretCoder.a secretCoder.o
```

You can distribute secretCoder.h and libsecretCoder.a to your friends. In their implementation they will just use the functions as defined in the header. The source must be compiled as following.

```
user@machine $ gcc -L. main.c -o main -lsecretCoder
```

Here '-L' is used to give a search directory, which in this case is '.'[the current directory- where the library is].

4 Encoding

The process of encoding is to convert any input text message into a code by using a key. In the current assignment, your input text message can be any text file and the key can be selected from one of the songs provided. You use the song as a key. The process of using it is described beneath.

You have a song library included with your home exam. It has about 121 plain text files with lyrics of famous songs from various artists. The title of the song is reflected on the file name of the text file. Inside every text file, you can be sure that you will find all the alphabets.

4.1 Task - Input Key preprocessing

The lyrics contain all sorts of characters. Your initial task is to preprocess the lyrics and create your own lyrics in the memory with the following conditions. Please be careful that only your key is filtered, not the message. The process of encoding should not usually lose information unless explicitly mentioned! In the current assignment, we do not intend to lose any information in the message.

The filtered key should have the following requirements:

- It should contain only lower-case characters[a-z]
- All the upper-case characters are converted to lower-case
- It should not contain any spaces, line-breaks, punctuation marks
- It should not contain any numeric character[0-9]

Key file:

```
She's got a smile that it seems to me
Reminds me of childhood memories
Where everything
Was as fresh as the bright blue sky

Now and then when I see her face
She takes me away to that special place
And if I stared too long
I'd probably break down and cry

Sweet child o' mine
Sweet love of mine

She's got eyes of the bluest skies
As if they thought of rain
I'd hate to look into those eyes
And see an ounce of pain

Her hair reminds me of a warm safe place
Where as a child I'd hide
And pray for the thunder and the rain
To quietly pass me by

[3x]
Sweet child o' mine
Sweet love of mine

[4x]
Where do we go?
Where do we go now?
Where do we go?
Sweet child o' mine

abcdefghijklmnopqrstuvwxyz
```

Filtered output:

```
shesgotasmilethatitseemstomeremindsmeofchildhoodmemorieswhereeverythingwasasfreshasthebrightblueskynowand
thenwheniseeherfaceshetakesmeawaytothatspecialplaceandifistaredtoolongidprobablybreakdownandcrysweetchild
omineSweetloveofmineshesgoteyesofthebluestskiesasiftheythoughtofrainidhatetolookintothoseeyesandseeanounc
eofpainherhairremindsmeofawarmsafeplacewhereasachildidhideandprayforthethunderandtheraintoquietlypassmeby
xsweetchildominesweetloveofminexwheredowegowheredowegonowwheredowegosweetchildomineabcdefghijklmnopqrstuv
wxyz
```

PS: Please note that the above output has no line-breaks. It is just split as different lines for printing it in this file.

4.2 Task - Encoding

Now, let us assume that you have a message file that looks as following.

inputMessage.txt:

```
There is a concert today at telenor Arena. The most awesome band Eagles are
playing at it. Let us meet up downtown and head there.
```

Your encoding should produce the following output, encodedText.txt:

```
[-14][783][345][123][847] [12][309] [756] [827][736].....
```

This encoding must follow certain rules:

- The integers are the positions of characters in the filtered key text[not the original]
- Capital letters in the message are coded to be negative integers.
- All the punctuation marks are retained - spaces, linebreaks, commas, full-stops.
- All integers must be enclosed in [ ].
- Adjacent codes should be at least *d* units in distance and your interface should support various values of *d*. Ex: [12][13] - d=1, [14][16] - d=2
- The function should output the information about success/fail cases.

Some failure cases include:

- Unable to open message file
- Unable to open key file
- Unable to satisfy the *d* condition!
- Unable to encode using the current key

5 Decoding

5.1 Task - Filtering

This is similar to the task performed in section 4.1

5.2 Task - Decoding

The encoded code is decoded using the keyfile. This is exactly the opposite of what happened in the encoder. Try decoding using a key that is not the original and see if you can get the original message - In most cases you shouldn't be able to get a good text! Notice that you don't really need *d* anymore for decoding.

6 Bonus task: Cracking

Warning: This task can be a bit challenging, so try to think before you start coding this.

Imagine you are a hacker listening to the stream of coded message and you know that they are coded using one of the lyrics from the song library. Is there a way you can decode the original message without knowing the actual key?

If this is too easy, use Project Gutenberg as the key library.

If it still is too easy, try cracking without any hints about the key at all! (I cannot promise that this is possible at all.)

HINT: Check the contents of /usr/share/dict/words!