DD1324 Applied Programming and Computer Science, Part 2

Laboratory Assignment 1 – Simple text encryption/decryption

The goal of this lab assignment is write a small program capable of simple text encryption and decryption. In order to complete this assignment, you will need to be familiar with basic C syntax, data types, functions, etc. You will also need to know how to use Makefiles and work with a standard project folder structure. Some basic usage of pointers will also be required (in order to pass arrays to functions), but most of the assignment can be completed without using pointers. Laboratory 2 will concentrate more on the use of pointers.

NAME

lab01 (simple lookup-based encryption/decryption)

SYNOPSIS

lab01 [OPTIONS] … [FILE]
lab01 [OPTIONS] … [STRING]

DESCRIPTION

The program takes a string or a text file as input and either encrypts (default) or decrypts the input depending on the selected option. The program outputs the encrypted/decrypted text to the screen. If the –t option is given, both the input and output are printed in a 2-column format.

-e	[Default] Encrypt the file or text given in the argument.
-d	Decrypt the file or text given in the argument.
-t	[Optional] Displays both the input file/text and output text in a 2-column format.
-h	Displays a help text and exits.

EXAMPLES

Calling the program to encrypt the text “hello there”

```
$./lab01 "hello there"

hRttj ahRNR
```

Calling the program to decrypt the same text

```
$./lab01 –d "hRttj ahRNR"

hello there
```

Calling the program to encrypt another string, this time using the –e flag

```
$./lab01 -e "This is another example."

Lh>@ >@ ~TjahRN R[~uvtRx
```

Calling the program to encrypt the text file `crypt.txt`

```
$./lab01 –e crypt.txt

mT |N'vajKN~vh'8 RT|N'va>jT >@ ahR vNj|R@@ jJ RT|jg>TK uR@@~KR@ jN
>TJjNu~a>jT >T @G|h ~ E~' ah~a jTt' ~GahjN>URg v~Na>R@ |~T NR~g >ax
VT|N'va>jT gjR@ Tja jJ >a@RtJ vNR4RTa >TaRN|Rva>jT8 YGa gRT>R@ ahR
uR@@~KR |jTaRTa aj ahR >TaRN|RvajNx mT ~T RT|N'va>jT @|hRuR8 ahR
>TaRTgRg |juuGT>|~a>jT >TJjNu~a>jT jN uR@@~KR8 NRJRNNRg aj ~@
vt~>TaR[a8 >@ RT|N'vaRg G@>TK ~T RT|N'va>jT ~tKjN>ahu8 KRTRN~a>TK
|>vhRNaR[a ah~a |~T jTt' YR NR~g >J gR|N'vaRgx
```

Encoding `crypt.txt` and sending the result to a file `secretcode00.txt` using the shell redirect >

```
$./lab01 –e crypt.txt > secretcode00.txt
```

Calling the program to decrypt the file `secretcode00.txt`

```
$./lab01 –d secretcode00.txt

In cryptography, encryption is the process of encoding messages or
information in such a way that only authorized parties can read it.
Encryption does not of itself prevent interception, but denies the
message content to the interceptor. In an encryption scheme, the
intended communication information or message, referred to as
plaintext, is encrypted using an encryption algorithm, generating
ciphertext that can only be read if decrypted.
```

Calling the program to decrypt the file secretcode00.txt with the twocolumn option

```
$./lab01 -d -t /secretcode00.txt

mT |N'vajKN~vh'8 RT|N'va>jT >@ ahR vNj|R    In cryptography, encryption is the proce
@@ jJ RT|jg>TK uR@@~KR@ jN >TJjNu~a>jT >    ss of encoding messages or information i
T @G|h ~ E~' ah~a jTt' ~GahjN>URg v~Na>R    n such a way that only authorized partie
@ |~T NR~g >ax VT|N'va>jT gjR@ Tja jJ >a    s can read it. Encryption does not of it
@RtJ vNR4RTa >TaRN|Rva>jT8 YGa gRT>R@ ah    self prevent interception, but denies th
R uR@@~KR |jTaRTa aj ahR >TaRN|RvajNx mT    e message content to the interceptor. In
 ~T RT|N'va>jT @|hRuR8 ahR >TaRTgRg |juu     an encryption scheme, the intended comm
GT>|~a>jT >TJjNu~a>jT jN uR@@~KR8 NRJRNN    unication information or message, referr
Rg aj ~@ vt~>TaR[a8 >@ RT|N'vaRg G@>TK ~    ed to as plaintext, is encrypted using a
T RT|N'va>jT ~tKjN>ahu8 KRTRN~a>TK |>vhR    n encryption algorithm, generating ciphe
NaR[a ah~a |~T jTt' YR NR~g >J gR|N'vaRg    rtext that can only be read if decrypted
x                                           .
```

SPECIFICATIONS

The program should be written to the following specifications.

1. Your project should use standard folder structures for software development.
   a. Source files in /trunk/src
   b. Executables in /trunk/bin
   c. Libraries (if necessary) in /trunk/build
   d. Makefile in /trunk

    e. Readme.md in root folder
    f. LICENSE file in root folder

2. Your program should parse inputs from the command line (the string, the name of the file, and the options `-e`, `-d`, `-t`). Do this in the main function using the standard function definition:
   `int main(int argc, char *argv[])`
3. The main file of your program should do the following:
    a. Parse and handle the options
    b. Generate a lookup table to perform the encryption/decryption (using a function or however you choose to implement this)
    c. Store the input string or the text from the input file into a string called `input`
    d. Call appropriate functions to encrypt or decrypt `input` and store the results in a string called `output`
    e. Call a function to display `output` to the screen. If the option -t is given, print `input` and `output` in a 2-column format
4. Functions for encryption, decryption, and display should be declared and defined in other files.
    a. Be sure to write header files and use `#include` to share declarations between files.
    b. Guard header files using `#ifndef` and #define
5. Use the debug header `dbg.h` to debug your code and to print errors for incorrect usage.
6. You should use good programming style. This includes informative comments.
7. The encryption is simply a re-ordering of the printable ASCII characters. Encryption and decryption can be performed by looking up the input character in a lookup table and replacing it with the result.

ASCII code	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	
Input	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	…
Output	w	6	M	f	V	:	-	o	m	Z	s	_	]	B	;	?	

Using this example, the input string "`A BIG COFFEE`" would be encrypted "`w 6m- M;::VV`".
You should only encode/decode the **printable ASCII characters**, that is, char with value 32 to 126 (https://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters). Other sequences such as '`\0`' and '`\n`' should not be translated. For legibility, translate ASCII 32 (`space`) to (`space`).

RESOURCES

On the course web page you will find the following resources

- A zip file containing a program skeleton with a few hints in the comments to get you started. Note: *the program skeleton does not contain all necessary files and does not use the required project directory organization*.
- Two files `code_ascii.txt` and `code_binary.dat` which both contain the lookup table for encryption. You can use either of them. The file `code_ascii.txt` is a text file organized into columns. The first column contains the input character, the second column contains the encrypted character, and the third column contains the ASCII code. Note that it goes from 32 to 126. The file `code_binary.dat` contains the encrypted characters in order. This file can be read directly into a string array in C.
- A zip file `texts` containing encrypted texts you can use to test your code.

HINTS

The following list of hints may prove helpful.

- You will need to be able to read a text file if you are given a file name. The C standard library stdio.h contains useful functions including `fopen`, `fread`, and `fclose`. You can read an entire text file to a string if you know the length of the string. Hint: look up `ftell`, `fseek`, `SEEK_END`, `SEEK_SET`.
- You can either code the lookup table by hand or you can read it into your program from the provided files.
- Remember to check if characters from the input string/text are **ASCII printable characters** or not, and handle them accordingly. There are exactly 94
- It might be a useful to store your lookup table as a global variable shared between files.
- You can set a maximum buffer size for the `input` string of 3000 characters.
- Check for the end of a string in a `for` or `while` loop by looking for `'\0'`.

EVALUATION

Arrange to demonstrate your code to one of the instructors when you are ready.

1. Before meeting, print out the Laboratory Assessment Protocol form (link) and bring it to the meeting.
2. You will be asked to demonstrate your code. This will consist of several steps:
   a. You should show that your program meets the specifications and produces the correct output.
   b. You will be asked to compile your code.
   c. The instructor will test if your code can decrypt a text file you have not seen.
   d. You should walk the instructor through each step of your program, explaining the overall architecture, and the purpose of each function.
3. You will be asked to demonstrate that your code does not have any memory leaks using `valgrind`
4. You should be prepared to show how to debug your code using `gdb`.