Lab 02: To perform Data Cryptography and De-cryptography

Setup:

1. Create a directory **`Lastname02`**. You are going to develop your lab here.

Caesar Cipher

One of the simplest examples of a substitution cipher is the Caesar cipher, which is said to have been used by Julius Caesar too communicate with his army. Caesar is considered one of the first persons to have every employed encryption for the sake of securing messages. Caesar decided that shifting each letter in the message would be his standard algorithm, and so informed all of his generals of his decision and was then able to send them secured messages. Use the Caesar Shift, with a rotation value of 3, the message "Return to Rome!" would be encrypted as "Uhwxuq wr Urph!". In this example, 'R' is shifted to 'U', 'e' is shifted to 'h' and so on. Now, even if the enemy did intercept the message, it would be useless, since only Caesar's generals could read it.

Only alphabetic characters encrypted (respecting upper and lowercase): a, b, c, A, B, C, etc. All other symbols are ignored. For example, numbers (1, 2, 3, etc.) and punctuation symbols (~, !, @, #, $, etc.) are all ignored.

Resources

- #include <ctype.h>
- #include <string.h>
- [Wikipedia] https://en.wikipedia.org/wiki/Caesar_cipher
- [GitHub] https://github.com/hurdleg/CaesarCipher

Program #1:

Write a small C program **`cc_encrypt.c`** that:

1. Prompt the User for a message to encrypt.
2. Read the message. The message cannot exceed 80 characters.
3. Prompt the User for a rotation key.
4. Reads the integer number for the key.
5. Validate the key's value: must be in the range 0 to 26 (inclusive).
6. If the key is invalid, print an error message and re-prompt the User for the key.
7. If the key is valid, your program should encrypt the message using the rotation key, following the Caesar cipher encryption algorithm given above.
8. At the end, your program should print the encrypted message and terminate with a value of: **`EXIT_SUCCESS`**

The following demonstrates the run-time behaviour of the program:

```
kraken:hurdleg02 hurdleg$ ./make.sh
gcc -g -ansi -pedantic -Wall -o cc_decrypt cc_decrypt.c
gcc -g -ansi -pedantic -Wall -o cc_encrypt cc_encrypt.c
kraken:hurdleg02 hurdleg$ ./cc_encrypt
Enter message to encrypt:
Return to Rome!
Enter rotation key:
3

Encrypting message (key = 3): Return to Rome!

Uhwxuq wr Urph!
kraken:hurdleg02 hurdleg$ ./cc_encrypt
Enter message to encrypt:
Remember to validate key's value.
Enter rotation key:
abc
Error - key must be in range 0 and 26. Try again.
Enter rotation key:
-1
Error - key must be in range 0 and 26. Try again.
Enter rotation key:
27
Error - key must be in range 0 and 26. Try again.
Enter rotation key:
0

Encrypting message (key = 0): Remember to validate key's value.

Remember to validate key's value.
kraken:hurdleg02 hurdleg$ echo $?
0
kraken:hurdleg02 hurdleg$
```

SAMPLE TEST OUTPUT: cc_encrypt

To successfully complete this program and obtain all the marks, you will need to:

1. Use the macros **`EXIT_FAILURE`** and **`EXIT_SUCCESS`** defined in the library **`stdlib.h`** to indicate unsuccessful or successful termination of your program
2. Define **`MAX_MESSAGE`** as a macro in your program. **`MAX_MESSAGE`** is be defined as: 80
3. Compile your program with the flags: **`-Wall -ansi -pedantic`**

Program #2:

Copy your **cc_encrypt.c** to a new program **cc_decrypt.c.** Your decrypt program should be able to decrypt an encrypted message using a rotation key.

```
kraken:hurdleg02 hurdleg$ diff cc_encrypt.c cc_decrypt.c
17c17
<     puts("Enter message to encrypt:");
---
>     puts("Enter message to decrypt:");
44c44
<   printf("\nEncrypting message (key = %d): %s\n\n", key, input_buffer);
---
>   printf("\nDecrypting message (key = %d): %s\n\n", key, input_buffer);
45a46
>   key = MAX_ROTATION - key;

kraken:hurdleg02 hurdleg$ ./cc_decrypt
Enter message to decrypt:
Uhwxuq wr Urph!
Enter rotation key:
-1
Error - key must be in range 0 and 26. Try again.
Enter rotation key:
27
Error - key must be in range 0 and 26. Try again.
Enter rotation key:
3

Decrypting message (key = 3): Uhwxuq wr Urph!

Return to Rome!
kraken:hurdleg02 hurdleg$ echo $?
0
kraken:hurdleg02 hurdleg$
```

SAMPLE TEST OUTPUT: decrypt

Marking

This lab is out of 20 points:

- ☐ 10 points for cc_encrypt.c
  - 05 for coding correctness (i.e., correct results) and compliance to “Assignment Submission Standard”
  - 05 for demonstration during scheduled lab
- ☐ 10 points for cc_decrypt.c
  - 05 for coding correctness (i.e., correct results) and compliance to “Assignment Submission Standard”
  - 05 for demonstration during scheduled lab

Submission and Demonstration

- ☐ Make a zip-file (named: yourLastname02.zip) that contains:
  - ‘C’ source code files (*.c)
  - make.sh (if you made one)
  - acceptance-test.sh (if you made one)
  - screenshots of:
    - gcc -g -ansi -pedantic -Wall cc_encrypt.c
    - gcc -g -ansi -pedantic -Wall cc_decrypt.c
    - output of cc_encrypt; your output is to match the “sample test output” exactly, as seen on page 2.
    - output of cc_decrypt; your output is to match the “sample test output” exactly, as seen on page 3.
- ☐ Demonstration during your scheduled lab period.