CSE 220: Systems Fundamentals I

Stony Brook University
Homework Assignment #3
FALL 2023

Due: Wednesday Oct 11 at 11: 59 PM EST

Learning Outcomes

After completion of this homework assignment you should be able to:

- Perform non-trivial string processing in C.
- Some basic understanding of C pointers

IMPORTANT:

- You will be provided strPtr.c for Part 1, and caesar.c for part 2. Only these files will be used by the TAs to mark your code and implementation. Do Not include the main() function [the driver function] in these files. Use a separate file for this. Then include/call the defined functions through pre-processors.
- Include a text document if you feel the need to explain the scope of your work.
- Use comments to explain your implementation to a person who is looking at your code for the first time. There will be credits for comments.
- No time extension will be given.
- Do not submit your code on Brightspace. It will not be graded.

Part 1:

Use of C string lib support is not allowed! You have to make it your own. Use the standard array approach to implement the functions.

Implement the following string functions in file strgPtr.c.

- **int strgLen(char* s)**: return the length of string s.
- **void strgCopy(char* s, char* d)**: copy the content of string s (source) to d (destination).

- **void strgChangeCase(char* s)**: for each character in the string, if it is an alphabet, reverse the case of the character (upper to lower, and lower to upper). Keep the non-alphabet characters as is.
- **int strgDiff(char* s1, char* s2)**: compare strings s1 and s2. Return the index where the first difference occurs. Return -1 if the two strings are equal.
- **void strgInterleave(char* s1, char* s2, char* d):** copy s1 and s2 to d, interleaving the characters of s1 and s2. If one string is longer than the other, after interleaving, copy the rest of the longer string to d. For example, given s1 = “abc" and s2 = “123”, then d = “a1b2c3”. If s1 = “abcdef” and s2 = “123”, then d = “a1b2c3def”

Test Cases

- **int strgLen(s):** Return the length of the strings

Sample Input	**Return Value**
“Stony Brook”	11
“CSE 220”	6
“C”	1
“System Fundamental”	18
“1”	1
“”/empty string	0
NULL	-1

- **void strgCopy(s, d)**: copy the content of string s (source) to d (destination).

Content of s	**Content of d (after calling the function)**
“Computer Science”	“Computer Science”
“CSE-220”	“CSE-220”
“System Fundamental”	“System Fundamental”
“1”	“1”
“”/empty	“”/empty

Note:
1- In case any of s or d is NULL, do nothing and return from the function
2- You should copy everything including the null terminator
3- If d does not have enough space to hold s, copy until you reach the last index of d and make it null-terminated. For example if d is enough for 5 bytes and you have “Computer Science” in s, after calling the function, d should have “Comp”

- **void strgChangeCase(s)**: for each character in the string, if it is an alphabet, reverse the case of the character (upper to lower, and lower to upper). Keep the non-alphabet characters as is.

Content of s (before calling the function)	Content of s (after calling the function)
“Stony Brook”	“sTONY bROOK”
“CSE220”	“cse220”
“C”	“c”
“System Fundamental”	“sYSTEM fUNDAMENTAL”
“1”	“1”
“”/empty	“”/empty
NULL	Do nothing and return from the function

- **int strgDiff(s1, s2):** compare strings s1 and s2. Return the index where the first difference occurs. Return -1 if the two strings are equal.

Content of s1	Content of s2	Return Value
“Hello”	“Hello”	-1
“CSE-220”	“CSE220”	3
“CSE220”	“SE220”	0
“”	“”	0

Note: In case any of s1 or s2 is NULL, return -2

- **void strgInterleave(s1, s2, d):** copy s1 and s2 to d, interleaving the characters of s1 and s2. If one string is longer than the other, after interleaving, copy the rest of the longer string to d. For example, given s1 = “abc” and s2 = “123”, then d = “a1b2c3”. If s1 = “abcdef” and s2 = “123”, then d = “a1b2c3def”

Content of s1	Content of s2	Content of d (after calling the function)
“abc”	“123”	“a1b2c3”
“abcdef”	“123”	“a1b2c3def”
“cse”	“12345”	“c1s2e345”
“1234”	“cs”	“1c2s34”
“”/empty	“”/empty	“”/empty
“”	“123”	“123”

Note: In case any of s1, s2, or d is NULL, do nothing and return from the function. If the d does not have enough space, interleave until you reach the last byte of d and null-terminate it. For example, give s1 = “abc” and s2 = “123”, and give d has only 5 bytes, after calling the function, d should be = “a1b2”

Part 2:

Overview

The Caesar Cipher is one of the oldest methods to convert data into a format unauthorized users cannot recognize. Encryption is a process of converting data into secret code, and decryption is the exact opposite process, i.e., converting code to original data. It shifts each letter a few positions right to encrypt it. Further, this can be shifted to the left to get the message i.e to decrypt it (Caesar Cipher).

Caesar gave a very simple method to encrypt and decrypt the information. It is also known as shift Caesar as the method shifts the character key positions ahead. If the current character is **a** and the **key = 4**, then Cipher text will store **e**, i.e., 4 positions ahead of **a**. To decrypt the same, we go **4** positions behind for **e**, which gives us **a** back. The same is true for the digits. If the current digit character is **1** and the **key = 4**, then Cipher text will store **5.** To decrypt the same, we go 4 positions back from **5** to get **1**.

For this assignment, the upper case letter will be replaced with upper case letter when shifted. For example, “A” will be replaced with “C” when shifted right four steps using Key= 2. Similarly, the lowercase letter will be replaced with lower case letter and did will be replaced with a digit.

A question may arise about the extreme characters such as **y or Y** and the **key=3**. We are not left with alphabets, so it starts counting again with **b or B**. Similarly, if the encrypted message is on decrypting, we get **y or Y** with the same key value. The same logic is applied to the digits. For digit 9 and the key=3, we will have 2. We will get 9 with key=3 by moving back from 2 during the decryption process.

Encryption and Decryption

In this assignment, you will write a C program for the Caesar Cipher encryption. Files caesar.c and caesar.h will be provided to you that will contain the interfaces that you need to implement.

Part A: Encryption

```
int encrypt( const char *plaintext, char *ciphertext, int key)
```

Complete the function encrypt, which encrypts a message with caesar cypher strategy using a given key.

Arguments:

- plaintext: a null-terminated string to be encrypted. The string contains only ASCII characters
- ciphertext: a null-terminated string used to encrypt the plaintext according to the caesar

cipher approach. A mutable character array is passed as ciphertext so that its contents may be changed.

Return Value:

- The number of characters from plaintext successfully encoded, or
- -1 if there was insufficient memory to encode the EOM marker (__EOM__).
- -2 if any of the plaintext or ciphertext is NULL.

ciphertext can be an empty string or it might contain text. Characters in ciphertext may be modified to encode characters or the EOM marker. (The EOM marker is a 7 character string "__EOM__")

Example:

plaintext: "System Fundamentals"

ciphertext before function call: "I can store any message!";

ciphertext after function call with Key=1: "Tztufn Gvoebnfoubmt__EOM__"

Return value : 18

Part B: Decryption

```
int decrypt(const char *ciphertext, char *plaintext, int key)
```

Arguments:

- ciphertext: a null-terminated string that encodes a message using cipher.
- plaintext: a null-terminated string of (possibly) random characters. A mutable character array is passed as plaintext so that its contents may be changed.

The function must null-terminate the plaintext message. If the ciphertext contains more than one EOM marker, decrypt the message up to the first EOM marker. If the plaintext string cannot store all of the decrypted ciphertext, store the first strlen(plaintext) characters of the encrypted ciphertext in plaintext.

Returns:

- the number of encrypted characters successfully decrypted, or one of the following error codes:
    - 0 if the length of the plaintext string is 0 (e.g., as computed by strlen). This means we cannot decrypt any portion of the ciphertext.
    - -1 if the ciphertext is missing the EOM marker. This means the ciphertext is invalid.
    - -2 if any of the ciphertext or plaintext is NULL.
- When any error is present in the arguments (meaning that the function is going to return an error code), the function must not change the contents of the plaintext.
- Check for the error cases in the order provided. i.e., if both errors are present, return the maximum (i.e., least negative) valid error code.

Characters in plaintext may be modified only to decrypt the ciphertext.

Test Cases:

Part A: Encryption

```
int encrypt( const char *plaintext, char *ciphertext, int key)
```

Ciphertext Test Cases

Plaintext	Key	Ciphertext After Function Call	Return Value
abc	2	cde__EOM__	3
Ayb	3	Dbe__EOM__	3
Cse220	1	Dtf331__EOM__	6
CS	0	CS__EOM__	2
empty	Any key	undefined__EOM__	0

Part B: Decryption

```
int decrypt(const char *ciphertext, char *plaintext, int key)
```

Ciphertext Test Cases

Ciphertext	Key	Plaintext After Function Call	Return Value
cde__EOM__	2	abc	3
Dbe__EOM__	3	Ayb	3
Dtf331__EOM__	1	Cse220	6
CS__EOM__	0	CS	2
empty__EOM__	Any key	undefined	0

Running, Testing and Submitting Your Code

Running test cases using Criterion is optional for this homework. You can have your way of testing the code. However, as we move to the next assignment we will adopt this framework for testing the homework. The testing framework is provided in the starter code. Try it and familiarize yourself.

To run the provided unit tests (written using Criterion):

- `make` to build your code
  - The directory structure for this homework assignment is different from HW #2. Binaries are now located in the `bin` directory. If you want to understand more, feel free to explore the provided `makefile`.
- `make test` to test your code
- `make gcov` to check the test coverage (see below)

The provided test cases are nowhere near comprehensive. It is your responsibility to write additional test cases to verify the correctness of your code. If you create new test cases using Criterion (instead of `printf`), then both `make tests` and GitHub will run them automatically (GitHub only when you `git push` your work).

Remember to regularly `git commit` your work. Occasionally, `git push` your work to run the same tests on the git server ***and to submit your work for grading*** by the due date.

Grading Notes

During grading, only your `ceasar.c` file will be copied into the grading framework's directory for processing. Make sure all of your code is self-contained in that file.

> ***Extensions, resubmissions and regrades will not be granted because a student did not use git properly to submit the assignment.***
>
> Academic Honesty Policy
>
> Academic honesty is taken very seriously in this course. By submitting your work for grading you indicate your understanding of, and agreement with, the following Academic Honesty Statement:
>
> 1. I understand that representing another person's work as my own is academically dishonest.
> 2. I understand that copying, even with modifications, a solution from another source (such as the web or another person) as a part of my answer constitutes plagiarism.
> 3. I understand that sharing parts of my homework solutions (text write-up, schematics, code, electronic or hard-copy) is academic dishonesty and helps others plagiarize my work.
> 4. I understand that protecting my work from possible plagiarism is my responsibility. I understand the importance of saving my work such that it is visible only to me.
> 5. I understand that passing information that is relevant to a homework/exam to others in the course (either lecture or even in the future!) for their private use constitutes academic dishonesty. I will only discuss material that I am willing to openly post on the discussion board.
> 6. I understand that academic dishonesty is treated very seriously in this course. I understand that the instructor will report any incident of academic dishonesty to the University's Academic Judiciary.
> 7. I understand that the penalty for academic dishonesty might not be immediately

administered. For instance, cheating on a homework assignment may be discovered and penalized after the grades for that homework have been recorded.
8. I understand that buying or paying another entity for any code, partial or in its entirety, and submitting it as my own work is considered academic dishonesty.
9. I understand that there are no extenuating circumstances for academic dishonesty. 8