Lab 2: C Strings

Due Friday by 8:59pm **Points** 0 **Available** after Jan 29 at 10am

Goals: The goal of this lab is to understand and work with characters and strings in C.

1 Fun with Chars

A character set is a table that maps each character to a unique numeric code. C uses the ASCII character set to represent characters. You can convert between a character and its corresponding integer code easily, like this:

```
char ch;
int code;

//character to integer
ch = 'a';
code = (int)ch;

//integer to corresponding character:
code = 99; //must be less than 256!
ch = (char) code;
```

Furthermore, all the letters of the English alphabet are next to each other in a contiguous chunk in the character set, and in alphabetical order. This means that the integer code for 'b' is exactly one more than that of 'a', and so on. The same happens for uppercase letters: the integer code for 'M' is exactly one more than that of 'L', and exactly one less than that of 'N'.

This can be used to modify character using Math.

1.1 Get the next and previous character

Write a function `char next(char ch)` that takes a single character. If it is a valid letter (uppercase or lowercase) it should return the character after it. For example, passing 'a' should return 'b', passing 'C' should return 'D'. Passing 'z' should return 'a' (i.e. it wraps around). If the character is not a letter, it should simply return the same character.

Think about the following questions as you design the logic of this function:

1. How "far" from 'a' is any lowercase character? How "far" from 'A' is any uppercase character? Can you use this to determine if the character is a valid letter?
2. Can you use the above relative position from 'a' or 'A' to compute what the next character in order should be?

1.2 Change case

Write a function `change_case(char ch)` that takes a single character. If this character is a letter in the English alphabet, it should return its counterpart in the other case. For example, 'a' should return 'A', 'Z' should return 'z'. If the character is not a letter, this function should simply return the same character.

How can the above questions help here?

Fun With Strings

A string is quite simply a sequence of characters. Thus we can represent a string as ... an array of characters!

```
//not quite a string, but almost!
char arr[] = {'c','s','5','0','0','8'};
```

The above is almost a string. The missing piece is a special character that signifies the "end of a string". Without it, C would not be able to tell whether there is a string that occupies only a part of an array. This character is the "null" character: '\0'.

```
char word[] = {'c','s','5','0','0','8','\0'};

char another_word[4]; //can store strings of length at most 3

another_word[0] = 'b';
another_word[1] = 'o';
another_word[2] = 'b';
//not a valid string yet...
another_word[3] = '\0'; //now its a string: "bob"

another_word[2] = '\0'; //now its a string: "bo"
```

As the above example shows, there needs to be space to store the null character. This means that an array of characters of length `n` can be used to store strings of length up to `n-1`.

C has several inbuilt functions to process strings. These (and many others) can be found in `string.h`.

1. `strlen(char *word)` : returns the length of the word (the length does not count the '\0').
2. `strcmp(char *word1,char *word2)` : return a negative number, 0 or positive number if the first word is before, the same or after the second word in lexicographical (dictionary) order.
3. `strcat(char *one,char *two)` : concatenates (adds to the end of) the second string to the first string. The first argument should be an array long enough to hold the result (plus the null character!)

As you may have guessed, most of these functions need to know when the string ends (and thus, the '\0' needs to be present in the arrays passed to them!)

2.1 Jumble it up!

Write a function `void shift(char *str,int x)` that takes a valid string, and changes it so that each letter is "shifted x places to the right". For example if `x=1` , then 'a' turns to 'b', 'b' turns into 'c', 'Z' turns into 'A', and so on. If `x=3` , then 'a' turns into 'd', 't' turns into 'w', 'x' turns into 'a', and so on. Characters that are not letters will remain unchanged.

A negative value of `x` has the effect of shifting left instead of right.

Note that it is possible that only part of the array passed to this function is a string (i.e. '\0' may be somewhere in the middle of the array). In this case, you are expected to only modify the "string" part of the array.

This shifting function is a simple way to encrypt text: this is called a shift cipher.

1. Choose a value of 'x'.
2. Encrypt a string by calling `shift` with 'x'.
3. Decrypt an encrypted string by calling `shift` with '-x'! To decrypt, one must know what value of 'x' was used to encrypt it.

What to Submit

You can choose to distribute these functions among files in any suitable way. Write a `main` function that tries these function. It is highly recommended that you write unit tests for them, as the assignment specifies.

Before submitting, please ensure that your code works correctly on the Khoury Linux machines!

Submit all your source code for both problems to gradescope by the deadline. You do not need to submit executable programs.