Lab5: C Strings

General Instructions:

1- Create a C project in eclipse called Lab5
2- Download Lab5.zip files and extract it to a folder in your machine. Drag and drop the files to the Lab5 project.
3- Rename: "Lab5_template.txt" to "Lab5.c". Add your credentials on top of the file.
4- The file "Lab5.c" is the file you are going to edit and the file that you need to submit at the end.
5- Test your results using "Lab5_test.c" file. Your results should 100% match those presented in Lab5_output.txt.

Instructions:

Implement the two functions:

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

Both receive a pointer to a string that has been dynamically created using a malloc operation. The second parameter is an integer representing key, which is the number of shifts.

The purpose of the above two functions is to perform encryption and decryption using Caesar cipher (see here for more information).

Both functions should ignore spaces and special characters, i.e. these characters are not encrypted or decryption and are left unchanged. However, the function should maintain the case of the letter, i.e. an upper case in the plaintext should show as an upper case in the ciphertext and vice versa.

The key represent the number of shifts to be applied to the alphabet. A key value of 0 would mean no shifts are applied, and thus the plaintext and ciphertext will be identical. Note that a key value of 26 is also equivalent to a key value of 0, because the shifts will return to the original position since the English language has 26 characters.

In encryption, you need to create a ciphertext pointer using malloc/calloc and return it. In decryption, you need to create a plaintext pointer using malloc/calloc and return it.

In your solution, you can use any of the following libraries: <stdio.h>, <stdlib.h>, <string.h>, <math.h> and <ctype.h>.

Some hints:

1- Solve the encryption function first. The decryption is the reverse process.
2- Create a loop that encrypts/decrypts each character separately. The loop should stop when the end of string is reached.
3- Use character arithmetic, i.e. ASCII values, similar to solution of has_double in R5.
4- Do not worry about upper case characters in your initial attempt. Deal with all characters as lower case. If you results are valid, modify the function to handle the upper/lower case issue.