Workshop 2 Solution

Strings: Caesar Cypher

Julius Caesar protected his confidential information by encrypting it using a cipher. Caesar's cipher shifts each letter by a number of letters. If the shift takes you past the end of the alphabet, just rotate back to the front of the alphabet. In the case of a rotation by 3, w, x, y and z would map to z, a, b and c. Complete the Caesar cipher function to encrypt/decrypt a string

Original alphabet: abcdefghijklmnopqrstuvwxyz
Alphabet rotated +3: defghijklmnopqrstuvwxyzabc

Note: The cipher only encrypts letters; symbols, such as -, remain unencrypted. Capital letters stay capital and vice versa.

Solution:
First, mod the rotation. Then for each character, if it is an alphabet, decide if it is lowercase or uppercase. Then, add the rotation to the character. If it goes out of bounds, use ‘a’ and ‘z’ or ‘A’ and ‘Z’ to correct the overflow. Add this to the return string. If it was not an alphabet character, then simply add it to the return string.

```
def caesarCipher(word, rotation):
 lower_start = 'a'
 lower_end = 'z'
 upper_start = 'A'
 upper_end = 'Z'

 if rotation == 0 or rotation%26 == 0:
  return word

 modded_rotation = rotation%26

 ret = ""
 for c in word:
  if not c.isalpha():
   ret += c
   continue
  if c.isupper():
   new_c = ord(c) + modded_rotation
   if new_c > ord(upper_end):
    overflow = new_c - ord(upper_end)
    new_c = ord(upper_start) + overflow - 1
   ret += chr(new_c)
  else:
   new_c = ord(c) + modded_rotation
   if new_c > ord(lower_end):
    overflow = new_c - ord(lower_end)
    new_c = ord(lower_start) + overflow - 1
   ret += chr(new_c)
 return ret

print(caesarCipher("z", 1))
print(caesarCipher("There's-a-starman-waiting-in-the-sky", 3))
```

Runtime O(n)
Note: depending on the Python implementation, string concatenation might require copying and the actual runtime would then be O(n^2)
Space O(n) (could be O(1) if you reuse the passed in word)

C Solution:

```
#include <cctype>

struct RANGE {
  static const char upperBase = 'A';
  static const char lowerBase = 'a';
  static const char range = 'Z' - 'A';
};

inline char shift(char c, size_t k) {
  auto distance = k % RANGE::range;
  auto base = std::isupper(c) ? RANGE::upperBase : RANGE::lowerBase;
  return ((c + distance) % base % RANGE::range) + base;
}

char* caesarCipher(size_t n, char* my_string, size_t k) {

  for (size_t i = 0; i < n; i++) {
    if (std::isalpha(my_string[i])) {
      my_string[i] = shift(my_string[i], k);
    }
  }

  return my_string;

}
```