COMP 211: Principles of Imperative Computation, Fall 2014
Programming 2: Crypto

Due: Wednesday, 17 Sep, 6pm

1 Mechanics

You should first download `crypto.c0` from the assignments web page and save it in your `comp211` directory. You should write your code for this assignment in this file.

To hand in this homework, upload your `crypto.c0` file to your handin directory on WesFiles, which is located at

```
/courses/COMP-211-01-dlicata/handin/YOURNAME/
```

Ask the course staff if you have trouble doing this.

2 Overview

Suppose you are sending a message, and that people other than the intended recipient can read the text of what you send, but you don't want them to understand it. For example, when you send your credit card info to a web site, you want the business at the other end to be able to understand your number, but you don't want anyone else who spies on it along the way to be able to understand it.

One way to do this is something called *encryption.* The idea is that you take a message (often called the *plaintext*)

```
"this is a secret message"
```

and you change it to a different collection of letters (often called the *ciphertext*)

```
"bpqa qa i amkzmb umaaiom"
```

that is hard to read, and send that instead. This is called *encryption.*

The intended receipient is supposed to know how to turn the ciphertext back into the plaintext, getting the original message back out. This is called *decryption.* A good encryption method will have the property that it is very difficult for anyone besides the intended recipient to decrypt the message. *Breaking* an encryption method means coming up with a way to decrypt messages without being the intended recipient.

In this homework, you will implement an encryption method called a *shift cipher* or *Caesar cipher* (because it is said to have been used by Julius Caesar). You will also show that this encryption method is not very good, by developing a method to break it. However, there are better methods that use similar ideas, e.g. a *one-time pad* or *Vigenère cipher*, which you can choose to implement for extra credit. Even though they are easy to break, shift ciphers are still used to protect information against accidental reading—e.g. ROT13, a particular shift cipher, was used for a long time in email and newsgroups to hide spoilers and punchlines of jokes.

Here is how a Caesar cipher works: We will consider messages that consist entirely of lowercase letters `a` through `z`. A *shift* is a number between 0 and 25. Given a shift s, to encrypt a message, each letter in the message is *shifted* s letters later in the alphabet.

For example, shifting by one turns `a` into `b`, `b` into `c`, etc. and then wraps around and turns `z` into `a`.

```
Plaintext:  abcdz
Ciphertext: bcdea
```

Here is a longer example of shifting by one:

```
Plaintext:  thisisasecretmessage
Ciphertext: uijtjtbtfdsfunfttbhf
```

Shifting by two turns `a` into `c`, `b` into `d`, . . . `y` into `a`, and `z` into `b`:

```
Plaintext:  abcyz
Ciphertext: cdeab
```

Shifting by 9 works like this:

```
Plaintext:  thisisasecretmessage
Ciphertext: cqrbrbjbnlancvnbbjpn
```

To communicate with someone "securely", you first agree on a particular shift s (via "out of band" communication, not transmited using this encryption method). Then, you encrypt your message by shifting each letter s positions *later* in the alphabet. Once your intended recipient has received the message, they can decrypt it by shifting each letter s positions *earlier* in the alphabet.

3 Encryption

3.1 Encrypting a character

First, we consider encrypting a single character.

Task 1 (3pts) *Write a function*

```
char encrypt_char(char input, int shift)
```

The input and output characters must be between `'a'` *and* `'z'`*. The number* `shift` *must be a number between 0 and 25. The result character should be the input character shifted* `shift` *letters later in the alphabet.*

Hint: you will want to use `char_ord` and `char_chr` to convert characters to and from numbers; see the description of these functions below.

3.2 Encrypting a character array

Next, using `encrypt_char`, write a function to encrypt a whole character array:

Task 2 (3pts) *Write a function*

```
void encrypt(char[] message, int message_length, int shift)
```

that encrypts each character in the array `message` *by shifting it by* `shift`*. You can assume that* `message_length` *is the number of characters in the array* `message`*. The function should* change *each character in the array message to the encryption of that character.*

3.3 Encrypting a string

Overall, we would like an encryption function that works on strings, not character arrays, because strings are much easier to type in.

Task 3 (3pts) *Write a function*

```
string encrypt_string(string message, int shift)
```

that encrypts the string `message` *by* `shift`*, producing a new string.* For example,

```
--> encrypt_string("thisisasecretmessage",13);
"guvfvfnfrpergzrffntr" (string)
```

Hint: use `encrypt` from the previous task and the functions `string_to_array` and `array_to_string` (see the descriptions of them below).

Task 4 (1pt) *Encrypt a "secret" message and post it on Piazza in the thread for this homework. (Don't make it anything too secret; see the tasks below.) Do NOT post the shift that you used to create the message.*

4 Decryption

Task 5 (3pts) *Write a function* `decrypt_string` *that is given a string (representing some ciphertext), and a shift, such that the string was encrypted with that shift. The function should return the plaintext that results from decrypting the message.*

```
string decrypt_string(string message, int shift)
```

For example,

```
--> decrypt_string("pq bpmzm",8);
"hi there"
```

Hint: there is a *very short* way to write this function using what you have already done.

5 Breaking the encryption

For the messages your classmates posted on Piazza, you are not the "intended recipient" and you do not know what shift they were encrypted with. However, it is not very difficult to *break* a Caesar cipher, because there are only a small number of possible ciphertexts for a given message. (Despite this, the Caesar cipher is conjectured to have been effective in Caesar's time, because the intercepting armies were largely illiterate, and because the concept of encryption was not widely known, so messages would have thought to have been written in an unknown foreign language.)

Task 6 (4pts) *Write a function*

```
string[] possible_decryptions(string message)
```

that returns an array of all possible decryptions of `message`*, for all possible shifts.*

You can use the function `print_strings` described below to print out the decryptions and read them.

Task 7 (1pts) *Use* `possible_decryptions` *and* `print_strings` *to decrypt five of your classmates' "secret" messages from Piazza. Include the decrypted messages in a comment in* `crypto.c0`*.*

Task 8 (2pts) *Briefly describe the algorithm that you used to decide which of the possible decryptions is the plaintext. Why does this algorithm work? Hand the description in in a comment in* `crypto.c0`*.*

6 Bonus Task: Vigenère Cipher

This task is worth a little extra credit, which will be considered at the end of the semester if you are on the boundary between letter grades.

There is an improvement on a shift cipher called a *Vigenère cipher*. The idea is to increase the number of possible encryptions of a message by shifting different letters by different amounts.

Instead of a single shift number, encryption and decryption use a *codeword*, which is a string consisting of all lowercase letters. The letters of the codeword are thought of as shifts, so `'a'` means shift by 0, `'b'` means shift by 1, `'c'` means shift by 2, etc.

A message is encrypted by shifting each letter in it by the amount indicated by the corresponding letter of the codeword. If the codeword is shorter than the message, the codeword is repeated once you get to the end of it.

For example, suppose you have a message:

```
thisisasecretmessage
```

and have picked the codeword `"badwolf"`. Because the codeword is shorter than the message, we repeat it, which we can visualize as follows:

```
message:  thisisasecretmessage
shift by: badwolfbadwolfbadwol
```

Now, each character is shifted, like in a Caesar cipher, by the amount indicated by the letter of the codeword:

```
message:   t h i s  ...

shift by:  b a d w  ...
means:     1 0 3 22 ...

result is: u h l o
```

Overall, the ciphertext is

```
"uhlowdftefnserfsvwup"
```

To decrypt, you shift each letter in the message the other way, undoing the shifts that encrypt does.

Task 9 (Extra Credit) *Write functions*

```
string encrypt_string_v(string message, string codeword)
string decrypt_string_v(string message, string codeword)
```

that implement Vigenère encryption and decryption. All of the inputs and outputs should consist entirely of lowercase letters.

Task 10 (Extra Credit) *Pick a partner who is also doing the bonus task. If you like, you can post on Piazza in the "partners" thread to find someone.*

With your partner, agree on a codeword (not on Piazza—by email, or in class). Now, one person should post an encrypted message on Piazza in the Viginère thread (don't post anything too secret, since Vigenère ciphers can still be attacked—look up Vigenère cipher on wikipedia), which asks a question of your partner. The other person should answer the question by posting another message, also in the same thread on Piazza.

Hand in the name of your partner and the codeword you used.

7 String Functions

Defined in the `string` library:

```
// concatenates two strings
string string_join(string a, string b)

// gets the character of s at position i
// where positions are number 0,1,...
char string_charat(string s, int i)
```

```
// turn a number (which must be 0...127)
// into the corresponding character, based on ASCII encoding
char char_chr(int n)

// turn a character into the corresponding number
// based on ASCII encoding
// result will be 0...127
int char_ord(char c)

// gets the number of characters in a string
int string_length(string s)

// tests whether two strings are equal
bool string_equal(string a, string b)
```

Defined in the homework handout code:

```
// given an array of strings (whose length is 'length'),
// print them all out so you can read them
void print_strings(string[] strings, int length)

// given an array C of characters whose length is 'length',
// return a string with the same characters in the same order
string array_to_string(char[] C, int length)

// given a string, turn it into an array of characters,
// with the same letters in the same order.
// the length of the result will be string_length(s)
char[] string_to_array(string s) {
```

For example, `string_to_array("hello")` produces the array (of length 5)

$'h'$	$'e'$	$'l'$	$'l'$	$'o'$

and `array_to_string(<this array>,5)` produces the string `"hello"`.