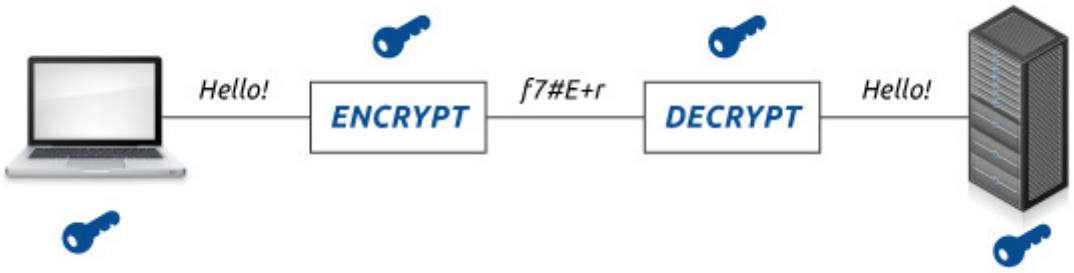
Lab 7. Strings and Ciphers

Cryptography is the practice and study of techniques for secure communication in the presence of third parties called adversaries.



Until modern times, cryptography referred almost exclusively to **encryption** and **decruption**. Encryption is the process of converting ordinary information (called **plaintext**) into unintelligible text (called **ciphertext**). Decryption is the reverse process.

A **cipher** is a pair of algorithms for performing encryption and decryption. In this lab, you will implement Caesar and Vigenere ciphers.

Before we introduce these ciphers, lets us give you more information on types `char` and `string`.

`char`

For historical reasons, type `char` is a numeric type, just like `int`. Each character in C++ is represented by its **integer code**, for example:

`'A'` == `65`

`'B'` == `66`

`'C'` == `67`

`'D'` == `68`

...

`'Z'` == `90`

...

`'a'` == `97`

This encoding is called **ASCII** (see full table here or type `man ascii` in the terminal).
Since each `char` is really an integer, if we add `'A' + 25`, we will get code `90`, which is character `'Z'`:

```
'A' + 25 == 'Z'  // is true
```

When working with type `char`, many useful functions are defined in header `<cctype>`, some of them are listed below:

`isalpha(c)`	Checks if character `c` is alphabetic,
`isdigit(c)`	Checks if character `c` is decimal digit,
`isspace(c)`	Checks if character `c` is a white-space.

`string`

`string` is a C++ type for representing sequences of characters. For this lab, we will need to know how to use the following `string` operations: Given strings `s` and `t`, and character `c`,

`s.size()` and `s.length()`	Return the number of characters in string `s`
`s + t`	Returns the concatenation of strings `s` and `t` (can also concatenate `string` with `char`, `s + c`)
`s += t`	Appends `t` to string `s` (can also append `char`, `s += c`)
`s == t`	Checks string equality (returns `true` or `false`)
`s[i]`	Returns the character at the index `i`

Note that the concatenation operator `+` does not change neither `s` nor `t`, while the append operator `+=` updates variable `s` with the result of `s+t`. So, intuitively, the definition of these operators for strings is very similar to their definition for numeric types.

A small example program:

```
#include <iostream>
using namespace std;

int main() {
    string h = "Hello";
    string w = "World";

    string message = h + ", " + w;
    cout << message << endl;     // Will print:  Hello, World

    message += "!!!";
    cout << message << endl;     // Will print:  Hello, World!!!

    cout << h.length() << endl; // Will print:  5
    cout << w[0] << endl;        // Will print:  W
}
```

To read user input that may contain space characters, one can use function `getline`, which extracts all characters **until an end-of-line** is reached. A minimal usage example:

```
int main() {
    string s;
    getline(cin, s);
    cout << "Your line was: " << s << endl;
}
```

Task A. Testing ASCII

Write a program `test-ascii.cpp` that asks the user to input a line of text (which may possibly include spaces). The program should report all characters from the input line together with their ASCII codes.

Make sure to print *exactly one* space between the character and it's code.

Example:

```
$ ./test-ascii

Input: Cat :3 Dog

C 67
a 97
t 116
  32
: 58
3 51
  32
D 68
o 111
g 103
```

Hint:

When you are printing a value of type `char` on the screen, it is normally shown as a symbol. To print it as a number (as its ASCII code), type cast it to integer:

```
cout << (int)c;
```

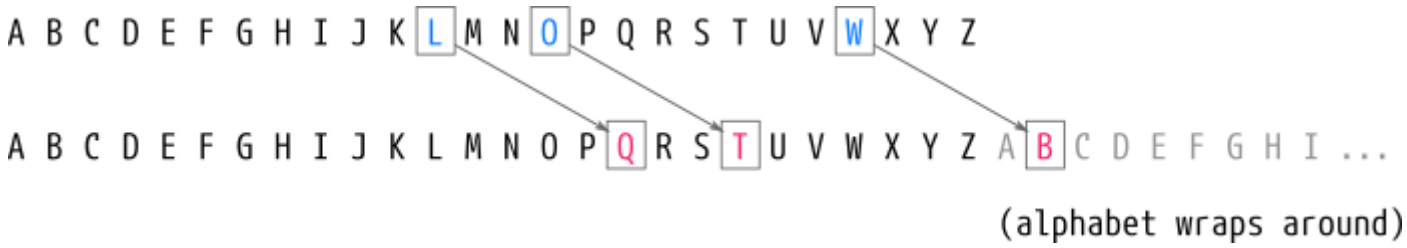
Caesar cipher theory

The Caesar cipher is a simple and widely known encryption technique. The action of a Caesar cipher is to replace each letter in the plaintext with a letter some fixed number of positions down the alphabet.

For example, when the shift is **+5**, every `A` becomes an `F`, every `B` becomes a `G`, and so on:

```
Plaintext : A Light-Year Apart
Ciphertext: F Qnlmy-Djfw Fufwy
```

This is a Caesar cipher with the **right shift of 5**. Note that we assume that the alphabet wraps around, so, for example, each `W` becomes a `B`, and each `Y` becomes a `D`.



Task B. Implementing Caesar cipher encryption

Write a program `caesar.cpp` with functions implementing Caesar cipher encryption:

```
// A helper function to shift one character by rshift
char shiftChar(char c, int rshift);

// Caesar cipher encryption
string encryptCaesar(string plaintext, int rshift);
```

The argument `rshift` is the magnitude of the right shift. For the sake of the lab, you may assume it to be in the range ***0 ≤ rshift ≤ 25*** (although, implementing it correctly for an arbitrary integer shift is also possible, of course).

Your functions should preserve case, and any non-alphabetic characters should be left unchanged. For example,

```
encryptCaesar("Way to Go!", 5) == "Bfd yt Lt!"
```

Feel free to write more additional helper functions when you need them.

The `main` function should ask the user to input a plaintext sentence, then enter the right shift, and report the ciphertext computed using your encryption function.

Example:

```
$ ./caesar

Enter plaintext: Hello, World!
Enter shift    : 10
Ciphertext     : Rovvy, Gybvn!
```

Vigenere cipher theory

In a Caesar cipher, each letter is always shifted by the same number of positions. What if we shifted each letter by a different value? Such a code would be much harder to break. This encryption strategy is known as the Vigenere cipher.

Since each letter of plaintext must be shifted differently, a single right-shift parameter is not sufficient, we need to have a sequence of such shifts. This sequence is determined by a **keyword**, in which

each letter corresponds to specific shift: `a` shifts by ***0***, `b` shifts by ***1***, `c` shifts by ***2***, and so on. The ***n***-th letter of the alphabet will shift by ***n − 1*** to the right.

For example, suppose that the plaintext to be encrypted is:

```
Hello, World!
```

and the keyword is

```
cake
```

The first letter of the keyword is `c`, which is the third letter of the alphabet. That means we shift the first letter of the plaintext `H` to the right by *3-1* = ***2***, which gives `J`.

Then repeat the procedure for the remaining letters. If you reach the end of the keyword, go back and use the first letter of the keyword. If a letter in the plaintext is not alphabetic, skip it without using a shift from the keyword.

Following these steps, the resulting ciphertext is:

```
Jevpq, Wyvnd!
```

Task C. Implementing Vigenere cipher encryption

Write a program `vigenere.cpp`. It should contain a function `encryptVigenere` implementing this cipher:

```
string encryptVigenere(string plaintext, string keyword);
```

You may assume that the *keyword* contains only *lowercase alphabetic* characters `a - z`.

The `main` should implement a testing interface similar to the one in Task B, the user enters the plaintext and the keyword, and the program reports the ciphertext.

Example:

```
$ ./vigenere

Enter plaintext: Hello, World!
Enter keyword  : cake
Ciphertext     : Jevpq, Wyvnd!
```

Task D. Decryption

Implement two **decryption functions** corresponding to the above ciphers. When decrypting ciphertext, ensure that the produced decrypted string is equal to the original plaintext:

```
decryptCaesar(ciphertext, rshift) == plaintext

decryptVigenere(ciphertext, keyword) == plaintext
```

Write a program `decryption.cpp` that uses the above functions to demonstrate encryption and decryption for both ciphers.

It should first ask the user to input plaintext, then ask for a right shift for the Caesar cipher and report the ciphertext and its subsequent decryption. After that, it should do the same for the Vigenere cipher.

Example:

```
$ ./decryption

Enter plaintext: Hello, World!

= Caesar =
Enter shift    : 10
Ciphertext     : Rovvy, Gybvn!
Decrypted      : Hello, World!
```

```
= Vigenere =
Enter keyword  : cake
Ciphertext     : Jevpq, Wyvnd!
Decrypted      : Hello, World!
```

(When reporting decrypted strings, they should be the result of applying decryption functions to the ciphertext, not the original plaintext variable.)

How to submit your programs

Each program should be submitted through Gradescope

Write separate programs for each part of the assignment.
Submit only the source code (.cpp) files, not the compiled executables.
Each program should start with a comment that contains your name and a short program description, for example:

```
/*
Author: your name
Course: CSCI-136
Instructor: their name
Assignment: title, e.g., Lab1A

Here, briefly, at least in one or a few sentences
describe what the program does.
*/
```