Assignment 1

Objectives

In this assignment, you will:

- Practice with C++ basics: functions, loops, input/output, arrays, etc.
- Use C++ libraries you did not see in class.
- Learn about reading from files.

You will do this while implementing code that analyzes famous world literature books!

Deadline

March 28 @ 10:59 PM.

Reading from files (basics)

15 points

This exercise teaches you how to read from files. This is a preparation for the next exercises.
Step 1. Watch the following video about reading from files.
https://static.us.edusercontent.com/files/67GEXThzO3SfLy8NBtVezcPY

Step 2. Modify `ex1.cpp` by writing a program that does the following:

1. **Read** from the user the **name of the file** to be read from.
2. **Count the number of words in the file**. Repeatedly read words as strings, update a counter and then print out the result.
3. **Count the number of characters in the file** (excluding whitespace). While reading the words, add the number of characters in each word to a running sum. Output the result when the reading is over.

To test your program, you can use any of the given files (`empty.txt`, `one-word.txt`, `ten-words.txt` and `mobydick.txt`, `alice.txt`). You can also create any text file in the workspace (click on the + button in the top left of the workspace) and add any number of words to the file.

! There are `215863` words and `1033932` characters in **Moby Dick**.
There are `29647` words and `139645` characters in **Alice in Wonderland**.

To run the program, click on **Run** (bottom right of the workspace) and enter the name of the file whose words you would like to count.

A Word Counter

🍭 `20 points`

In this exercise, you will ask the user to enter a word and the name of a text file and then count how many times this word appears in the text file. To achieve this, you will have to **clean** the words we read from the file.

Step 1. Take a look at the following documentation page for functions from the `<cctype>` library that perform operations on characters.
https://doc.bccnsoft.com/docs/cppreference2015/en/cpp/header/cctype.html

Step 2. Implement function `string clean(string word)` that returns a *"cleaned"* version of `word`. The cleaned version must contain only **lowercase alphabetic** characters (`a-z`).

Examples:

```
             Ibrahim becomes ibrahim
              Hello! becomes hello
             history does not change
ialbluwi@psut.edu.jo becomes ibrahimpsutedujo
                1998 becomes an empty string
```

! **Hint**: Use what you have learned in **Step 1** to implement **Step 2**!
Hint: You can start with an empty "cleaned" string go through the characters of "word" one by one and add it to the "cleaned" version only if it meets the requirements.

Step 3. Write a program that counts how many times a certain word appears in a text file. Proceed as follows:

- Read from the user a word to search for and a name of a file to search in.
- Create a lowercase version of the word.
- Read words from the file one by one. Check if the lowercase version of the search word equals the cleaned version of the read word.
- Output how many times the search word was found!

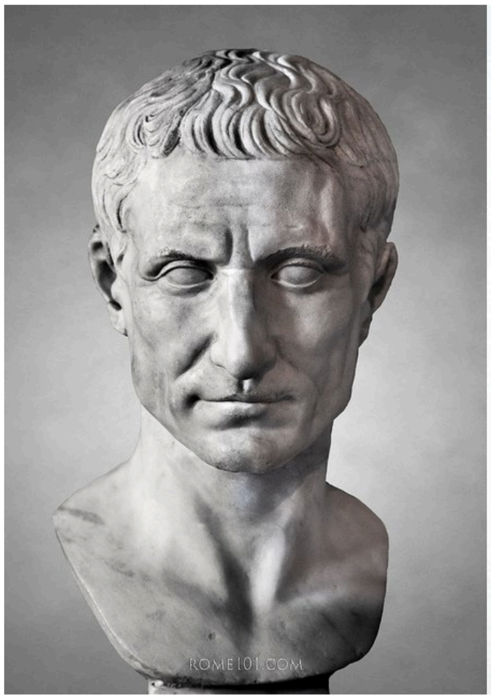
To test your program, you can use any of the given books. Here are some examples:

```
The word "Alice"   appears 385 times in Alice in Wonderland (alice.txt).
The word "rabbit"  appears 44  times in Alice in Wonderland (alice.txt).
The word "Cosette" appears 279 times in Les Miserables (miserables.txt).
The word "Jean"    appears 426 times in Les Miserables (miserables.txt).
```

Optional. Improve your program by making it repeatedly ask for words to search for in the given file. Note that if you would like to read from the beginning of the file again, you need to call function close() on the file input stream object and then call open() again using the name of the file.

If Caesar had a Computer

50 points



Overview

The Caesar Cipher is one of the earliest and most basic substitution ciphers. It is named after Julius Caesar, who used to cipher important military messages by replacing every letter with a different letter.

> "If he had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet, that not a word could be made out. If anyone wishes to decipher these, and get at their meaning, he must substitute the fourth letter of the alphabet, namely D, for A, and so with the others."
> — Suetonius, *Life of Julius Caesar*

The main idea is to shift-rotate the alphabet and match the shifted alphabet sequence with the original sequence. Julius used a right shift of size 3 as shown below. In this scheme, every **A** is substituted with an **X**, every **B** with a **Y**, every **C** with a **Z**, every **D** with an **A**, etc.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W

For example, Caesar would say `PMOBXA ZLOLKX` instead of `SPREAD CORONA`.
In this exercise, you will implement a program that ciphers and deciphers text using this idea.

Requirements

Step 1. Implement the following shifting functions:

- `void shift_right(char a[], int size, int steps)`:
  Shift-rotates the given array to the right by the given steps.

- `void shift_left(char a[], int size, int steps)`:
  Shift-rotates the given array to the left by the given steps.

! **Hint:** Implement first the coding for performing only one shift step (this will involve a loop). Once this step works fine, place the shifting code into another loop that repeats it according to the given number of `steps`.

Step 2. Implement the encryption functions:

- `string replace(char cipher[], string word)`:
  Returns a word that is made of the replacement of every character in `word` with the corresponding character in `cipher`. For example, the character `a` in `word` must be replaced with whatever character is available at index `0` in `cipher`, the character `b` must be replaced with the character at index `1`, etc.

Note the following:

- All characters in `word` must be **letters**. If not, the function returns an **empty string**.
- All the letters in `word` must be in **lowercase**. If not, convert them to lowercase.
- The array is assumed to be of **size 26**.

! **Hint**: To create a mapping between the alphabet and indices, you can subtract `'a'` from the character. This would bring `'a'` to index `0`, `'b'` to index `1`, etc.

- `string encrypt(string word, int steps)`:
  Encrypts the given `word` using a Caesar cipher with the given `steps`.

- `string decrypt(string word, int steps)`:
  Decrypts the given `word` assuming it was encrypted with a Caesar cipher with the given `steps`.

! **Hint**: Begin by creating an array containing the lowercase letters. Use functions `rotate_left()` and `rotate_right()` to rotate the array and then use function `replace()` to encrypt or decrypt the word.

Step 3. Implement a simple program for encryption and decryption using a Caesar cipher. The program should ask the user to enter a word followed by the number of shifts followed by `e` for encrypt or `d` for decrypt. If the user enters an invalid input, output an error message and terminate the program.

Sample Input/Output.

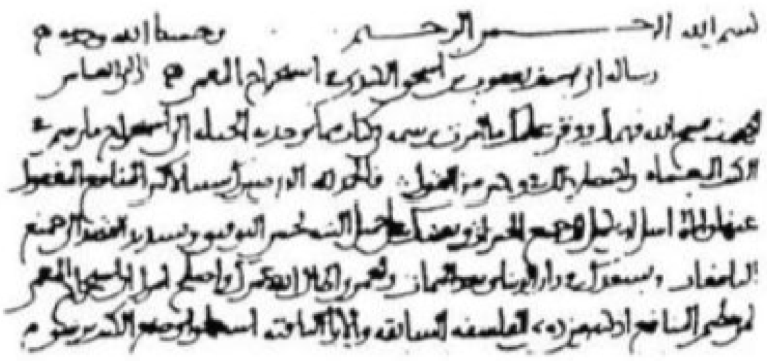
```
Input:  corona 3 e
Output: zlolkx

Input:  zlolkx 3 d
Output: corona

Input:  corona 26 e
Output: corona

Input:  corona -3 e
Output: Invalid number of steps (must be >= 0).

Input:  corona 3 f
Output: Invalid operation (must be e=encrypt or d=dycrypt).

Input:  covid19 3 e
Output: Invalid characters (only lowercase letters are supported).
```

! **Optional:** The requirements in this exercise are meant only to help you practice with manipulating arrays. How would you implement a Caesar cipher without having to shift-rotate an array? The code should be simpler than what this exercise requires!

Optional: Now that you know how to read from files, how about encrypting a whole book?

If Al-Kindi Had a Computer

🍭 15 points



بسم الله الرحمن الرحيم وحسبنا الله ونعم

[illegible]

Overview

Caesar ciphers are very easy to break nowadays. However, this was not necessarily true at Caesar's time. In the 9th century, Al-Kindi pioneered breaking substitution ciphers using letter frequency analysis. The intuition is simple:

- Pick a large Arabic book and analyze the frequency of letters in the book to know which letters typically appear more than which letters in Arabic.
- Analyze the frequency of letters in the encrypted message and try to decrypt the letters based on their frequency.

The work of Al-Kindi is considered to have paved the way for modern cryptanalysis.

In Al-Kindi's time, analyzing the frequency of letters required tremendous effort. If he had a computer, it would have taken him minutes to write a frequency analysis program and fractions of seconds to run it on any book of his choice!
In this exercise, you will write a program that performs letter frequency analysis.

Requirements

Step 1. Copy your implementation of function `clean()` from the previous exercises.

Step 2. Write a program that reads from the user the name of the file to be analyzed. The program should then print out the lowercase letters of the alphabet with their frequencies. Use function `clean()` to clean every word read from the book so that it contains only lowercase letters before analyzing the frequency of its letters.

The output of the program must be formatted as shown below (these are the letter frequencies in `alice.txt`).

```
a    9868
b    1768
c    3046
d    5504
e    15512
f    2390
g    2958
h    7932
i    8676
j    236
k    1299
l    5234
m    2465
n    8095
o    9548
p    1984
q    223
r    6685
s    7301
t    12305
u    4009
v    974
w    2980
x    182
y    2610
z    80
```

Hint: Use an array to keep track of the frequency of the letters, where the number stored at index **0** would be the frequency of letter **a**, the number stored at index **1** would be the frequency of letter **b**, etc.
For every letter that you read, clean it and then go through its characters and update their frequencies.

!

Hint: Use an array to keep track of the frequency of the letters, where the number stored at index **0** would be the frequency of letter **a**, the number stored at index **1** would be the frequency of letter **b**, etc.
For every letter that you read, clean it and then go through its characters and update their frequencies.

Optional. If you would like to see a histogram visualization of the frequencies, perform the following:

- Click on **Terminal** in the lower part of your screen. Click on where it says `"click here to activate the terminal"`.
- Compile the program using the command `g++ ex4.cpp`.
- Run the program and send the output to `gnuplot`. You can cut and paste the following command:

```
./a.out > freq.txt && gnuplot -persist -e "set boxwidth 2; set style fill solid
noborder; plot 'freq.txt' using 2:xticlabels(1) with histogram"
```

- Enter the name of the file to be analyzed and hit Enter. Wait for a pop-up screen to appear showing a histogram.
- If the pop-up screen does not appear (it will likely stop appearing the second time you run the command), click on the mini screen-like button at the top right of your workspace. (the button says "remote app" when you hover over it).

For these steps to work, your program must output only the results and only in the described format. For example, the program should not output a message like `"Enter the file name"`. Use `cin` immediately to read the file name without prompting the user with any message.

!

Optional. Since we have powerful computers that can process millions of characters in seconds, can you think of a much simpler way to break a Caesar Cipher than frequency analysis?

Optional. Read here about what happens when you are unaware of how easy it is to break ciphers like Caesars'!