Project 1 - Ciphers

CS 251, Spring 2024

## Projects in CS 251

Projects in CS 251 are individual work. You may discuss high-level concepts with other students, but any viewing, sharing, copying, or dictating of code is forbidden.

Projects will have a large design component where **you** have to decide how to set things up. We will ramp up slowly on this; so, for this project, we will still give you a lot of guidance. However, *if you get stuck for more than 30 minutes on a bug, you should come to office hours or ask a question on Piazza*. We want this experience to be an enjoyable one, and nobody really enjoys fruitless debugging for hours on end.

## Restrictions

This project is intended as a “warmup” for a (re)introduction to C++ after break. As such, we have the following restrictions:

* Don’t add any additional headers beyond the ones we have given:
  + <cctype> (functions on chars)
  + <fstream> (reading files)
  + <iostream> (console input, output)
  + <string>
  + <vector>
* Don’t use structs, pointers, new, or malloc.
* Don’t add global variables. (Define “global” variables in main, and use pass-by-reference.)
  + The one exception to this is the given ALPHABET constant.

If you find yourself trying to break these restrictions, you’re probably trying to do something we didn’t intend and might be making the project harder. These aren’t intended to be “gotchas”; we’re just trying to limit the scope of what you have to deal with. If something isn’t clear, please ask!

## Logistics

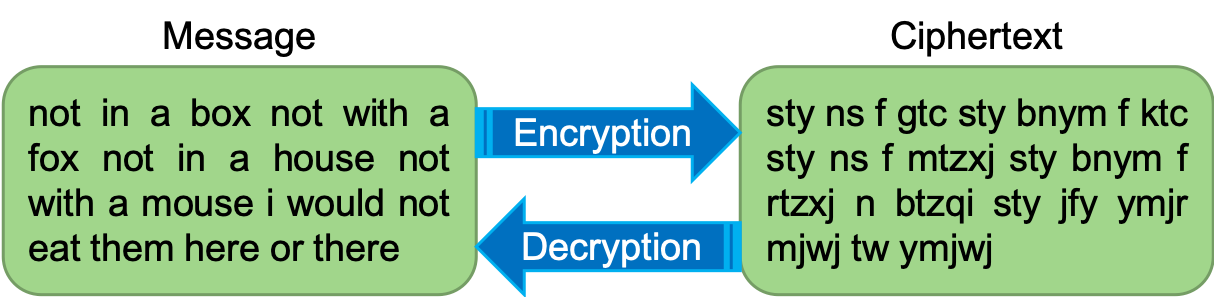
* Due: **Thursday 2/1, 11:59 PM**
* Submit to: zyBooks, Gradescope (main.cpp)
* Use grace tokens: [FORM](https://docs.google.com/forms/d/e/1FAIpQLSejjzmTltSkhja_QwK8MXAk4Vw3u3eJw0E9b2rsBknDdonq7A/viewform)
  + This requires setting up a UIC Google account. If you have not yet done so, visit <https://learning.uic.edu/resources/virtual-collaboration/google-workspace/>.

**The beginning of this project will give you a lot of explicit guidance. The end of the project will give less guidance, and more closely resemble later CS 251 projects.**

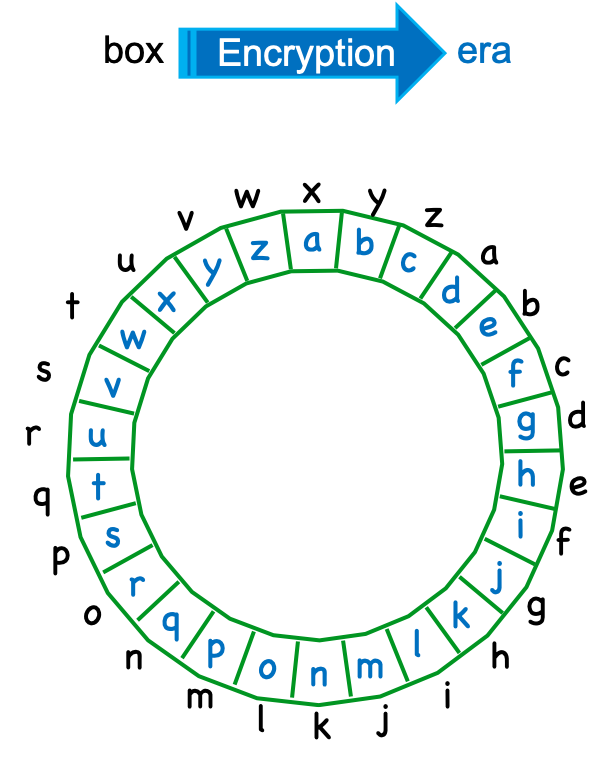
## Tasks

### Task: Caesar Cipher Encoder

A **cipher** is an algorithm for encrypting or decrypting text. We call the unencrypted text “plaintext” and the encrypted version “ciphertext”. Here is an example of a **Caesar cipher**, which you will implement in this project.



Notably, the original text is readable, but the encrypted version is not. A Caesar Cipher is computed by “shifting” each letter of the text forward in the alphabet by a constant number, as if it was in a circle. For example, the following “ring” shows the mapping for a Caesar Cipher where each letter is shifted forward by 4.



Ultimately, the goal for this part is to take in a piece of text (the “plaintext”) and a number of letters to rotate by (the “key”), and output the encrypted text (the “ciphertext”).

#### Subtask: String Utility Functions

First, we’re going get some practice by implementing some small string utility functions that will be useful:

**int findIndexInAlphabet(char c)**  
You should use the string::find method here.

In some programming languages (Python), you can’t have two methods named the same thing, even if they take a different argument. In languages like C++, this is allowed and called **method overloading**. Next, you will implement two versions of the rot method (short for rotate): one for if we want to rotate a character and another for rotating a whole string:

**char rot(char c, int amount)**

You should convert the input character into a number. Then, add the rotation amount to that number, making sure to wrap the number around so it is between 0 and 25. Then, convert it back to a character. Assume that the input character is an uppercase letter.

**string rot(string line, int amount)**

You should rotate each letter by the rotation amount (converting lowercase to uppercase), and return a new string with all the characters rotated. Don’t rotate non-letter characters. You’ll probably find some methods from the [<cctype> header](https://cplusplus.com/reference/cctype/) to be helpful here, such as [isalpha](https://cplusplus.com/reference/cctype/isalpha/) and [toupper](https://cplusplus.com/reference/cctype/toupper/).

You will need to change the return statement.

**You might have noticed that a lot of the previous information was duplicated between this guide and comments in the code. Rather than repeating the exact same thing in the guide and the comments, we will often split the information across the two sources. Usually, the high-level requirements and hints will be in the guide and the details and examples will be in the code itself. We recognize that you may not be used to this, but this is the way the real world works: there is no one source of information in real projects.**

#### Subtask: Encrypt with Caesar shift

Now, it’s time to put these methods together. For this command, you should prompt for input twice:

* “Enter the text to encrypt:”
* “Enter the number of characters to rotate by:”

Then, output the encrypted text.

Implement the “Encrypt with Caesar Cipher” command as described above.

You should always use getline in this project, since it reduces the number of possible bugs when dealing with input streams. You’ll also find [stoi](https://cplusplus.com/reference/string/stoi/) useful; don’t worry about the case where input that should be a number isn’t a number.

Here, we’ve demonstrated function decomposition: breaking functions down to accomplish smaller tasks, then combining them to accomplish a larger task. Later in the project, you’ll likely find some of the smaller functions to be useful again, and (we think) you’ll find at least one other place where writing a new helper function will be useful!

### Task: Vigenère Cipher Encoder

The Vigenère cipher is like a Caesar cipher, except that each letter of the plaintext is encoded with a different shift, which comes from a “key”. Let’s say that we’re encrypting the text “program”, with the key “abc”. The encryption would look like this:

| **Plaintext** | P | R | O | G | R | A | M |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Key (shift)** | A | B | C | A | B | C | A |
| **Cipher** | P | S | Q | G | S | C | M |

The first character in the plaintext, ‘P’, is shifted by the first character in the key, ‘a’. ‘A’ is at index 0 in the alphabet; so the output is ‘P’ shifted by 0, or ‘P’. The next character in the plaintext (‘R’) is shifted by the next character in the key, ‘B’. ‘B’ is at index 1, so the output is ‘R’ shifted by 1, or ‘S’. When the key is shorter than the plaintext, we go back to the beginning of the key.

For the next command, you should prompt for input twice:

* “Enter text to encrypt:”
* “Enter the Vigenère key:”

Non-alphabetic characters in the plaintext should be left alone. For the plaintext, convert letters to uppercase. For the Vigenère key, convert to uppercase and remove characters that are not letters.

Implement the “Encrypt with Vigenère” command as described above.

### Task: Caesar Cipher Decoder

In the first part, we **encrypted** a piece of text using a Caesar cipher. Now, we’re going to go in the opposite direction – decrypting the ciphertext back into the plaintext, without knowing which value was used. For a Caesar cipher, this means we need to rotate each letter by , where was the shift used originally. Since there are only 26 possible shifts, we can just try all of them and see which one “works best” – this is called **brute forcing** a solution.

How do we know which shifts work “better”, though? We’ll compare against a dictionary and check how many words in the decrypted text are actually words in the dictionary.

Your general strategy for implementing the “Decrypt Caesar Cipher” command will be the following:

* Load the dictionary (dictionary.txt) ***once*** in the main method
* Ask for text to decrypt (“Enter the text to Caesar-cipher decrypt:”)
* Break the text into a vector of words (separated by spaces), converting lowercase to uppercase and removing non-letters.
* For each possible rotation of the entire sentence:
  + If more than half (>1/2) of the words in the rotated text are also in the dictionary:
    - Print out the entire decrypted text.
    - In contrast to the words vector, convert lowercase to uppercase and leave non-letter characters alone.
* If no lines were output, print “No good decryptions found”

Implement the “Decrypt Caesar Cipher” command as described above. We recommend approaching this in 2 parts: when the input text is one word, then expanding on that to handle when the input text is multiple words.

For debugging, we’ve provided a dictionary-small.txt. You may also notice that the full dictionary.txt is a bit weird in what it does or doesn’t contain, and therefore may not output all decryptions that you think are valid. Make sure to check the dictionary file!

We won’t be implementing an algorithm to decrypt a Vigenère cipher – several exist, but the interesting parts are too complex for Project 1.

### Cut Task: Substitution Cipher Encoding + Decoding

If you finish the project early, and would like to see the part we removed, email Ethan. We will not give extra credit for completing it, but (we think) it’s an interesting exercise that’s unfortunately too long for the adjusted timeline.

## Submitting

The project is due **11:59 PM on 2/1**. You should submit to **two** distinct places:

* Project 1 on zyBooks
  + Main suite of correctness tests
* Project 1 on Gradescope
  + Style grading
  + Other manual verification

The test suite will be added to zyBooks on **Wednesday night 1/24**, and the Gradescope submission will be available around the same time. Before then, you can manually compare against the provided solution executable.

## Grading Breakdown

Out of 100 points:

| **Item** | **Points** |
| --- | --- |
| String Utility Functions | 5 |
| Caesar Encryption | 15 |
| Vigenere Encryption | 15 |
| Caesar Decryption (single word) | 25 |
| Caesar Decryption (multi-word) | 30 |
| Manual grading (style, function decomposition) | 10 |

The grading is intentionally backloaded towards the end – we expect all students to complete the project.

#### Gradescope Rubric (Manual Grading)

**Function Decomposition**

* 5 points: Function decomposition well-applied to all aspects of code.
* 3 points: Attempted function decomposition, but flawed (e.g. significant code is not decomposed, repeated subroutines in different places, etc.)
* 0 points: No additional functions added; everything in main.

**Style**

* 2 points: Code is styled consistently; for example, using the VSCode formatter.
* 1 point: Code is reasonably styled, but there are significant stylistic issues (e.g. indentation, line length, spacing, etc.)
* 0 points: No credit (e.g. entire program is on one line)

**Documentation + Commenting**

* 3 points: Code is well-documented with descriptive variable names and comments, but not overly documented.
* 1.5 points: Code is partially documented, due to a lack of comments and/or poor naming; or code is overly documented with unnecessary comments.
* 0 points: Code has no documentation or appropriate names.

### Sample Execution

User inputs are highlighted in **red**.

➜ ./ciphers\_solution

Welcome to Ciphers!

-------------------

Ciphers Menu

------------

C - Encrypt with Caesar Cipher

D - Decrypt Caesar Cipher

V - Encrypt with Vigenère

X - Exit Program

Enter a command (case does not matter): **c**

Enter the text to encrypt:

**Hello, world!**

Enter the number of characters to rotate by:

**10**

ROVVY, GYBVN!

Ciphers Menu

------------

C - Encrypt with Caesar Cipher

D - Decrypt Caesar Cipher

V - Encrypt with Vigenère

X - Exit Program

Enter a command (case does not matter): **v**

Enter text to encrypt:

**Attack at dawn, OK?**

Enter the Vigenère key:

**le mon**

LXFOPV EF RNHR, AY?

Ciphers Menu

------------

C - Encrypt with Caesar Cipher

D - Decrypt Caesar Cipher

V - Encrypt with Vigenère

X - Exit Program

Enter a command (case does not matter): **d**

Enter the text to Caesar-cipher decrypt:

**nk**

BY

HE

IF

Ciphers Menu

------------

C - Encrypt with Caesar Cipher

D - Decrypt Caesar Cipher

V - Encrypt with Vigenère

X - Exit Program

Enter a command (case does not matter): **d**

Enter the text to Caesar-cipher decrypt:

**ai nyug**

GO TEAM

Ciphers Menu

------------

C - Encrypt with Caesar Cipher

D - Decrypt Caesar Cipher

V - Encrypt with Vigenère

X - Exit Program

Enter a command (case does not matter): **x**

## Changelog

* 1/23, 1PM: Assignment released
* 1/23, 7:30 PM: Skeleton code updated to make documentation for rot(string, int) consistent with project guide.
* 1/24, 12 noon: Updated signature of findIndexInAlphabet from size\_t to int.
* 1/25, 1:30 PM: Marked user input in the sample execution
* 1/25: 5:40 PM: Corrected typo in Vigenere prompt
* 1/26, 10 PM: Added Gradescope rubric

## Acknowledgements

Assignment by Adam Blank, adapted for CS 251 by Ethan Ordentlich.