Ciphers

1 - Introduction

In this project, you will implement three classical ciphers: the Caesar, Vigenère, and Rail Fence ciphers. This project includes four C++ classes:

- Base class: Cipher,
- Derived classes: CaesarCipher, VigenereCipher, and RailFenceCipher.

The Cipher class serves as an abstract base class that defines common behavior for the derived cipher classes. Specifically, it declares two pure virtual member functions: encode() and decode(). By making the Cipher class abstract, we enforce that encode() and decode() must be implemented in the derived classes, allowing each cipher to define its unique approach. The Cipher class only provides the interface for encoding and decoding operations.

The main() function demonstrates polymorphism by creating a Cipher class pointer (named cipher) that can point to any of the derived cipher objects (CaesarCipher, VigenereCipher, or RailFenceCipher). This allows for dynamic calling based on the type of object assigned to the pointer, showing the flexibility and extensibility of the design.

2 - Background

2.1 - ASCII (American Standard Code for Information Interchange)

ASCII is a character encoding standard used to represent text in computers and communication systems. Each character in ASCII is assigned a unique number from 0 to 127. For example, the letters 'A' to `Z` are represented by the decimal numbers 65 to 90, respectively.

2.2 - Caesar Cipher

This is one of the simplest and most well-known substitution ciphers. It works by shifting each letter in the input string a fixed number of places down or up the alphabet. For reference, whitespace remains unaffected. It’s easy to implement but also easy to break. Here is a Caesar encoding (shifting up in this project) example:

Input string:	L	O	N	G		D	A	Y
Input shift:	3							
Output string:	O	R	Q	J		G	D	B

See https://en.wikipedia.org/wiki/Caesar_cipher for more information.

2.3 - Vigenère Cipher

This is a more complex substitution cipher that uses a key to determine the shift for each letter. The key is repeated to match the length of the input string, and each letter in the input string is shifted according to the corresponding letter in the key. This makes it much harder to break compared to the Caesar cipher, as the shifts vary throughout the text. Here is a Vigenère encoding example when the key is END:

Input string:	L	O	N	G		D	A	Y
Input key:	E	N	D					
Converted key:	E	N	D	E		N	D	E
Shift from key (distance from A)	4	13	3	4		13	3	4
Output string:	P	B	Q	K		Q	D	C

See https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher for more information.

2.4 - Rail Fence Cipher

This is a transposition cipher that arranges the input string in a zigzag pattern across multiple "rails". The input string is then read off line by line to create the encoded string. Here is a Rail Fence encoding example when the number of rails is 3:

Input string:	L	O	N	G		D	A	Y
Intermediate (2D array with 3 rows):	L	*	*	*		*	*	*
	*	O	*	G	*	D	*	Y
	*	*	N	*	*	*	A	*
Output string:	L		O	G	D	Y	N	A

See https://en.wikipedia.org/wiki/Rail_fence_cipher for more information.

3 - Tasks

The files you should implement for this project are: **Cipher.h**, **CaesarCipher.cpp**, **VigenereCipher.cpp**, **RailFenceCipher.cpp**, and **main.cpp**. For reference, the Cipher class is quite simple, so there is no separate Cipher.cpp file. Check the comments marked with TODO and refer to “Section 6. Grading” for more details about the tasks.

In this project, **it is assumed that all characters are either uppercase letters or whitespace**, which simplifies the implementation of the code.

Below, you will find the pseudocode that can guide you in implementing the encode() and decode() member functions.

To validate each character (ensuring it is either an uppercase letter A-Z or whitespace), **you must use the isValidCharacter() member function** from the Cipher class.

3.1 - Caesar Cipher Encode

```
For each character in the input string:
    If the character is valid:
        Convert the character from ASCII code to an alphabet number (by subtracting 65 or 'A')
        Shift the number by adding the member shift
        If the shifted number > 25:
            Subtract 26 from the number
        Else if the shifted number is < 0:
            Add 26 to the number
        Reconvert the number to ASCII code (by adding 65 or 'A')
        Add (concatenate) this code to the encoded string
        // if the input character is a whitespace (' '), the code for this iteration is a whitespace
    Else:
        Return False
Store the encoded string in the member `encoded`
Return True
```

3.2 - Caesar Cipher Decode

The decoding algorithm is almost the same procedure as the encoding algorithm. Think about the differences to implement the decode() member function.

3.3 - Vigenère Cipher Encode

```
Convert (extend) the original input key
For each character in the input string and each character in the converted key:
    If the input string character is valid:
        Add the key character and the input string character together
        Take this value mod 26
        Reconvert the number to ASCII code (by adding 65 or 'A')
        Add (concatenate) this code to the encoded string
        // if the input character is a whitespace (' '), the code for this iteration is a whitespace
    Else:
       Return False
Store the encoded string in the member `encoded`
Return True
```

3.4 - Vigenère Cipher Decode

The decoding algorithm is almost the same procedure as the encoding algorithm. Think about the differences to implement the decode() member function.

3.5 - Rail Fence Cipher Encode

```
Dynamically allocate memory for a 2D array (rail x input string length)
Set a variable 'goingDown' to 1 (to control whether we move down or up in the zigzag pattern)
Set a variable 'row' to 0 (to represent the current rail)
For each character in the input string:
    If the character is valid:
        Place the input character in the 2D array at that location
```

```
        Update 'row'
        Update ‘goingDown’
    Else:
        return False
Read the array by concatenating characters from each row in order
Store the encoded string in the member `encoded`
Deallocate the 2D array
Return True
```

3.6 - Rail Fence Cipher Decode

```
Dynamically allocate memory for a 2D array (rail x input string length)
Make the zigzag pattern in the 2D array (e.g., mark elements with a specific symbol) // you may need to use variables ‘goingDown’ and ‘row’ for this
For each element in the 2D array:
    If the character is valid:
        If the element is marked with the specific symbol:
            Place the input character in the 2D array at that location
    Else:
        Return False
Read the 2D array to extract the original string // you may need to use variables ‘goingDown’ and ‘row’ for this
Store the decoded string in the member `decoded`
Deallocate the 2D array
Return True
```

4 - Compile and Test

Type the following commands on Terminal.

```
g++ -o main *.cpp *.h
./main
```

Inputs:

- Name of a cipher to use (Caesar, Vigenere, or RailFence)
- Specific cipher info (shift for Caesar, key for Vigenere, and rail for RailFence)
- Action (DECODE/ENCODE)
- Input string to encode/decode **// all characters must be uppercase letters or whitespace**

```
#input example for rail fence encoding
#you can find more input examples in the inputs folder
RailFence
4
ENCODE
THANKSGIVING BREAK
```

Your results will be saved in the generated **result.txt** file. Please open this file to verify if your results are correct. The expected outputs for the above inputs are as follows:

```
Encoding message using a Rail Fence Cipher encoder!
Encoded Message is:
TG HSIGBKAKVNRANIE
```

To check all cases together, you can use the following commands:

```
chmod 777 run.sh
./run.sh
```

You must see the following messages:

```
Encode Caesar - PASS
Decode Caesar - PASS
Encode Vigenere - PASS
Decode Vigenere - PASS
Encode RailFence - PASS
Decode RailFence - PASS
```

5 - Submit

- Please upload your **Cipher.h, CaesarCipher.cpp, VigenereCipher.cpp, RailFenceCipher.cpp, and main.cpp** files (5 files) to myCourses > Assignments > Project3.
- Do not submit *.docx, *.pdf, *.txt, or *.zip file.
- Do not change other files.
- Only one student from each team can submit.

6 - Grading

- [91 pts] - implementations for the TODO parts
    - [8 pts] 4 parts in main() of main.cpp
    - [8 pts] 4 parts in the Cipher class of Cipher.h
    - [5 pts] constructor of the CaesarCipher class
    - [10 pts] encode() of the CaesarCipher class
    - [10 pts] decode() of the CaesarCipher class
    - [5 pts] constructor of the VigenereCipher class
    - [10 pts] encode() of the VigenereCipher class
    - [10 pts] decode() of the VigenereCipher class
    - [5 pts] constructor of the RailFenceCipher class
    - [10 pts] encode() of the RailFenceCipher class
    - [10 pts] decode() of the RailFenceCipher class
- [9 pts] - program execution
    - **The program execution portion of the grade is ALL or nothing.**