Encryption 101

Due Feb 24, 2020 by 11:59pm **Points** 20
Available Feb 17, 2020 at 11:59pm - Feb 24, 2020 at 11:59pm 7 days

This assignment was locked Feb 24, 2020 at 11:59pm.

0. READ THE INSTRUCTIONS BELOW **COMPLETELY AND CAREFULLY** BEFORE PROCEEDING.

0.1 THE CLASS LECTURE ON ENCRYPTION HAS MORE DETAILS ON HOW THE CIPHERS WORK. READ IT BEFORE PROCEEDING.

INTRODUCTION

1. In this assignment, you will be implementing a program that encrypts and decrypts using several different ciphers, as discussed in class. The ciphers are

  a) substitution cipher,

  b) Caesar cipher,

  c) ROT13 cipher,

  d) Running Key cipher, and

  e) Vigenere cipher.

2. Do not change the provided skeleton code. You may add new code to the provided files, but the existing code MUST NOT BE CHANGED. In particular, the main.cc function should not be changed AT ALL (i.e., do not **change** or **add** new code), since it will be used to test your implementation. Your code **must conform** to the way it is used in main.cc.

REQUIREMENTS:

3. Substitution cipher (implemented in cipher.h and cipher.cc) class has already been defined (but not implemented) for you. The other cipher classes MUST

  a) inherit from the substitution cipher, or

  b) inherit from one of the four other cipher classes (e.g., Caesar inherits from Substitution, ROT13 inherits from Caesar).

4. Each new cipher class MUST have **a constructor**, **a default constructor** (except the ROT13 cipher, as described below), and a **destructor** .

a) Each **constructor** should accept the input appropriate for the cipher:

i) substitution - cipher alphabet - must contain **every letter** in the alphabet, and **only once,** and they must all be **lower case.** It should contain **no other characters** (e.g., no space, punctuation, etc.). Cipher alphabet length should always be 26.

ii) Caesar - **positive shift/rotation/offset integer.** This value can be **larger than 26**, in which case it has rotated completely one or more times.

iii) ROT13 - **no input** required for constructor (see default constructor below).

iv) Running Key - **a vector of strings**. The vector represents the cipher "book," and each element in the vector is a "page" from the book. Each page should be a **non-empty** string. This string should consist of **only lower case letter or spaces.**

v) Vigenere - **a string** representing the **key word.** The key word should consist of **only lower case letters**.

b) Each **default constructor** should not accept any inputs.

i) ROT13: ROT13 should **only have a default constructor**, and it should function properly as a ROT13 cipher with the default constructor.

ii) Other ciphers: default constructor should initialize the object so that encrypting any plain text should return a cipher text that is **identical to the plain text**. For example, for Caesar, setting the shift to 0 will achieve this. **YOU CANNOT SIMPLY RETURN THE PLAIN TEXT - IT MUST STILL ENCRYPT THE TEXT using the proper method.**

b) the **destructor** should free all memory. **Valgrind** will be used to make sure there are **no memory leaks.**

5. Each cipher must also provide **encrypt** and **decrypt** functions. They can either be implemented explicitly, or implicitly using inheritance. This is a design decision, but the decision should be made to minimize/modularize your code.

i) Encrypt: Assume that the input plain text will always be **a) lower case letter, b) upper case letter, or c) a space.** Space should not be encrypted (i.e., if the plain text is "hello world", the encrypted text should look something like "abcde fghij". Upper case letter should be encrypted to an upper case letter (you don't have to, but you will find that meeting the decrypt requirement below will be difficult otherwise).

ii) Decrypt: This should take in the encrypted text and return the origin plain text. It should

a) **retain the case** (i.e., if the input plain text has an upper case letter, the decrypted plain text should have that letter as upper case.

b) **retain the space** - If the plain text had spaces, the decrypted text should still have those spaces in the correct place.

6. Data members in all cipher classes should be hidden using the **Cheshire smile.**

7. Each cipher should also check to make sure that the provided cipher text, cipher alphabet, rotation etc. meets the requirements before initializing the object with it. For example, see the is_valid_alpha function in cipher.cc for the substitution cipher. Below shows just **A COUPLE OF EXAMPLES.**

a) For the Running key cipher, each "page" of the book should be a non-empty string. If it is not, exit with EXIT_FAILURE.

b) When encrypting with Running Key, make sure the key (not counting the spaces) is longer than or equal to the text you are trying to encrypt (not counting the spaces). If it is not, exit with EXIT_FAILURE. .

TESTING:

8. After implementing your code, test it against the provided test files. For example, by executing:

./cipher -m c -i caesar.txt -o caesar_out.txt

a) In the above example, '**-m c**' means you are using the Caesar cipher method.

b) '**-i caesar.txt**' specifies that caesar.txt is the input file and it should contain the shift distance in the first line, and the plain text you want to encrypt in the second line.

c) '**-o caesar_out.txt**' specifies that caesar_out.txt is the output file and it should contain the cipher text (text after encryption) in the first line, and the decrypted text in the second line.

d) The program will also generate **def_caesar_out.txt** which contains the cipher text and decrypted text when the Caesar cipher object was created using the default constructor (i.e., encryption is done using 0 shift).

9. See test3_error directory to see examples of errors. In those cases, the code exited with EXIT_FAILURE, so no output file was generated. The output to the console (i.e., cout or cerr) is shown. In your code, make sure you generate a meaningful error message when it exits with EXIT_FAILURE.

GRADING:

10. For this homework, some points will also be assigned depending on how well the code uses inheritance and polymorphism to minimize/modularize the code (see the rubric).

11. Do the homework in your own repo, commit, and **push to Bitbucket**. If you do not push to Bitbucket, the TA and I cannot see the code, and it will be considered a late assignment (i.e., not graded).

12. Due to the open-ended nature of this homework, there may be many questions. So a) **start early**, and b) **post any questions about the ciphers' functionality** (or any other question regarding the homework) in the Discussions section on Canvas.

Optional Reading:

11. The code uses a library (getopt.h) for reading optional parameters. Read the code and the tutorial (**https://azrael.digipen.edu/~mmead/www/Courses/CS180/getopt.html (https://azrael.digipen.edu/~mmead/www/Courses/CS180/getopt.html)**) to get an understanding of how it works. You may find this feature useful in the future.

Rubric

Criteria	Ratings		Pts
Test Input 1 1 point for each correct cipher implemented.	**5 pts Full Marks**	**0 pts No Marks**	5 pts
Test Input 2 1 point for each correct cipher implemented.	**5 pts Full Marks**	**0 pts No Marks**	5 pts
Comments, Readability, and Understandability	**3 pts Full Marks**	**0 pts No Marks**	3 pts
Overall program design You will also be graded on how the program was designed - i.e., how well it uses inheritance and polymorphism to modularize your code	**7 pts Full Marks**	**0 pts No Marks**	7 pts
			Total Points: 20