

# cis15-ozgelen, assignment III, part 2 (OPTIONAL)

## instructions

- Create a mail message addressed to *ozgelen@sci.brooklyn.cuny.edu* with the subject line **cis15 hw3 part 2**. Attach ONLY the **.cpp** source code file created below and name it `< yourname > _cis15_a3p2.cpp`.

## program description

For this assignment, you will develop a program that builds on the one you write for HW III Part 1.

In HW III, Part 1, you wrote a program that used the Caesar cipher to encrypt a string. In this assignment you will write a program to decrypt a string that was encrypted using the Caesar cipher.

To do this, you will *reuse* a number of the classes that you wrote for Part 1 of the assignment. Reuse of code is good software engineering.

As before, the intermediary test programs are for your benefit as a developer and should not be submitted. These *unit tests* are created to let you test and debug the individual units of a complex program.

### A. the “message” class

Your final program will, once again do the cryptography character by character, but also needs to deal with whole strings. The first thing to do is to adapt the `message` class to do this.

1. You need to add one function member to `message`:

- `void addToContent( char next );`

This function adds the character *next* to the data member *content*.

Since you already have a function member `void addToContent( string next )`, this new function *overloads* `addToContent`.

2. Create a `main()` for testing that creates an object `myMessage` of the `message` class, prompts for a string, reads one in, sets *content* of `myMessage` to the string that was entered, prompts for a character, reads that in, adds it to the *content* of `myMessage`, and then uses the `print` method of `myMessage` to print the string out.

This `main` is for unit testing, you don't hand it in.

### B. the “caesar” class

1. Add to the class `caesar` a function `decrypt` that takes a `char` as an argument and returns a character. `decrypt` should reverse the Caesar cipher.

`decrypt` performs a static cast on *c* to make it into an integer, subtracts the shift from the integer (remembering to apply modulus arithmetic so that applying a shift of 2 to the letter z will give b), and then casts the result back to a character.

2. Modify the `main()` that you created in part A to prompt for and read in another character, create an instance of the `caesar` class, and use the `encrypt` method to encrypt that character.

Again, this `main()` is for unit testing only. Don't hand it in.

### C. the “fileHandleString” class

1. Create a `fileHandleString` class that will be used as an interface between your program and an output file. It should have the following data and function members:
  - A data member of type `ifstream`
  - A constructor that opens the file `output.txt` for output using `ios::in`.
  - A destructor that closes the file `output.txt`.
  - A function member `getFromFile` that reads a string from the file `output.txt`.

This is obviously very similar to the `fileHandleChar` you wrote for Part 1.

2. Now modify the `main()` from the previous step so that it creates an object of type `fileHandleString` and uses it to read a string from a file (which you can create using your solution to Part 1).

### D. the “encryption” class

1. You need to add the following to the `encryption` class:

- A second data member of type `message`
- A data member of type `fileHandleString`.
- A function member called `decryptThis`

This function takes an integer as its input, and uses it, along with methods from the `message`, `caesar`, and `fileHandleString` to read in a string from a file, decrypt the string by shifting the characters the amount of the integer.

The reason you need two data members of type `message` is so you can use one to hold the string read from the file (and send it to the decryption routine) and one to build up the decrypted string, using the `addToContent` function that takes a `char` argument.

- A function member `printDecrypted` that prints the decrypted string.

2. Now modify the `main()` from the previous step so that it creates an object of type `encryption`, passes it the cipher text from a file and the encryption key that the user enters, and then uses `decryptThis` to decrypt the string and print the decrypted string.

### E. a menu for the user

1. Modify `main()` so that it offers the user a menu. Choosing one option allows the user to enter plain text and a key, and writes the cipher text to a file.

Choosing the other option allows the user to enter a key, and the program uses the key to decrypts the cipher text.

### F. a revised “encrypt”

The `encrypt` that you wrote for Part A is not very realistic. For extra credit, make it more realistic.

1. First modify the `encrypt` function so that it only generates upper case (capital) letters. This makes it harder to see where words begin.
2. Next, modify `encrypt` so that it generates a cipher text that is blocks of four letters followed by a space. In other words, `encrypt` removes any spaces from the cipher text and then introduces a space after every four characters.