CIS 15 Spring 2010, Assignment IV

Instructions

- This is the assignment for Unit IV.
- It is worth 10 points.
- **It is due on Sunday April 11th** and must be submitted by email (as below).
- **Follow these emailing instructions:**
  1. Create a mail message addressed to **parsons@sci.brooklyn.cuny.edu** with the subject line **CIS15 HW4**.
  2. Write your name, that is the name under which you registered for the course, in the email. When I get an email from `deathmetal@aol.com` or `pinkprincess@yahoo.com`, I can usually guess whose program it is, but that is not as good as *knowing* whose program it is.
  3. Attach ONLY the **.cpp** source code file created below.
  4. Failure to follow these instructions will result in points being taken away from your grade. The number of points will be in proportion to the extent to which you did not follow instructions . . . (which can make it a lot harder for me to grade your work)

Program description

For this assignment, you will develop a program that performs some simple cryptography. The program will allow the user to (1) enter a string and have the program encrypt the string send the output to a file; and (2) have the program read an encrypted message from a file and decrypt it.
To write this program, you will create four classes, each with multiple data and function members, and a `main()`. Each class is described in detail below, with step-by-step instructions for developing the various components of each class and testing the components individually. In the end, you'll write the `main()` and put all the pieces together—and this final product is what you'll submit.
The intermediary test programs are for your benefit as a developer and should not be submitted. Incidentally, these are called *unit tests* and are created to let you test and debug the individual units of a complex program. By the end of the semester, you should have gained the skills and experience to devise and construct your own unit tests, without me giving you step-by-step instructions.

A. The "message" class

Your final program will need to accept input as a string, but, because of the way that we will encrypt the message, it needs to be able to handle the string as a sequence of characters. The `message` class provides this functionality.

1. Create a `message` class. The class should have a `string` data member called $content$ and a `int` data member called $location$.

   The class should have two constructors:

   - One constructor should take no arguments. It should initialise $content$ to the null string `""` and $location$ to 0.
   - The other constructor should take a `string` as an argument. It should initialise $content$ to the string that is its argument and $location$ to 0.

   The class should have four function members:

   - `void print() const;`
     This function prints out the data member $content$.

- `void addToContent( string next );`
  This function adds the string *next* to the data member *content*.
- `char getNextChar()`
  This function is used to return a character from *content*. The first time that `getNextChar()` is called, it returns the first character from *content*, the second time `getNextChar()` is called, it returns the second character from *content* and so on.
  Use the data member *location* to keep track of which character to return next, and remember that a string is just an array of characters.
- `bool empty()`
  This function returns true if you have read to the end of the string *content*. If you are at the end of the string, then the next character to read will be the null character `'\0'`.

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, 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. Create a class called `caesar` that has one `int` data member called *shift*.
   The class should have two constructors:
   - One constructor should take no arguments. It should do nothing.
   - The other constructor should take an `int` as an argument, and set *shift* to the value of that `int`.

   The class should have two function members:
   - One function member *setShift* has an `int` as an argument, and sets *shift* to the value of that argument.
   - The other function member *encrypt* takes a character c as an argument and returns a character.
     encrypt applies the Caesar cipher to its argument. It performs a static cast on c tomake it into an integer, adds the shift to 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 a 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 "fileHandleChar" class

1. Create a `fileHandleChar` 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 `ofstream`
   - A constructor that opens the file `message.txt` for output using `ios::app` so that each time the file is used, new data gets added to the end of the file.
   - A destructor that closes the file `message.txt`.
   - A function member `passToFile` that takes a character as an argument and writes that character to the file `message.txt`.
2. Now modify the `main()` from the previous step so that it creates an object of type `fileHandleChar` and uses it to write the encrypted character to a file.

D. The "encryption" class

1. Create an `encryption` class with the following :

   - A data member of type `caesar`
   - A data member of type `message`
   - A data member of type `fileHandleChar`.
   - A function member called `encryptThisString`
     This function takes a string and an integer as its input, and uses these, along with the methods from `message` and `caesar` to encrypt the string by shifting the characters the amount of the integer

2. Now modify the `main()` from the previous step so that it creates an object of type `encryption`, passes it the string (called the "plain text") and the integer (called the "encryption key") that the user enters, and then uses `encryptThisString` to encrypt the string and send the encrypted string (the "cipher text") to the file `output.txt`.

E. Extending the "message" class

After the last step, you have a program that can encrypt a message. You will now add the functionality to decrypt a message, that is, to convert it from coded form back to plain text.
The first thing to do is to adapt the `message` class.

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.

F. Extending 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 `b` will give `z`), and then casts the result back to a character.

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

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

G. The "fileHandleString" class

1. Create a `fileHandleString` class that will be used as an interface between your program and an input file. It should have the following data and function members:

   - A data member of type `ifstream`
   - A constructor that opens the file `message.txt` for output using `ios::in`.

- A destructor that closes the file `message.txt`.
- A function member `getFromFile` that reads a string from the file `message.txt`.

This is obviously very similar to the `fileHandleChar` you wrote above.

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).

H. Extending 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.

I. 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.

2. This version of `main()` should be handed in.

J. Extra credit: A revised "encrypt"

The `encrypt` that you wrote for Part B 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.

K. Marking rubric

This assignment is worth 10 points. The breakdown is as follows. To get full credit you will have to complete all the parts from A to I (in other words the full credit for classes like `message` require that you write both the pices necessary to encrypt and decrypt).

- A `message` class with two constructors, two data members and five function members.
  (2 points)

- A `caesar` class with one data member, two constructors, and three function members.
  (2 points)
- A `fileHandleChar` class with one data member, a constructor, a destructor and one function member.
  (1 point)
- `fileHandleString` class with one data member, a constructor, a destructor and one function members.
  (1 point)
- `encryption` class, with five data members and three function members.
  (2 points)
- `main()` which gives the user the option of encrypting or decrypting a string and either writes the encrypted text to a file, or reads the message to decrypt from a file.
  (2 points)
- An extended `encrypt` method that produces a cipher text in four character blocks, separated by spaces, and all in capitals.
  (1 extra credit point)