COMP 2012 Object-Oriented Programming and Data Structures

Lab 6: Abstract Base Class, Static Data Members/Member Functions and Friend

Image by storyset on Freepik

Overview

In this lab, you are going to implement a text encryption/decryption system to demonstrate your understanding of static data members, static member functions, and friend mechanisms in C++.

You can download the skeleton code in a zip package HERE.

Notes on Caeser Cipher

Caesar Cipher is a simple encryption technique that involves shifting each letter of a message by a fixed number of positions down the alphabet. It is named after Julius Caesar, who is said to have used a similar encryption technique in his private correspondence.

The Caesar Cipher takes a message and shifts each letter by a fixed number of positions, known as the "key". For example, if the key is 3, each letter in the message would be shifted three positions down the alphabet. So, the letter `a` would become `d`, `b` would become `e`, and so on.

To decrypt the message, the receiver would simply shift each letter back by the same number of positions. For example, if the key was 3, the receiver would shift each letter three positions up the alphabet to reveal the original message.

When a lowercase letter is used as a key for Caesar Cipher, we have the following pairs of equivalent keys:

- key = `a` ≡ key = 0
- key = `b` ≡ key = 1

- ...
- key = `y` ≡ key = 24
- key = `z` ≡ key = 25

Here are some examples of Caesar Cipher.

- key = `d` (or, key = 3)
  - plaintext: `hello`
  - ciphertext: `khoor`
  - In this example, each letter in the plaintext message has been shifted three positions down the alphabet to create the ciphertext message.
- key = `n` (or, key = 13)
  - plaintext: `University of Science and Technology`
  - ciphertext: `havirefvgl bs fpvrapr naq grpuabybtl`
  - In this example, each letter in the plaintext message has been shifted thirteen positions down the alphabet to create the ciphertext message. All uppercase letters in the plaintext are converted to lowercase letters before the encryption process. Note also that spaces are included in the encryption process but are not shifted.
- key = `f` (or, key = 5)
  - plaintext: `COMP 2012 is the best COMP course.`
  - ciphertext: `htru 2012 nx ymj gjxy htru htzwxj.`
  - In this example, each letter in the plaintext message has been shifted five positions down the alphabet to create the ciphertext message. All uppercase letters in the plaintext are converted to lowercase letters before the encryption process. Note also that numbers, spaces and punctuation marks are included in the encryption process but are not shifted.

Notes on Vigenere Cipher

Vigenere Cipher is a polyalphabetic substitution cipher that was invented by the French cryptographer Blaise de Vigenère in the 16th century. It is a more advanced encryption technique than the simple Caesar Cipher, which uses a series of Caesar ciphers with different keys to provide stronger encryption.

The Vigenere Cipher uses a key to generate a series of Caesar ciphers. Each letter in the plaintext message is then shifted by a different amount, depending on its relative position in the key. For example, if the key is `computer`, the first letter of the plaintext would be shifted by 2 positions (`c` is the 3rd letter of the alphabet), the second letter by 14 positions (`o` is the 15th letter), the third by 12 positions (`m` is the 13th letter), and so on.

A valid key for Vigenere Cipher is represented as a string of alphabet letters. All uppercase letters in the key are treated the same as lowercase letters. For example, the key `ComPuteR` is equivalent to `computer`.

Vigenere Cipher uses a key that is repeated throughout the message. This means that the same key is used to encrypt each letter in the message, but the shift for each letter depends on its relative position in the key. This makes the Vigenere Cipher much stronger than the simple Caesar Cipher, as it provides a different shift for each letter in the message.

To decrypt the message, the recipient must know the key used to generate the cipher. They can then apply the opposite shifts to each letter in the ciphertext to reveal the original plaintext message.

Here are some examples of Vigenere Cipher.

- key = computer
  - plaintext: `hello`
  - ciphertext: `jsxai`
  - In this example, the key `computer` is used to generate a series of Caesar ciphers, with each letter in the plaintext message shifted by a different amount based on its relative position in the key.
- key = computer
  - plaintext: `University of Science and Technology`
  - ciphertext: `wbukykwzvm dz wtkszry eef ftwarfncsn`
  - In this example, the key `computer` is used to generate a series of Caesar ciphers, with each letter in the plaintext message shifted by a different amount based on its relative position in the key.
- key = apple
  - plaintext: `University of Science and Technology`
  - ciphertext: `ucxgirhxec du wcxtyge pyh itnlndazky`
  - In this example, the key `apple` is used to generate a series of Caesar ciphers, with each letter in the plaintext message shifted by a different amount based on its relative position in the key.

- key = apple
  - plaintext: `COMP 2012 is the best COMP course.`
  - ciphertext: `cdba 2012 ih ele qpwt rzqp rzyrht.`
  - In this example, the key `apple` is used to generate a series of Caesar ciphers, with each letter in the plaintext message shifted by a different amount based on its relative position in the key.

Description of Main Classes

In this lab, you will be working with three classes: `Cipher`, `VigenereCipher`, and `CipherUtility`.

Cipher

```cpp
// Cipher is an abstract base class
class Cipher
{
  // TODO #7
  // Make CipherUtility as a friend of Cipher so that CipherUtility can access
  // all the data members, including those are private

  protected:
    string key;

  public:
    // Conversion constructor. Use MIL to resolve the name conflict
    Cipher(string key) : key(key) {}

    // Make the default destructor virtual
    virtual ~Cipher() = default;

    // Define two pure virtual functions

    virtual string encrypt(string plaintextMsg) const = 0;
    virtual string decrypt(string ciphertextMsg) const = 0;

    // Print the data member
    void print() const { cout << key; }
};
```

The definition of this class can be found in **Cipher.h**. The Cipher class is an abstract base class that contains two pure virtual functions, `encrypt` and `decrypt`. You are required to add one line of code to make `CipherUtility` as a friend of `Cipher` so that `CipherUtility` can access all the data members, including those are private in **Cipher.h**

VigenereCipher

```cpp
// VigenereCipher is a derived class of Cipher
class VigenereCipher : public Cipher
{
  private:
    // A private static constant variable
    // Need to be initialized in VigenereCipher.cpp
    // => TODO #1: Initialize the static data member alphabetSize in VigenereCipher.cpp
    static const int alphabetSize;

  public:
    // [IMPLEMENTED] Conversion constructor
    // It calls the conversion constructor of the base class to initialize key
    VigenereCipher(string key) : Cipher(key) {}

    // [TO BE IMPLEMENTED] Override the two pure virtual functions inherited from Cipher class
    // => TODO #2: Implement the encrypt member function in VigenereCipher.cpp
    // => TODO #3: Implement the decrypt member function in VigenereCipher.cpp
    virtual string encrypt(string plaintextMsg) const override;
    virtual string decrypt(string ciphertextMsg) const override;

    // [TO BE IMPLEMENTED] A static member function, which generates a random key with length
    // specified by the parameter
    // => TODO #4: Implement the static member function generateRandomKey in VigenereCipher.cpp
    static string generateRandomKey(int length);
};
```

The definition of this class can be found in **VigenereCipher.h**. The VigenereCipher class is a derived class of Cipher that contains a static constant data member, `alphabetSize`. It also overrides the two pure virtual functions inherited from the Cipher class to implement a Vigenere cipher. In addition to the member functions inherited from the Cipher class, the VigenereCipher class also contains a static member function, `generateRandomKey`, which generates a random keyword of a specified length.

Implementation of encrypt function

You are required to implement the `encrypt` function according to the Vigenere Cipher encryption algorithm, which is a method of encrypting plaintext using a series of interwoven Caesar ciphers. The algorithm takes in a plaintext message as input and returns a ciphertext message as output.

A valid key for Vigenere Cipher is represented as a string of alphabet letters. All uppercase letters in the key are treated the same as lowercase letters. For example, an `E` in the key string would be treated as an `e` which indicates a shift of 4 positions.

The Vigenere Cipher encryption algorithm starts by initializing an empty string to hold the encrypted message and an index variable to keep track of which character in the key is currently being used. Then, for each character in the plaintext message, the algorithm checks if it is an alphabet letter. If it is, the algorithm converts it to lowercase, determines the amount of shift by taking the corresponding letter in the key, wrapping around to the beginning of the key if necessary, and subtracting the ASCII value of 'a'. It then applies the shift to the current character, wrapping around to the beginning of the alphabet if necessary, and adds the ASCII value of 'a' to convert it back to a character in the alphabet. Finally, the encrypted character is appended to the ciphertext message.

The algorithm repeats this process for every character in the plaintext message, interweaving the different Caesar ciphers based on the key. Once all the characters have been encrypted, the algorithm returns the ciphertext message.

Implementation of decrypt function

You are required to implement the `decrypt` function according to the Vigenere Cipher decryption algorithm. It requires knowledge of the secret key that was used to encrypt the message.

The decryption algorithm works by reversing the steps of the encryption algorithm. It starts by initializing an empty string to hold the decrypted message and an index variable to keep track of which character in the key is currently being used. Then, for each character in the ciphertext message, the algorithm checks if it is an alphabet letter. If it is, the algorithm converts it to lowercase, determines the amount of shift by taking the corresponding letter in the key, wrapping around to the beginning of the key if necessary, and subtracting the ASCII value of 'a'. It then applies the reverse shift to the current character, wrapping around to the end of the alphabet if necessary, and adds the ASCII value of 'a' to convert it back to a character in the alphabet. Finally, the decrypted character is appended to the decrypted message.

The algorithm repeats this process for every character in the ciphertext message, interweaving the different Caesar ciphers based on the key. Once all the characters have been decrypted, the algorithm returns the decrypted message.

Implementation of generateRandomKey function

You are required to implement the `generateRandomKey` function to generate a random key of a specified `length` for use in the Vigenere Cipher encryption algorithm.

The function takes in an integer value `length` that specifies the desired length of the random key. The algorithm initializes an empty string key to hold the key. It then loops `length` times, generating a random character for each position in the key. For each iteration of the loop, the algorithm generates a random number between 0 and `alphabetSize` (the number of lowercase characters in the alphabet), adds this number to the ASCII value of the character 'a', and converts the resulting ASCII value to a character. This character is then added to the key string.

In this lab, you **MUST** use the pseudo-random generator `_rand()`, as implemented in **Utility.h**, to generate a random number (so that all the programs submitted to ZINC can be assessed consistently with respect to the expected outputs). Specifically, you can use the following code to generate a random number between 0 and `alphabetSize` (the number of lowercase characters in the alphabet):

```
_rand() % alphabetSize
```

Once the loop has been completed, the algorithm returns the generated key as a string. The resulting key will be a random combination of lowercase characters from the alphabet, with a length equal to the value of length. This key can then be used in the Vigenere Cipher encryption algorithm to encrypt a plaintext message.

CipherUtility

```
class CipherUtility
{
  public:
    // [TO BE IMPLEMENTED] A static member function, which validates the key of the given cipher
    // => TODO #5: Implement the static member function isKeyValid in CipherUtility.cpp
    static bool isKeyValid(const Cipher* cipher);

    // [TO BE IMPLEMENTED] A recursive static member function, which recursively remove all
    // the non-alphabet characters in the string, str
    // => TODO #6: Implement the static member function removeNonAlphaCharsHelper in CipherUtility.cpp
    static string removeNonAlphaCharsHelper(string str);

    // [IMPLEMENTED] A static member function, which calls removeNonAlphaCharsHelper
    // to remove all the non-alphabet characters in the key of the given cipher
    static void removeNonAlphaChars(Cipher* cipher) {
      cipher->key = removeNonAlphaCharsHelper(cipher->key);
    }
};
```

The definition of this class can be found in **CipherUtility.h**. The CipherUtility class is a utility class that contains two static member functions, `isKeyValid` and `removeNonAlphaCharsHelper`, which are used to validate the keyword and remove non-alphabetic characters from a key, respectively.

Implementation of isKeyValid function

You are required to implement the `isKeyValid` function to check whether a key used in a cipher is valid.

The function takes a pointer to a Cipher object as input and returns a boolean value indicating whether the key is valid or not. The function should loop through each character in the key string of the Cipher object. For each character, it checks whether it is an alphabet letter by using the `isalpha` function. If the character is not an alphabet letter, the function returns `false` to indicate that

Lab Tasks

You have to complete the following TODO tasks so that the program can produce the correct results as expected. In particular, you are required to add appropriate lines of code in 7 different places in **VigenereCipher.cpp**, **CipherUtility.cpp**, and **Cipher.h**.

TODO #1
Initialize the static data member `alphabetSize` in **VigenereCipher.cpp**

TODO #2
Implement the `encrypt` member function in **VigenereCipher.cpp**

TODO #3
Implement the `decrypt` member function in **VigenereCipher.cpp**

TODO #4
Implement the static member function `generateRandomKey` in **VigenereCipher.cpp**

TODO #5
Implement the static member function `isKeyValid` in **CipherUtility.cpp**

TODO #6
Implement the static member function `removeNonAlphaCharsHelper` in **CipherUtility.cpp**

TODO #7
Make `CipherUtility` as a friend of `Cipher` so that `CipherUtility` can access all the data members, including those are private in **Cipher.h**

Sample Sessions

A minimal `Makefile` for Windows platform is included in the zip package of the skeleton code as a reference. You might need to make appropriate modifications for it to work on other platforms (e.g. Linux workstations in CS Lab 2).

After you have properly completed the TODO tasks (**TODO #1 - TODO #7**), you can compile the files by typing `make all` to produce 2 executable files: `lab6_test1.exe` and `lab6_test2.exe`.

The expected output after executing `./lab6_test1.exe` is shown below.

```
Generate a random key: [vobwzqlk]

Encrypt the message: "COMP 2012 is the best COMP course."
Cipher Text: [xcnl 2012 jo jso pfos nyhd ynkccz.]

Decrypt the message: "xcnl 2012 jo jso pfos nyhd ynkccz."
Plain Text: [comp 2012 is the best comp course.]

The key "my secret key :)" is [invalid]

Output the new key: [mysecretkey]

Encrypt the message: "COMP 2012 is the best COMP course."
Cipher Text: [omet 2012 ge llg fxcx omet tsnbwc.]

Decrypt the message: "omet 2012 ge llg fxcx omet tsnbwc."
Plain Text: [comp 2012 is the best comp course.]
```

The expected output after executing `./lab6_test2.exe` is shown below.

```
Generate a random key: [vobwzqlklkru]

Encrypt the message: "Desmond and Cecia are the COMP2012 course instructors."
Cipher Text: [ystindo lxu xsdez lbp kbz dklf2012 wjisod txddioxhpnr.]

Decrypt the message: "ystindo lxu xsdez lbp kbz dklf2012 wjisod txddioxhpnr."
Plain Text: [desmond and cecia are the comp2012 course instructors.]
```

Submission & Deadline

The deadline of submission for all lab sections is **20:00:00 on 21 April 2023 (Fri)**.

Please submit your completed work to ZINC by zipping the following 3 files.

```
VigenereCipher.cpp
CipherUtility.cpp
Cipher.h
```

ZINC usage instructions can be found here. You can make multiple submissions to ZINC before the deadline. Only the last submission received before the deadline will be graded.

Menu

Page maintained by

Namkiu Chan
Last Modified: 04/17/2023 15:44:47

Homepage

Course Homepage