LulzBank™ Security

CSC 122: Data Structures

Due: Monday 2/17 at 8:00 AM, via Moodle

I recommend you try to finish this by Wednesday 2/12 or Thursday 2/13. That will give you plenty of time to ask me questions on Friday 2/14, and/or ask Erica questions on Sunday 2/16. Please don't save it for the last weekend! That probably won't result in a positive experience.

Welcome to LulzBank™! *Come for the lulz, stay for the bank*. We are very pleased that you have joined our team. As you know, LulzBank™ is a new player in the global finance industry. At LulzBank™, we take our customer's security[1] very seriously. We also really like to say LulzBank™.

As a new employee at LulzBank™, you'll start with some on-the-job training. Once you've completed that, you'll join our cybersecurity team. So you'll need to make two completely separate Visual Studio projects, which you'll then make one zip out of to turn in. To make one zip containing two projects, just get to the folder that contains the main folders for both of your projects. Single-click on one project folder. Then, while holding ctrl, single-click on the other. Then right-click and choose send to --> compressed (zipped) folder, and continue as before. You can verify that the zip contains both of your projects.

Also check hw0 for a review on making projects and such if you'd like. We'd bet your supervisor, Steve Bogaerts, could help you with that if you have trouble. Only one of the people in your pair should submit your work to Moodle, so please determine who it will be beforehand.

We've hired you and your partner to write some important C++ code to help us get off the ground. While working as a pair, it is ***essential*** that you do all the work together, sitting next to each other, working on the same computer and talking together to figure out the solution. If there's ever a time when only one person is working on this project (while the other is busy with something else, elsewhere), then only that one person will have gained maximum benefit from working on that part – *even if he/she explains it to the other person carefully later*. The other person won't learn the material nearly as well, which can set up a chain of negative consequences. So please only work together on the project, not separately.

On-the-Job Training: hw2, Part 1

Before you start with our cybersecurity team, we need you to get a little training. Make a Visual Studio project called *lastname1-lastname2*-hw2-1. Please practice working with classes in C++ by doing the following:

- Define a Student class that has a name and GPA fields.
- Define a constructor that takes a name and GPA as arguments.

---

[1] That's not a typo. I do mean customer's, not customers', because right now we only have one customer.

- Define an accessor for the name field called getName.
- Define an accessor and mutator for the GPA field called getGPA and setGPA, respectively.
    - If you don't recall what an accessor or mutator is, check with Supervisor Steve!
- Finally, define a method called printInfo that does some couts indicating the student's information. (See example below.)

Define a header file (.h) and definition file (.cpp) for the class, of course.

Also define a main.cpp in which you paste the following main:

```
#include <iostream>
#include "Student.h"

using namespace std;

int main() {
	Student s("Lemuel", 3.2);

	cout << s.getName() << endl;
	cout << s.getGPA() << endl;

	cout << "Changing gpa..." << endl;
	s.setGPA(3.6);

	s.printInfo();

	return 0;
}
```

Write your class so that when you run the main, you get the following output:

```
Lemuel
3.2
Changing gpa...
Lemuel is a student with GPA: 3.6.
```

Notes:

- Please do not change the main! If you feel you have to change the main to make your code work, that is a sign that your code is incorrect in some way. Changes to the main above will likely result in lost points.
- Don't forget to `#include <string>` and say `using namespace std;` when you want to use strings! (Even in your .h file.) Otherwise you can get some confusing error messages.
- Also, don't forget the semicolon at the end of the class declaration:
  `class Student {`
  ... class declaration here
  `};` <--- Don't forget this semicolon!

On-the-Job Training: String Concatenation

For your next step, please look at the provided StringConcatenationDemo project. You'll see a simple function in main.cpp called makeAcronym that asks the user to type 5 words. The first letter of each

word is put together ("concatenated") to make an "acronym", which is returned. Please run the code and explore it a bit to see how it works. Basically, the + operator, when applied to strings, merges the two strings together into one new string. This is how the acronym is built up one letter at a time.

You'll need to understand string concatenation in C++ to be able to tackle your next task.

LulzBank Security: hw2, Part 2

So you've completed your training and you're ready to really make a difference in our company? Excellent. Please make a new Visual Studio project called *lastname1-lastname2*-hw2-2, but read the rest of this document in its entirety before you start any additional coding.

A Substitution Cipher: Summary

A *cipher* is an encoding of characters in a message in order to hide their meaning. A *substitution cipher* is a particular kind of cipher that encodes characters via a simple substitution. That is, in encoding, each letter is swapped with a different one according to some cipher key. For decoding, the reverse is done. For example, a cipher key of "qwertyuiopasdfghjklzxcvbnm" means that all a's are replaced with q's, all b's with w's, etc. To help you visualize, it is convenient to line things up like this:

```
abcdefghijklmnopqrstuvwxyz
qwertyuiopasdfghjklzxcvbnm
```

So, for example, "hello" would be encoded as "itssg". We would call "hello" the *plaintext* and "itssg" the *ciphertext*.

Your task is to write a class called `SubstCipher` that performs substitution ciphers. You should write the class such that the following main() function:

```
#include <iostream>
#include <string>
#include "SubstCipher.h"
using namespace std;

int main() {
                        // abcdefghijklmnopqrstuvwxyz
    SubstCipher cipher("qwertyuiopasdfghjklzxcvbnm");

    cout << "==================================================" << endl;
    cout << "Using keys: " << cipher.getCipherKeys() << endl;
    cout << endl;

    cout << "Encode 'a': " << cipher.encodeChar('a') << endl;
    cout << "Decode 'q': " << cipher.decodeChar('q') << endl;
    cout << endl;

    cout << "Encode \"hello\" : " << cipher.encodeString("hello") << endl;
    cout << "Decode \"itssg\" : " << cipher.decodeString("itssg") << endl;
    cout << endl;
```

```
    cout << "=====================================================" << endl;
    cipher.setCipherKeys("wrong");
    cout << endl;

    cout << "=====================================================" << endl;
                        // abcdefghijklmnopqrstuvwxyz
    cipher.setCipherKeys("laksjdhfgqpwoeirutyzmxncbv");
    cout << "Using keys: " << cipher.getCipherKeys() << endl;

    cout << "Encode 's': " << cipher.encodeChar('s') << endl;
    cout << "Decode 'q': " << cipher.decodeChar('q') << endl;
    cout << endl;

    cout << "Encode \"sup\" : " << cipher.encodeString("sup") << endl;
    cout << "Decode \"fjwwi\" : " << cipher.decodeString("fjwwi") << endl;
    cout << endl;

    return 0;
}
```

gives the following results:

```
=====================================================
Using keys: qwertyuiopasdfghjklzxcvbnm

Encode 'a': q
Decode 'q': a

Encode "hello" : itssg
Decode "itssg" : hello

=====================================================
Cipher keys string must have length 26. Request ignored.

=====================================================
Using keys: laksjdhfgqpwoeirutyzmxncbv
Encode 's': y
Decode 'q': j

Encode "sup" : ymr
Decode "fjwwi" : hello
```

Note in the main above that I've included in comments the alphabet a...z lined up above the cipherKeys string, so that you can easily see what should encode/decode to what. Study the code in the main and compare it to the output so that you understand what is happening – what the class is supposed to do.

More specifically, the SubstCipher class should have the following:

- A private field called cipherKeys of type string.
- A public constructor that takes a string argument and initializes the cipherKeys field to that argument.
- A public accessor and mutator for the cipherKeys field.
- A public method called encodeChar. It takes a char representing a single plaintext character and returns the corresponding ciphertext character, according to the cipherKeys field.
- A public method called decodeChar. It takes a char representing a single ciphertext character and returns the corresponding plaintext character, according to the cipherKeys field.
- A public method called encodeString. It works the same as encodeChar, except that it takes an entire string and returns an entire string.

- A public method called decodeString that again works like decodeChar, except for strings.
- Your class should also have two additional *private* methods, explained below.

Some Code to Get You Started

I have pasted the implementation of the private methods here:

```
int SubstCipher::charToAlphabetPosition(char c) {
      return static_cast<int>(c) - static_cast<int>('a');
}

char SubstCipher::alphabetPositionToChar(int pos) {
      return static_cast<char>(pos + static_cast<int>('a'));
}
```

This uses a concept we haven't talked about at this point. It's not a big deal, but basically charToAlphabetPosition takes a character, and tells you what position it is in the alphabet (starting with 0). For example, if you pass 'b' as an argument, it will return 1. If you pass 'z' as an argument, it will return 25.

Similarly, alphabetPositionToChar takes an alphabet position and returns the corresponding letter. For example, if you pass 25 as an argument, it will return 'z'. If you pass 3 as an argument, it will return 'd'.

These methods should be defined as *private* because they'll be useful to you in defining your other methods, but they aren't intended for use outside of the class.

These methods will be used in the encodeChar and decodeChar methods, which I'm also pasting below:

```
char SubstCipher::encodeChar(char plain) {
      return cipherKeys[charToAlphabetPosition(plain)];
}

char SubstCipher::decodeChar(char cipher) {
      return alphabetPositionToChar(cipherKeys.find(cipher));
}
```

Ponder for a bit how they work. encodeChar takes a plaintext character. It finds the position in the alphabet where the plaintext character is, using charToAlphabetPosition. That is then the index that is used to access the cipherKeys string.

So for example, if you pass 'd' to encodeChar, then we call charToAlphabetPosition('d'), which will return 3. So we return the character at index 3 in the cipherKeys field.

decodeChar is the same basic idea in reverse. It takes a ciphertext character, searches for that character using the `find` method defined in the string class, and returns the character at the corresponding position in the normal alphabet. For example, suppose we have the cipherKeys string below, again written with the normal alphabet above it so you can see the correspondence:

```
abcdefghijklmnopqrstuvwxyz
qwertyuiopasdfghjklzxcvbnm
```

Suppose we then pass 't' to decodeChar. The `find` method, defined in the string class, will return where the 't' occurs in the cipherKeys string: index 4 in this case. We then call alphabetPositionToChar to determine the letter at index 4 in the normal alphabet, which is 'e'. So 't' decodes to 'e'.

Defining the Rest of the Class

So your task is to write the rest of the SubstCipher class. You'll need to write the declaration in a SubstCipher.h file as usual. Make sure you make the charToAlphabetPosition and alphabetPositionToChar methods private, as well as the cipherKeys field.

Also create a SubstCipher.cpp file in which you'll define the methods for the class. To start with, paste the methods given above into that file. Then define:

- A constructor
  - This should take a string argument and initialize the cipherKeys field accordingly.
- getCipherKeys
  - This is a standard accessor for the cipherKeys field.
- setCipherKeys
  - This is a standard mutator for the cipherKeys field, except for one additional thing: it should double check that the length of the provided string is 26, as would be required for a valid value of cipherKeys. (See the main for example behavior.)
- encodeString
  - This takes a string representing plaintext. It should run through the string one character at a time, calling encodeChar on each character, and concatenating each result onto a string that is returned.
  - It is important that you call encodeChar from encodeString – don't simply copy encodeChar's code into encodeString! This is a better program design. To receive credit for this part, you must call encodeChar from encodeString and use it appropriately.
- decodeString
  - This behaves the same as encodeString, except it takes a string representing ciphertext, and calls decodeChar on each character. Again, to receive credit for this part, you must call decodeChar from encodeString and use it appropriately.

Double check the main and the sample output for examples of how these methods work.

Finally, make a main.cpp file in which you paste the provided main, so that you can test your work. Do not change the main at all – your code should work as expected for the given main. Also note that when you paste the main from this file, the indentation might not be saved. That's ok – you can easily highlight the entire body of the main in Visual Studio and hit tab, and it will all get tabbed over for you.

Some Final Notes

- You may assume that you're dealing only with lower-case letters: no upper-case, no other symbols, etc.

- You may get a warning about a signed/unsigned mismatch in a for loop. That's ok. You may safely ignore the warning, or you could make your loop variable be of type `unsigned int` if you want.