Contents - Overloaded Operators lab

Part 1

Setup

1.1 Turn-in instructions

- All .hpp/.cpp files you've modified for this project.

1.2 Setup

The following files are included in this program:

```
OverloadedOperators_Lab/
├── main.cpp
├── Program.hpp and Program.cpp
├── CipherText.hpp and CipherText.cpp
│   └── utilities/
│       └── Menu.hpp and Menu.cpp
└── Project_CodeBlocks/
    ├── (cbp project file and other files)
    └── (etc)
```

1.3 The program

```
-------------------------------------------
 | CIPHER TEXT WIDGET |
 ----------------------

m_cipherTexts[0]: Example text... 1 2 3 (0)
m_cipherTexts[1]: J}fruqj%yj}y333%6%7%8 (5)


OPERATIONS MENU
 1.  QUIT
 2.  + operator
 3.  - operator
 4.  Display
 5.  Get offset
 6.  Get text
 7.  Input with >>
 8.  Offset with []
 9.  Output with <<
 10. Set text
 11. a == b?

 >>
```

In this program, there is a CipherText class that stores a string and an offset. This offset shifts the letters stored in the string over, creating a simple (and honestly not very secure) cipher.

The CipherText object contains several overloaded operators used to modify its string and offset. You will be filling out those functions for this assignment.

The Program.hpp/cpp and main.cpp files don't need to be modified at all.

1.4 The lab

CipherText
+ CipherText() + CipherText() + CipherText(const CipherText& other) + CipherText(const CipherText* other) + CipherText& operator=(const CipherText& other) + void SetText(string text) + string GetText() const + int GetOffset() const + void Display() const + CipherText operator[ ] (const int index) friend CipherText operator+(const CipherText& item, int amount) friend CipherText operator-(const CipherText& item, int amount) friend bool operator==(const CipherText& a, const CipherText& b) friend ostream& operator<<(ostream& out, CipherText& item) friend istream& operator>>(istream& in, CipherText& item)
- int m_offset - string m_text - void OffsetText(int amount) - CipherText RemoveOffset()

And as a bonus, the **copy constructors** are already written for you!

```
1  CipherText::CipherText( const CipherText& other )
2  {
3      m_offset = other.m_offset;
4      m_text = other.m_text;
5  }
6
7  CipherText::CipherText( const CipherText* other )
8  {
9      m_offset = other->m_offset;
10     m_text = other->m_text;
11 }
```

1.4.1 The Assignment Operator (operator=)

```
1 CipherText& CipherText::operator=( const CipherText&
     other )
2 {
3     if ( this == &other ) return *this;
4
5     // Basically do the copy contructor stuff here
6
7     return *this;
8 }
```

The assignment operator is very similar to the copy constructor, except we have to add an error check: We have to make sure we're not assigning an object to itself.

At the beginning of the function, make sure to check if the `other` object has the same memory address as `this`. If the addresses are the same, then just de-reference `this` as a return object. (All of this is Line 3 in that example up there.)

At the end of the function, we also return a de-referenced `this` - the pointer to the class instance we're currently working with.

In the middle copy over the member variables from `other` to the local instance, just like in the copy constructors.

1.4.2 The Addition Operator (operator+)

```
1 CipherText operator+( const CipherText& item, int amount
      )
2 {
3 }
```

For this function you will need to make a CipherText copy to work with. Then, you'll do the "plus" operation on that copy, and return the copy as the updated information. We do this because we don't necessarily want to update the original CipherText when we use the + operator, like if we did this:

```
CipherText newText = oldText + 2;
```

The good news is, I've already implemented a function called **OffsetText** that you can call do to the cipher offset. All you need to do is this:

1. Create a variable of type `CipherText` called `changed`, and assign it `item` (this makes a copy via the assignment operator).
2. Call `changed.OffsetText( amount );`, passing the amount passed in as a parameter to the offset function.
3. Return `changed`.

1.4.3 The Subtraction Operator (operator-)

```
1 CipherText operator-( const CipherText& item, int amount
      )
2 {
3 }
```

This is just like the + operator, except when you pass in `-amount` when you call OffsetText...

```
changed.OffsetText( -amount );
```

1.4.4 The Output Stream Operator (operator<<)

```
1 ostream& operator<<( ostream& out, CipherText& item )
2 {
3 }
```

The output stream operator is used so that we can output this object to text files and to the screen with `cout`. We just have to decide *what* we want to output.

For this function, output `item.m_text` only; we don't want to output `m_offset` as part of the output stream.

1. Stream `item.m_text` to the `out` ostream object using the stream operator `<<`.
   (`out << item.m_text;`)
2. Return `out`.

1.4.5 The Input Stream Operator (operator>>)

```
1 istream& operator>>( istream& in, CipherText& item )
2 {
3 }
```

Similarly, we can tell C++ *how* to read data from the keyboard or a text file using the stream operator `>>` by overloading this operator.

In this case, when we load in something via `cin >>` we will only want to read in contents for `m_text`, and then reset `m_offset` to 0.

1. Stream from `in` to `item.m_text` using the stream operator `>>`.
   (`in >> item.m_text;`)
2. Set the `item`'s `m_offset` to 0.
3. Return `in`.

1.4.6 The Subscript Operator (operator[ ])

```
1 CipherText CipherText::operator[] ( const int index )
2 {
3     CipherText temp( this );
4     temp.OffsetText( index );
5     return temp;
6 }
```

While we can use this operator with a C++ string to get a certain letter in a string, we're going to implement this differently. We're going to use this to set an offset without using the + and - operators.

First, we will have to undo any current offset there is, and then apply the new offset.

1. Create a new `CipherText` variable named `temp`, passing in `this` as an argument to its constructor.
   (`CipherText temp( this );`)
2. Call the RemoveOffset() function on `temp`.
3. Call the OffsetText() function on `temp`, passing in the `index` parameter.
4. Return `temp`.

1.4.7 The Equivalent Operator (operator==)

```
1 bool operator==( const CipherText& a, const CipherText&
     b )
2 {
3 }
```

This operator is meant to check if two things are equivalent to each other. We aren't going to just compare the two items' `m_text` strings together because one or both could be "encrypted". Instead, we're going to "de-encrypt" both strings to compare and return true or false.

1. Create a new `CipherText` variable named `unencryptA`, passing in `a` as an argument to its constructor.
2. Create a new `CipherText` variable named `unencryptB`, passing in `b` as an argument to its constructor.
3. Call the RemoveOffset() function on `unencryptA`, store it back in `unencryptA`.

   (`unencryptA = unencryptA.RemoveOffset();`)
4. Call the RemoveOffset() function on `unencryptB`, store it back in `unencryptB`.
5. If `unencryptA`'s `m_text` is equivalent to `unencryptB`'s `m_text`, then return true. Otherwise, return false.

1.5 Example output

Program start:

```
-------------------------------------------
 | CIPHER TEXT WIDGET |
 ----------------------

m_cipherTexts[0]: Example text... 1 2 3 (0)
m_cipherTexts[1]: Example text... 1 2 3 (0)

OPERATIONS MENU
 1.  QUIT
 2.  + operator
 3.  - operator
 4.  Display
 5.  Get offset
 6.  Get text
 7.  Input with >>
 8.  Offset with []
 9.  Output with <<
 10. Set text
 11. a == b?

 >>
```

+ operator:

```
----------------------------------------
 | + Operator |
 --------------

m_cipherTexts[0]: Example text... 1 2 3 (0)
m_cipherTexts[1]: Example text... 1 2 3 (0)


----------------------------------------
 Enter which CipherText to modify (0, 1)

 >> 0

 Enter a number amount to add to the offset.

 >> 1


The CipherText string is now: Fybnqmf!ufyu///!2!3!4

 Press ENTER to continue...
```

Afterward the main menu will show this:

```
m_cipherTexts[0]: Fybnqmf!ufyu///!2!3!4 (1)
m_cipherTexts[1]: Example text... 1 2 3 (0)
```

The number in the parentheses is the string's offset.

- operator:

```
----------------------------------------
 | - Operator |
 --------------

m_cipherTexts[0]: Fybnqmf!ufyu///!2!3!4 (1)
m_cipherTexts[1]: Example text... 1 2 3 (0)


----------------------------------------
 Enter which CipherText to modify (0, 1)

 >> 1

 Enter a number amount to subtract the offset.

 >> 1


The CipherText string is now: Dw‘lokdsdws---012

 Press ENTER to continue...
```

Remember to type a positive number. If you type something like -1, then it will subtract -1, resulting in adding 1 instead.

Afterward the main menu will show this:

```
m_cipherTexts[0]: Fybnqmf!ufyu///!2!3!4 (1)
m_cipherTexts[1]: Dw‘lokdsdws---012 (-1)
```

a == b?:

```
-----------------------------------------
 | == Operator |
 ---------------

m_cipherTexts[0]: Fybnqmf!uf (1)
m_cipherTexts[1]: Dw‘l (-1)

Are m_cipherTexts[0] and m_cipherTexts[1] equvalent?
They are NOT equivalent.

 Press ENTER to continue...
```

If your strings are the same when unencrypted, even if they're encrypted they will show as equivalent.

operator ¡¡:

```
-----------------------------------------
 | << Operator |
 ---------------

m_cipherTexts[0]: apple (0)
m_cipherTexts[1]: Example text... 1 2 3 (0)

m_cipherTexts[0] is apple
m_cipherTexts[1] is Example text... 1 2 3

 Press ENTER to continue...
```

asdfasdf:

```
-----------------------------------------
 | [] Operator |
 ---------------

m_cipherTexts[0]: apple (0)
m_cipherTexts[1]: Example text... 1 2 3 (0)


-----------------------------------------
 Enter which CipherText to modify (0, 1)

 >> 0

 Enter an offset amount to view (positive or negative is
    fine).

 >> 10

kzzvo

Do you want to save this string in m_cipherTexts[0]?
 1. yes
 2. no

 >> 1


 Press ENTER to continue...
```