B3 - C++ Pool

B-CPP-300

Day 17

Algorithms

1.25

{EPITECH.}

Day 17

binary name:	no binary
group size:	1
repository name:	cpp_d17
repository rights:	ramassage-tek
language:	C++

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).

All your exercises will be compiled with `g++` **and the** `-W -Wall -Wextra -Werror` **flags**, unless specified otherwise.

All output goes to the standard output, and must be ended by a newline, unless specified otherwise.

None of your files must contain a `main` function, unless specified otherwise.
We will use our own `main` functions to compile and test your code. It will include your header files.

For each exercise, the files must be turned-in in a separate directory called **exXX** where XX is the exercise number (for instance `ex01`), unless specified otherwise.

Read the examples CAREFULLY. They might require things that weren't mentioned in the subject...

If you do half the exercises because you have comprehension problems, it's okay, it happens. But if you do half the exercices because you're lazy, and leave at 2PM, you **WILL** have problems.
Do not tempt the devil.

The `*alloc`, `free`, `*printf`, `open` and `fopen` functions, as well as the `using namespace` keyword, are forbidden in C++.
By the way, `friend` is forbidden too, as well as any library except the standard one.

Unit Tests

It is highly recommended to test your functions as you implement them. It is common practice to create and use what are called **unit tests**.

From now on, we expect you to write unit tests for your functions (when possible). To do so, please follow the instructions in the **"How to write Unit Tests"** document on the intranet, available here.

Create a directory named `tests`. For each of the functions you turn in, create a file in that directory named `tests-FUNCTION_NAME.c` containing all the tests needed to cover all of the exercise's possible cases (regular or irregular).

Here is a sample set of unit tests for the **string** class:

```
#include <criterion/criterion.h>

Test(string, default_value)
{
        std::string s;

        cr_assert_eq(s, "");
}

Test(string, assign)
{
        std::string s;

        s = "test";
        cr_assert_eq(s, "test");
}

Test(string, append)
{
        std::string s("test");

        s += "ing";
        cr_assert_eq(s, "testing");
}
```

Exercise 0 - Find me that

Turn in: `find.hpp`

You must create a `do_find` function template taking 2 parameters:

- a reference to a templated type,
- an integer.

This function must search for the integer in the container.
If one or more occurences of this integer exist in the container, the function returns an `iterator` to the first occurence.
Otherwise, it returns an `iterator` to the end of the container (see `end()`).

You **MUST** use the `find` algorithm from the **STL** to solve this exercise.

The type parameter will always be an STL container of integers and compatible with the `find` algorithm.

The function code is trivial, one line should suffice.

EXERCISE 1 - I WANT MOAR

Turn in: `MyAlgorithms.hpp`

This exercise aims to help you get acquainted with the STL algorithms.
It is simple, but you will need to do some research throughout the language documentation.

We've provided you with a `MyAlgorithms.hpp` file containing some empty functions.
Fill in the blanks using the appropriate STL algorithms.
Your code must compile and return a result similar to the sample `main` function.

You **MUST** use the STL algorithms **EXCLUSIVELY**.
You **MUST NOT** recode these behaviors yourself.

We have provided you with a sample `main` function, which of course is not the one we'll use in the final correction.

The expected output is to be found on the next page.

Terminal

```
========== Step 01 ==========$
Dump (11) 4, 9, 1, 1, 99, 8, 42, 42, 42, -1, 3, $
Dump (10) 99, 1, -42, 21, 12, 21, 99, -7, 42, 42, $
========== Step 02 ==========$
3$
0$
========== Step 03 ==========$
true$
false$
false$
true$
========== Step 04 ==========$
Dump (11) 4, 9, 1, 1, 99, 8, 42, 42, 42, -1, 3, $
Dump (11) 4, 9, 1, -421, -421, 8, 42, 42, 42, -1, 3, $
========== Step 05 ==========$
Dump (11) 4, 9, 1, -421, -421, 8, 42, 42, 42, -1, 3, $
Dump (11) 4, 18, 2, -842, -842, 16, 84, 84, 84, -2, 3, $
========== Step 06 ==========$
Dump (11) 4, 18, 2, -842, -842, 16, 84, 84, 84, -2, 3, $
Dump ( 8) 4, 18, 2, -842, -842, 16, -2, 3, $
========== Step 07 ==========$
Dump (10) 99, 1, -42, 21, 12, 21, 99, -7, 42, 42, $
Dump (10) 42, 42, -7, 99, 21, 12, 21, -42, 1, 99, $
========== Step 08 ==========$
Dump (10) 42, 42, -7, 99, 21, 12, 21, -42, 1, 99, $
Dump ( 9) 42, -7, 99, 21, 12, 21, -42, 1, 99, $
========== Step 09 ==========$
Dump ( 8) 4, 18, 2, -842, -842, 16, -2, 3, $
Dump ( 8) -842, -842, -2, 2, 3, 4, 16, 18, $
========== Step 10 ==========$
Dump ( 9) 42, -7, 99, 21, 12, 21, -42, 1, 99, $
Dump ( 9) 99, 99, 42, 21, 21, 12, 1, -7, -42, $
========== Step 11 ==========$
Dump ( 9) 99, 99, 42, 21, 21, 12, 1, -7, -42, $
Dump ( 9) 21, 21, 12, 1, -7, -42, 99, 99, 42, $
Dump ( 9) 99, 42, 21, 21, 12, 1, -7, -42, 99, $
========== Step 12 ==========$
Dump ( 9) 99, 42, 21, 21, 12, 1, -7, -42, 99, $
Dump ( 9) 99, 42, 777, 777, 12, 1, -7, -42, 99, $
========== Step 13 ==========$
Dump ( 9) 99, 42, 777, 777, 12, 1, -7, -42, 99, $
Dump ( 7) 99, 42, 12, 1, -7, -42, 99, $
========== Step 14 ==========$
Dump ( 8) -842, -842, -2, 2, 3, 4, 16, 18, $
Dump ( 7) -42, -7, 1, 12, 42, 99, 99, $
Dump (15) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, $
Dump (15) -842, -842, -42, -7, -2, 1, 2, 3, 4, 12, 16, 1, 8, 42, 99, 99, $
```

EXERCISE 2 - AVE

Turn in: `Cesar.cpp`, `Cesar.hpp`, `OneTime.cpp`, `OneTime.hpp`

+ PART 1: CAESAR'S NUMBER

This part aims to encode and decode messages sent by **Caesar**.

You must create a `Cesar` class implementing the `IEncryptionMethod` interface (provided in the `IEncryptionMethod.hpp` file).

Here is the description of the methods:

```
// Encode the given character, and display it.
void encryptChar(char plainchar);

// Decode the given character, and display it.
void decryptChar(char cipherchar);

// Reset the internal values to their initial state.
void reset();
```

We must use a slightly different of the Caesar algorithm.

The Caesar algorithm consists in shifting each letter by 3 letters to the right to encode, and 3 letters to the left to decode.
For instance:

- *'a'* becomdes *'d'*
- *'B'* becomes *'E'*
- *'z'* becomes *'c'*

Non-alphabetic characters must not be modified.
Modify this algorithm to shift one more letter to the right every time `encryptChar` is called (you must make up for this in `decryptChar`, of course).

The first character must be shifted by 3, the second by 4, and so on.

The 27th character must logically be shifted by 29, right?
But the alphabet only contains 26 letters, so a 29 letter shift is the same as a 3 letter one!

The `reset` method must reset the internal values, so the shift will reset to a 3 letter shift after the call, as if we had never encrypted or decrypted any characters.

Here is a sample `main` function and its expected output:

```
#include "Cesar.hpp"
#include <string>
#include <iostream>

static void encryptString(IEncryptionMethod& encryptionMethod,
                          std::string const& toEncrypt)
{
        size_t len = toEncrypt.size();

        encryptionMethod.reset();
        for (size_t i = 0; i < len; ++i)
        {
                encryptionMethod.encryptChar(toEncrypt[i]);
        }
        std::cout << std::endl;
}

static void decryptString(IEncryptionMethod& encryptionMethod,
                          std::string const& toDecrypt)
{
        size_t len = toDecrypt.size();

        encryptionMethod.reset();
        for (size_t i = 0; i < len; ++i)
        {
                encryptionMethod.decryptChar(toDecrypt[i]);
        }
        std::cout << std::endl;
}

int main()
{
        Cesar c;

        encryptString(c, "Je clair Luc, ne pas ? Ze woudrai un kekos !");
        decryptString(c, "Mi isirb Xhq, ew jvo ? Zf zszjyir fz ytafk !");
        encryptString(c, "KIKOO");
        encryptString(c, "LULZ XD");
        decryptString(c, "Ziqivun ea Ndcsg.Wji !");
        return 0;
}
```

```
Terminal
~/B-CPP-300> ./a.out | cat -e
Mi isirb Xhq, ew jvo ?  Zf zszjyir fz ytafk !$
Je clair Luc, ne pas ?  Ze woudrai un kekos !$
NMPUV$
OYQF FM$
Welcome to Zombo.Com !$
```

+ Part 2 - One Time Pad

You will now encrypt and decrypt messages using the One time pad algorithm.

This algorithm repeatedly adds (to encrypt) and substracts (to decrypt) a key to the input message.
You must only encrypt alphabetical characters, but will always skip to the key's next character, even when passing a non-alphabetical character.

The alphabet's first letter is at the 0 index.

The key will only contain aphabetical characters.

For instance, encoding

```
A bah zour !
```

with *"bD"* will result in

```
B cdi arvu !
```

The message's case is kept, and the key's case is not taken into account.

Create a `OneTime` class implementing the `IEncryptionMethod` interface.
`OneTime` must have a single constructor, taking the key as parameter:

```
OneTime(const std::string &key);
```

The `reset` method resets the key to its inital position.

Here is a sample `main` function and its expected output:

```
#include "Cesar.hpp"
#include "OneTime.hpp"
#include <string>
#include <iostream>

static void encryptString(IEncryptionMethod& encryptionMethod,
                          std::string const& toEncrypt)
{
        size_t len = toEncrypt.size();

        encryptionMethod.reset();
        for (size_t i = 0; i < len; ++i)
        {
                encryptionMethod.encryptChar(toEncrypt[i]);
        }
        std::cout << std::endl;
```

```
}

static void decryptString(IEncryptionMethod& encryptionMethod,
                          std::string const& toDecrypt)
{
        size_t len = toDecrypt.size();

        encryptionMethod.reset();
        for (size_t i = 0; i < len; ++i)
        {
                encryptionMethod.decryptChar(toDecrypt[i]);
        }
        std::cout << std::endl;
}

int main()
{
        Cesar c;
        OneTime o("DedeATraversLesBrumes");
        OneTime t("TheCakeIsALie");

        encryptString(c, "Je clair Luc, ne pas ? Ze woudrai un kekos !");
        decryptString(c, "Mi isirb Xhq, ew jvo ? Zf zszjyir fz ytafk !");
        encryptString(c, "KIKOO");
        encryptString(c, "LULZ XD");
        decryptString(c, "Ziqivun ea Ndcsg.Wji !");

        encryptString(t, "Prend garde Lion, ne te trompes pas de voie !");
        encryptString(o, "De la musique et du bruit !");
        encryptString(t, "Kion li faras? Li studas kaj programas!");

        decryptString(t, "Iyipd kijdp Pbvr, xi le bvhttgs tik om ovmg !");
        decryptString(o, "Gi pa dunmhmp wu xg tuylx !");
        decryptString(t, "Dpsp vm xaciw? Pk cxcvad otq rrykzsmla!");

        return 0;
}
```

Terminal

```
~/B-CPP-300> ./a.out | cat -e
Mi isirb Xhq, ew jvo ?  Zf zszjyir fz ytafk !$
Je clair Luc, ne pas ?  Ze woudrai un kekos !$
NMPUV$
OYQF FM$
Welcome to Zombo.Com !$
Iyipd kijdp Pbvr, xi le bvhttgs tik om ovmg !$
Gi pa dunmhmp wu xg tuylx !$
Dpsp vm xaciw?  Pk cxcvad otq rrykzsmla!$
Prend garde Lion, ne te trompes pas de voie !$
De la musique et du bruit !$
Kion li faras?  Li studas kaj programas!$
```

Exercise 3 - Magic

Turn in: `Cesar.cpp/hpp`, `OneTime.cpp/hpp`, `Encryption.cpp/hpp`

+ Part 1 - Encryption Wrapper

Re-use all the files from the previous exercise and write the `Encryption` class in order to make the following example compile and run as expected.

```
#include "Encryption.hpp"
#include "Cesar.hpp"
#include "OneTime.hpp"
#include <string>
#include <iostream>

static void encryptString(IEncryptionMethod& encryptionMethod,
                          std::string const& toEncrypt)
{
        Encryption e(encryptionMethod, &IEncryptionMethod::encryptChar);

        encryptionMethod.reset();
        size_t len = toEncrypt.size();
        for (size_t i = 0; i < len; ++i)
        {
                e(toEncrypt[i]);
        }
        std::cout << std::endl;
}

static void decryptString(IEncryptionMethod& encryptionMethod,
                          std::string const& toDecrypt)
{
        Encryption e(encryptionMethod, &IEncryptionMethod::decryptChar);

        encryptionMethod.reset();
        size_t len = toDecrypt.size();
        for (size_t i = 0; i < len; ++i)
        {
                e(toDecrypt[i]);
        }
        std::cout << std::endl;
}

int main()
{
        Cesar c;
        OneTime o("DedeATraversLesBrumes");
        OneTime t("TheCakeIsALie");

        encryptString(c, "Je clair Luc, ne pas ? Ze woudrai un kekos !");
        decryptString(c, "Mi isirb Xhq, ew jvo ? Zf zszjyir fz ytafk !");
        encryptString(c, "KIKOO");
        encryptString(c, "LULZ XD");
        decryptString(c, "Ziqivun ea Ndcsg.Wji !");

        encryptString(t, "Prend garde Lion, ne te trompes pas de voie !");
```

```
        encryptString(o, "De la musique et du bruit !");
        encryptString(t, "Kion li faras? Li studas kaj programas!");

        decryptString(t, "Iyipd kijdp Pbvr, xi le bvhttgs tik om ovmg !");
        decryptString(o, "Gi pa dunmhmp wu xg tuylx !");
        decryptString(t, "Dpsp vm xaciw? Pk cxcvad otq rrykzsmla!");

        return 0;
}
```

Terminal

```
~/B-CPP-300> ./a.out | cat -e
Mi isirb Xhq, ew jvo ?  Zf zszjyir fz ytafk !$
Je clair Luc, ne pas ?  Ze woudrai un kekos !$
NMPUV$
OYQF FM$
Welcome to Zombo.Com !$
Iyipd kijdp Pbvr, xi le bvhttgs tik om ovmg !$
Gi pa dunmhmp wu xg tuylx !$
Dpsp vm xaciw?  Pk cxcvad otq rrykzsmla!$
Prend garde Lion, ne te trompes pas de voie !$
De la musique et du bruit !$
Kion li faras?  Li studas kaj programas!$
```

+ PART 2 - STL ALGORITHMS ARE MAGICAL

Add two static member functions to the `Encryption` class:

```
static void encryptString(IEncryptionMethod &encryptionMethod,
                          const std::string &toEncrypt);
static void decryptString(IEncryptionMethod& encryptionMethod,
                          const std::string &toDecrypt);
```

These member functions must have the same behavior as those with the same name from the previous example, but will only contain three lines of code:

- one to reset the encryption object,
- one to call the according STL algorithm,
- one to display the carriage return.

You **MUST** use an STL algorithm to complete this exercise.

Here is a sample `main` function and its expected output:

```
#include "Encryption.hpp"
#include "Cesar.hpp"
#include "OneTime.hpp"
#include <string>
#include <iostream>

int main()
{
        Cesar c;
        OneTime o("DedeATraversLesBrumes");
        OneTime t("TheCakeIsALie");

        Encryption::encryptString(c, "Je clair Luc, ne pas ? Ze woudrai un kekos !");
        Encryption::decryptString(c, "Mi isirb Xhq, ew jvo ? Zf zszjyir fz ytafk !");
        Encryption::encryptString(c, "KIKOO");
        Encryption::encryptString(c, "LULZ XD");
        Encryption::decryptString(c, "Ziqivun ea Ndcsg.Wji !");

        Encryption::encryptString(t, "Prend garde Lion, ne te trompes pas de voie !")
            ;
        Encryption::encryptString(o, "De la musique et du bruit !");
        Encryption::encryptString(t, "Kion li faras? Li studas kaj programas!");

        Encryption::decryptString(t, "Iyipd kijdp Pbvr, xi le bvhttgs tik om ovmg !")
            ;
        Encryption::decryptString(o, "Gi pa dunmhmp wu xg tuylx !");
        Encryption::decryptString(t, "Dpsp vm xaciw? Pk cxcvad otq rrykzsmla!");

        return 0;
}
```

```
Terminal
~/B-CPP-300> ./a.out | cat -e
Mi isirb Xhq, ew jvo ?  Zf zszjyir fz ytafk !$
Je clair Luc, ne pas ?  Ze woudrai un kekos !$
NMPUV$
OYQF FM$
Welcome to Zombo.Com !$
Iyipd kijdp Pbvr, xi le bvhttgs tik om ovmg !$
Gi pa dunmhmp wu xg tuylx !$
Dpsp vm xaciw?  Pk cxcvad otq rrykzsmla!$
Prend garde Lion, ne te trompes pas de voie !$
De la musique et du bruit !$
Kion li faras?  Li studas kaj programas!$
```

Exercise 4 - Meta

Turn in: `Container.hpp`

Create a `contain` class template that takes as a type parameter a "contained" type, and as a second type parameter an STL container.

Only scalar types will be contained

The `contain` class must have as an attribute an instance of the STL container, containing the "contained" type.

Implement two member functions:

- `void aff()`: walks through the container and calls the `aff` function described below
- `void add()`: walks through the container and calls the `add` function described below

Implement two member function templates, taking as parameter the type contained in the container:

- `void aff(?? b)`: prints *"Value: 7"*, if `b` is 7
- `void add(?? b)`: increments `b`, modifying the value in the container

Here is a sample `main` function and its expected output:

```
int main()
{
        contain<char, std::list> test;
        test.push('t');
        test.aff();
        test.add();
        test.aff();
        contain <int, std::vector> test2;
        test2.push(1);
        test2.aff();
        test2.add();
        test2.aff();
        return 0;
}
```

Terminal

```
~/B-CPP-300> ./a.out
Value:  t
Value:  u
Value:  1
Value:  2
```

The End Has No End

Congratulations. You've made it.
You survived the C++ Pool.

Here are a few suggestions:

- test everything at least three times,
- go back to the previous days and try to finish the exercises you may have failed,
- make sure you've perfectly understood everything you learnt. These are tools you'll be using throughout your life as an Object-Oriented developer,
- take a look at some of the latest and greatest features of modern C++! Browse the web and see what's new in the latest versions (big news: C++14 is growing old),
- try doing some meta-programming!