C++ Pool - d17

Algorithm

Koalab koala@epitech.eu

Abstract: *This document is the subject of d17: STL algorithms*

Contents

Chapter I

GENERAL RULES

- READ THE GENERAL RULES CAREFULLY!!

  You will have no possible excuse if you end up with a 0 because you didn't follow one of the general rules

- GENERAL RULES:

  - STL is something you need to see. So do all the exercises today. This IS NOT progressive. So don't be stupid and do all this day, STL is one of the C++'s vital concepts.

  - Classes have to be canonical (understand: we will test that, and if your classes are not, your exercise score will be 0).

  - The C language is not today's topic. All `[asvn]printf, *alloc` functions are forbiden.

- COMPILATION OF THE EXERCISES:

  - The Koalinette compiles your code with the following flags : `-W -Wall -Werror`

  - To avoid compilation problems with the Koalinette, include every required headers in your headers.

  - Note that none of your files must contain a `main` function. We will use our own to compile and test your code.

  - This subject may be modified up to 4h before turn-in time. Refresh it regularly !

  - The turn-in dirs are `(piscine_cpp_d17/exN)` , N being the exercise number

Chapter II

Exercise 0

	Exercise : 00	points : 4
	Find me that	
Turn-in directory: `(piscine_cpp_d17)/ex00`		
Compiler: `g++`	Compilation flags: `-W -Wall -Werror`	
Makefile: `No`	Rules: `n/a`	
Files to turn in : `find.hpp`		
Remarks : `n/a`		
Forbidden functions : `*alloc - free - *printf`		

In this exercise, you will create a template function `do_find` taking 2 parameters:

- A reference on the template type

- An integer

This function must search for the integer given in parameter in the container also given in parameter. If one or more occurence of this integer does exist in the container, a `iterator` on the first occurence will be returned. Otherwise, a `iterator` on the end will be returned (see `end()`).

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

Postulate: the template type will always be a STL container, containing integers and compatible with the STL `find` algorithm.

> Resolving symbols defined in a template type is not always possible with your compiler. typename could be useful.
> The function code is trivial, one line should be enough.

Chapter III

Exercise 1

	Exercise : 01	points : 6
	STL Algorithmsssssssss	
Turn-in directory: `(piscine_cpp_d17)/ex01`		
Compiler: `g++`	Compilation flags: `-W -Wall -Werror`	
Makefile: `No`	Rules: `n/a`	
Files to turn in : `MyAlgorithms.hpp`		
Remarks : `n/a`		
Forbidden functions : `*alloc - free`		

The aim of this exercise is to acquaint yourself with the STL algorithms.

It is simple, but requires some research in the documentation.

We provide to you a `MyAlgorithms.hpp` which contains some blank functions. You will have to fill the blanks using the appropriate STL algorithm. The final goal is that your code compile and returns a result similar with the sample main below.

> ⚠️ You have to use EXCLUSIVELY STL algorithms to solve this exercise, you SHALL NOT recode these behaviors yourself. Simply call the right algorithm!

You SHALL NOT edit the functions prototypes, simply replace the XXXXXXXXXX with the right call!

***** Example *****

We provide you with a sample main. This main is not the one that will be used at correction.

***** Output *****

```
Thefly$> g++ -W -Werror -Wall main.cpp -o test
Thefly$> ./test | cat -e
=========== 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, $
Thefly$>
```

> 💡 Take into consideration the function names and the given output in
> order to guess which algorithm must be used.

Chapter IV

Exercise 2

	Exercise : 02	points : 5
	Ave Cesar & One time pad	
	Turn-in directory: `(piscine_cpp_d17)/ex02`	
	Compiler: `g++`	Compilation flags: `-W -Wall -Werror`
	Makefile: `No`	Rules: `n/a`
	Files to turn in : `Cesar.cpp, Cesar.h, OneTime.cpp, OneTime.h`	
	Remarks : `n/a`	
	Forbidden functions : `*alloc - free - *printf`	

Provided file:

- IEncryptionMethod.h

IV.1 Part 1/2: Caesar's number

The aim of this part is to encode and decode messages sent by Caesar.

You therefore need to create a `Cesar` class implementing the `IEncryptionMethod` interface that is provided in the `IEncryptionMethod.h` file.
We will compile your files in copying this `IEncryptionMethod.h` file in the same folder, you can so include it, but not modify it!
Methods description:

```
1  // Encode the given character, and display it.
2  void encryptChar(char plainchar);
3
4  // Decode the given character, and display it.
5  void decryptChar(char cipherchar);
6
7  // Resets the internal values to their initial state.
```

```
8   void reset();
```

We will use a slightly different version of the Caesar algorithm.

The Caesar algorith consists in shifting each letter 3 letters to the right to encode, 3 letters to the left to decode.

For example:

- a becomes d

- B becomes E

- z becomes c // Yes, after 'z' we come back to 'a'

> 💡 As you see, we keep the case.

The non-alphabetic characters must not be edited.
We will modify this algorithm to shift one letter more to the right each time `encryptChar` is called (the same behavior applies to the `decryptChar` method, only it is each time one letter more to the left which has to be shifted).

The first character will be shifted from 3, the second from 4, and so on.
The 27th character will logically be shifted from 29, right? But the alphabet only contains 26 characters, so a 29 letters shifting is strictly equivalent to a 3 letters shifting!

The `reset` method must reset the internal values, so the shifting will reset to a 3 letters shifting after the call, as if we never encrypted / decrypted any character.
The `decryptChar` will do the `encryptChar` reverse operation, it will transform an encoded character into a decoded character.

> 💡 A char's maximal value is 127, so be careful during your computations! Be also careful with its minimal value during subtractions.

***** Example *****

```
1  #include ''Cesar.h''
2  #include <string>
3  #include <iostream>
4
5  static void encryptString(IEncryptionMethod& encryptionMethod,
6                            std::string const& toEncrypt)
7  {
8    size_t len = toEncrypt.size();
9
10   encryptionMethod.reset();
11   for (size_t i = 0; i < len; ++i)
12   {
13     encryptionMethod.encryptChar(toEncrypt[i]);
14   }
15   std::cout << std::endl;
16 }
17
18 static void decryptString(IEncryptionMethod& encryptionMethod,
19                           std::string const& toDecrypt)
20 {
21   size_t len = toDecrypt.size();
22
23   encryptionMethod.reset();
24   for (size_t i = 0; i < len; ++i)
25   {
26     encryptionMethod.decryptChar(toDecrypt[i]);
27   }
28   std::cout << std::endl;
29 }
30
31 int main()
32 {
33   Cesar c;
34
35   encryptString(c, ''Je clair Luc, ne pas ? Ze woudrai un kekos !'');
36   decryptString(c, ''Mi isirb Xhq, ew jvo ? Zf zszjyir fz ytafk !'');
37   encryptString(c, ''KIKOO'');
38   encryptString(c, ''LULZ XD'');
39   decryptString(c, ''Ziqivun ea Ndcsg.Wji !'');
40   return 0;
41 }
```

***** Sortie *****

```
1  Thefly$> g++ -W -Werror -Wall Cesar.cpp main.cpp -o test
2  Thefly$> ./test | cat -e
3  Mi isirb Xhq, ew jvo ? Zf zszjyir fz ytafk !$
4  Je clair Luc, ne pas ? Ze woudrai un kekos !$
```

```
5 NMPUV$
6 OYQF FM$
7 Welcome to Zombo.Com !$
8 Thefly$>
```

IV.2 Part 2/2: One Time Pad

The aim of this part is to crypt and uncrypt messages with the `One time pad` algorithm.
More information about this algorithm here:
http://en.wikipedia.org/wiki/One-time_pad

This algorithm repeatedly adds (to encode) or substracts (to decode) a key to your message. You will only encode alphabetical characters, but you will skip to the next key's character anyway. The alphabet's first letter is at the 0 position.

The key will only contain alphabetical characters.

For example, encode "A bah zour !" with "bD" will give "B cdi arvu !" (we keep the message's case, the key's case is without importance)

You therefore need to create a `OneTime` class implementing the `IEncryptionMethod` interface that is provided in the `IEncryptionMethod.h`.
We will compile your files in copying this `IEncryptionMethod.h` file in the same folder, you can so include it, but not modify it!
Methods description:

```
1  // Encode the given character, and display it.
2  void encryptChar(char plainchar);
3
4  // Decode the given character, and display it.
5  void decryptChar(char cipherchar);
6
7  // Resets the internal values to their initial state.
8  void reset();
```

`OneTime` will only have one constructor, taking the key as argument:

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

The `reset` method will reset the key to its initial position.

> A char's maximal value is 127, so be careful during your computations! Be also careful with its minimal value during subtractions.

***** Example *****

```cpp
1  #include ``Cesar.h''
2  #include ``OneTime.h''
3  #include <string>
4  #include <iostream>
5  static void encryptString(IEncryptionMethod& encryptionMethod,
6                       std::string const& toEncrypt)
7  {
8    size_t len = toEncrypt.size();
9
10   encryptionMethod.reset();
11   for (size_t i = 0; i < len; ++i)
12   {
13     encryptionMethod.encryptChar(toEncrypt[i]);
14   }
15   std::cout << std::endl;
16 }
17
18 static void decryptString(IEncryptionMethod& encryptionMethod,
19                       std::string const& toDecrypt)
20 {
21   size_t len = toDecrypt.size();
22
23   encryptionMethod.reset();
24   for (size_t i = 0; i < len; ++i)
25   {
26     encryptionMethod.decryptChar(toDecrypt[i]);
27   }
28   std::cout << std::endl;
29 }
30
31 int main()
32 {
33   Cesar c;
34   OneTime o(``DedeATraversLesBrumes'');
35   OneTime t(``TheCakeIsALie'');
36
37   encryptString(c, ``Je clair Luc, ne pas ? Ze woudrai un kekos !'');
38   decryptString(c, ``Mi isirb Xhq, ew jvo ? Zf zszjyir fz ytafk !'');
39   encryptString(c, ``KIKOO'');
40   encryptString(c, ``LULZ XD'');
41   decryptString(c, ``Ziqivun ea Ndcsg.Wji !'');
42
43   encryptString(t, ``Prend garde Lion, ne te trompes pas de voie !'');
44   encryptString(o, ``De la musique et du bruit !'');
45   encryptString(t, ``Kion li faras? Li studas kaj programas!'');
46
47   decryptString(t, ``Iyipd kijdp Pbvr, xi le bvhttgs tik om ovmg !'');
48   decryptString(o, ``Gi pa dunmhmp wu xg tuylx !'');
49   decryptString(t, ``Dpsp vm xaciw? Pk cxcvad otq rrykzsmla!'');
```

11

```
50
51   return 0;
52 }
```

***** Output *****

```
1 Thefly$> g++ -W -Werror -Wall Cesar.cpp OneTime.cpp main.cpp -o test
2 Thefly$> ./test | cat -e
3 Mi isirb Xhq, ew jvo ? Zf zszjyir fz ytafk !$
4 Je clair Luc, ne pas ? Ze woudrai un kekos !$
5 NMPUV$
6 OYQF FM$
7 Welcome to Zombo.Com !$
8 Iyipd kijdp Pbvr, xi le bvhttgs tik om ovmg !$
9 Gi pa dunmhmp wu xg tuylx !$
10 Dpsp vm xaciw? Pk cxcvad otq rrykzsmla!$
11 Prend garde Lion, ne te trompes pas de voie !$
12 De la musique et du bruit !$
13 Kion li faras? Li studas kaj programas!$
14 Thefly$>
```

Chapter V

Exercise 3

	Exercise : 03	points : 6
	STL algorithms are magical	

Turn-in directory: (piscine_cpp_d17)/ex03	
Compiler: g++	Compilation flags: -W -Wall -Werror
Makefile: No	Rules: n/a
Files to turn in : Cesar.cpp, Cesar.h, OneTime.cpp, OneTime.h, Encryption.cpp, Encryption.h	
Remarks : n/a	
Forbidden functions : *alloc - free - *printf	

V.1   Part 1/2: Encryption Wrapper

Take back all your files from the previous exercise, and write the `Encryption` class in order to make the following example work and compile correctly:

***** Example *****

```
1  #include ''Encryption.h''
2  #include ''Cesar.h''
3  #include ''OneTime.h''
4  #include <string>
5  #include <iostream>
6
7  static void encryptString(IEncryptionMethod& encryptionMethod,
8                            std::string const& toEncrypt)
9  {
10   Encryption e(encryptionMethod, &IEncryptionMethod::encryptChar);
11
12   encryptionMethod.reset();
13   size_t len = toEncrypt.size();
14   for (size_t i = 0; i < len; ++i)
15   {
16     e(toEncrypt[i]);
17   }
18   std::cout << std::endl;
19  }
20
21  static void decryptString(IEncryptionMethod& encryptionMethod,
22                            std::string const& toDecrypt)
23  {
24   Encryption e(encryptionMethod, &IEncryptionMethod::decryptChar);
25
26   encryptionMethod.reset();
27   size_t len = toDecrypt.size();
28   for (size_t i = 0; i < len; ++i)
29   {
30     e(toDecrypt[i]);
31   }
32   std::cout << std::endl;
33  }
34  int main()
35  {
36   Cesar c;
37   OneTime o(''DedeATraversLesBrumes'');
38   OneTime t(''TheCakeIsALie'');
39
40   encryptString(c, ''Je clair Luc, ne pas ? Ze woudrai un kekos !'');
41   decryptString(c, ''Mi isirb Xhq, ew jvo ? Zf zszjyir fz ytafk !'');
42   encryptString(c, ''KIKOO'');
43   encryptString(c, ''LULZ XD'');
44   decryptString(c, ''Ziqivun ea Ndcsg.Wji !'');
45
46   encryptString(t, ''Prend garde Lion, ne te trompes pas de voie !'');
47   encryptString(o, ''De la musique et du bruit !'');
48   encryptString(t, ''Kion li faras? Li studas kaj programas!'');
49
```

```
50   decryptString(t, ''Iyipd kijdp Pbvr, xi le bvhttgs tik om ovmg !'');
51   decryptString(o, ''Gi pa dunmhmp wu xg tuylx !'');
52   decryptString(t, ''Dpsp vm xaciw? Pk cxcvad otq rrykzsmla!'');
53
54   return 0;
55 }
```

***** Output *****

```
1 Thefly$> g++ -W -Werror -Wall Cesar.cpp OneTime.cpp Encryption.cpp main.cpp \
2         -o test
3 Thefly$> ./test | cat -e
4 Mi isirb Xhq, ew jvo ? Zf zszjyir fz ytafk !$
5 Je clair Luc, ne pas ? Ze woudrai un kekos !$
6 NMPUV$
7 OYQF FM$
8 Welcome to Zombo.Com !$
9 Iyipd kijdp Pbvr, xi le bvhttgs tik om ovmg !$
10 Gi pa dunmhmp wu xg tuylx !$
11 Dpsp vm xaciw? Pk cxcvad otq rrykzsmla!$
12 Prend garde Lion, ne te trompes pas de voie !$
13 De la musique et du bruit !$
14 Kion li faras? Li studas kaj programas!$
15 Thefly$>
```

V.2   Part 2/2: STL algorithms are magical

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

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

These member functions will have the same behavior as functions having the same name from the abobe example, but will only contain three code lines:

- The first one to reset the encryption class ;

- The second one to call the right STL algorithm ;

- The last one to display the carriage return.

You MUST use a STL algorithm to validate this exercise.

***** Example *****

```
1  #include ''Encryption.h''
2  #include ''Cesar.h''
3  #include ''OneTime.h''
4  #include <string>
5  #include <iostream>
6
7  int main()
8  {
9    Cesar c;
10   OneTime o(''DedeATraversLesBrumes'');
11   OneTime t(''TheCakeIsALie'');
12
13   Encryption::encryptString(c, ''Je clair Luc, ne pas ? Ze woudrai un kekos
         !'');
14   Encryption::decryptString(c, ''Mi isirb Xhq, ew jvo ? Zf zszjyir fz ytafk
         !'');
15   Encryption::encryptString(c, ''KIKOO'');
16   Encryption::encryptString(c, ''LULZ XD'');
17   Encryption::decryptString(c, ''Ziqivun ea Ndcsg.Wji !'');
18
19   Encryption::encryptString(t, ''Prend garde Lion, ne te trompes pas de voie
         !'')\
20 ;
21   Encryption::encryptString(o, ''De la musique et du bruit !'');
22   Encryption::encryptString(t, ''Kion li faras? Li studas kaj programas!'');
23
24   Encryption::decryptString(t, ''Iyipd kijdp Pbvr, xi le bvhttgs tik om ovmg
         !'')\
25 ;
26   Encryption::decryptString(o, ''Gi pa dunmhmp wu xg tuylx !'');
27   Encryption::decryptString(t, ''Dpsp vm xaciw? Pk cxcvad otq rrykzsmla!'');
28
29   return 0;
30 }
```

***** Output *****

```
1  Thefly$> g++ -W -Werror -Wall Cesar.cpp OneTime.cpp Encryption.cpp main.cpp \
2          -o test
3  Thefly$> ./test | cat -e
4  Mi isirb Xhq, ew jvo ? Zf zszjyir fz ytafk !$
5  Je clair Luc, ne pas ? Ze woudrai un kekos !$
6  NMPUV$
7  OYQF FM$
8  Welcome to Zombo.Com !$
9  Iyipd kijdp Pbvr, xi le bvhttgs tik om ovmg !$
10 Gi pa dunmhmp wu xg tuylx !$
11 Dpsp vm xaciw? Pk cxcvad otq rrykzsmla!$
```

```
12 Prend garde Lion, ne te trompes pas de voie !$
13 De la musique et du bruit !$
14 Kion li faras? Li studas kaj programas!$
15 Thefly$>
```

Chapter VI

Exercise 4

	Exercise : 04	points : 4
	The Meta-Container	
Turn-in directory: (piscine_cpp_d17)/ex04		
Compiler: g++	Compilation flags: -W -Wall -Werror	
Makefile: No	Rules: n/a	
Files to turn in : Container.hpp		
Remarks : n/a		
Forbidden functions : *alloc - free		

In this exercise you can use algorithms, but this is more just for fun.

You will create a meta-container.

Create a template class `contain` to receive as a template parameter the type that will be contained by the STL container, and as a second parameter the STL container.

> Only scalar types will be stored.

The class `contain` will have the following attribute:

- The STL container given as a template parameter, containing the type given as first parameter to our template class.

You will then do two member functions:

- `void aff()`, which must browse the container using `for_each` and call the `aff` function described below.

- `void add()` , which must browse the container using `for_each` and call the `add` function described below.

You must now create two template functions, which take as template parameter the type of objects contained by the container.

- `void aff(??  b)` will display "Valeur : 7", 7 being the value taken as the b parameter. A carriage return follows this output.

- `void add(??  b)` will add +1 to the value passed as a parameter, and will modify the value contained in the container

Example:

```
1  int main()
2  {
3   contain<char, std::list> test;
4   test.push('t');
5   test.aff();
6   test.add();
7   test.aff();
8   contain <int, std::vector> test2;
9   test2.push(1);
10  test2.aff();
11  test2.add();
12  test2.aff();
13  return 0;
14 }
```

Output:

```
1  thefly_d$>./a.out
2  Valeur : t
3  Valeur : u
4  Valeur : 1
5  Valeur : 2
6  thefly_d$>
```

Chapter VII

Exercise 5

If you succeeded in finishing the previous exercises and you still have some time before tonight's exam, here are a few suggestions:

- Test everything at least three times

- Review the notions learnt for tonight's exam

- Go back to your previous pool days, and try again to do exercises that you failed

- Be sure to have perfectly understood every principle seen in this pool. A good way to check it is to train yourselves to explain these principles. Try it!

- If you are still reading this, it's because you're not honest with yourself, check back the previous points

- You can still have fun pushing C++ to its limits, have a look to the evolutions brought by C++11, and why not have fun doing some meta-programming?