C++ Pool - d15

Templates

Koalab koala@epitech.eu

Abstract: This document is the subject for d15: Templates

Contents

Chapter I

BASIC RULES

- PLEASE READ BASIC RULES CAREFULLY!!!!!
    - You will have no excuse if you get a 0 because you forget one of the basic rules...
- BASIC RULES:
    - Today's topic is templates. You can consequently implement within headers what must logically be implemented within headers.
    - Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.
    - The imposed filenames must be followed TO THE LETTER, as well as class, function and method names.
    - You're coding in C++ now, and not in C. Therefore the following functions are FORBIDDEN, and their use will be punished by a -42, no question asked:
        * `*alloc`
        * `*printf`
        * `free`
        * `open`, `fopen`, etc...
    - Generally, files associated to a class will always be `CLASS_NAME.hh` and `CLASS_NAME.cpp` (class name can be in lower case if applicable).
    - Turn in directories are ex00, ex01, ..., exN

- Any use of `friend` or of `dynamic_cast` will result in the grade -42, `no questions asked` .
- Some of the exercises will be corrected thanks to a memory debugger. This is to verify that your allocations are in line with what is expected. Don't expect to succeed to an exercises thanks to an int tab[100000], while a new int[5] is required.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise description.
- Read each exercise FULLY before starting it!
- USE YOUR BRAIN. Please!

- EXERCISES COMPILATION:
  - The Koalinette compiles your code with the following flags: `-W -Wall -Werror -std=c++03`
  - In order to avoid compilation issues with the Koalinette, please include necessary files within the include files (*.hh) and (*.hpp).
  - Please note that none of your files must contain the `main` function. We will use our own `main` to compile and test your code.
  - The turn in repository is : `(cpp_d15)/exN` (N being the exercise number.)

Chapter II

Exercise 0

	Exercise : 00	points : 2
Swap, min, max, add		
Turn-in directory: `(cpp_d15)/ex00`		
Compiler: `g++`	Compilation flags: `-W -Wall -Werror -std=c++03`	
Makefile: `No`	Rules: `n/a`	
Files to turn in : `ex00.hpp`		
Remarks : `n/a`		
Forbidden functions : `*alloc - free - *printf`		

The platypus:

```
 The platypus is a small mammal, semi-aquatic, present only in the
 east of Australia.  It is one of the five species of Monotremes, and
 the only mammal laying eggs instead of giving birth to live youngs.
(The four others being echidna species)

 The Platypus (Ornithorhynchus  anatinus) looks like a Beaver:
 its body and its tail are wide and flat, covered with brown fur.
 The Platypus has webbed feet and a large, rubbery snout. This
 is why he has the nickname of "duck-billed platypus." The tail is
usually 10 to 15 cm long. Males are usually fatter than Females, and
Male size is usually a third longer than Female's. From
40 to 50 cm, it varies considerably from one region to another without
being related to the climate.
```

This exercise is not about Platypuses.

You have to write the following function templates:

- `swap` : Swaps the values of two arguments. Does not return anything.

- `min` : Compares the two arguments and returns the smallest one. If the two arguments are equal, then it returns the second one.

- `max` : Compares the two arguments and returns the biggest one. If the two arguments are equal, then it returns the second one.

- `add` : Adds the two arguments, and returns the result of said addition. Use a return by copy and not a return by reference.

These functions can be called with any type of argument, with the condition that the two arguments has the same type and supports all comparison operators.

The following code must compile:

```
#include <iostream>
#include <string>
#include "ex00.hpp"

int main()
{
        int a = 2;
        int b = 3;

        ::swap(a, b);
        std::cout << "a = " << a << ", b = " << b << std::endl;
        std::cout << "min(a, b) = " << ::min(a, b) << std::endl;
        std::cout << "max(a, b) = " << ::max(a, b) << std::endl;
        std::cout << "add(a, b) = " << ::add(a, b) << std::endl;

        std::string c = "chaine1";
        std::string d = "chaine2";

        ::swap(c, d);
        std::cout << "c = " << c << ", d = " << d << std::endl;
        std::cout << "min(c, d) = " << ::min(c, d) << std::endl;
        std::cout << "max(c, d) = " << ::max(c, d) << std::endl;
        std::cout << "add(c, d) = " << ::add(c, d) << std::endl;
}
```

The output must be:

```
a = 3, b = 2
min(a, b) = 2
max(a, b) = 3
add(a, b) = 5
c = chaine2, d = chaine1
min(c, d) = chaine1
max(c, d) = chaine2
add(c, d) = chaine2chaine1
```

Chapter III

Exercise 1

	Exercise : 01	points : 2
Compare		
Turn-in directory: `(cpp_d15)/ex01`		
Compiler: `g++`	Compilation flags: `-W -Wall -Werror -std=c++03`	
Makefile: `No`	Rules: `n/a`	
Files to turn in : `ex01.hpp`		
Remarks : `n/a`		
Forbidden functions : `*alloc - free - *printf`		

Emmanuelle's recipe of the Petits Lu cake. Extracted from the excellent book: "Je veux du Chocolat" from Trish Deseine:

```
Break into large chunks 3/4 of a package of Petits Lu, with 40g
of meringue.
Melt 100g of chocolate with 300g of butter. Let it cool down,
add two eggs lightly beaten, 150g of sugar and 50g of
cacao. Pout it on the pieces of Lu and of meringue, mix
well and put it all in a silicone or filmed cake mold. Compress it a
bit and let it chill for at least 6 hours.
```

Use these 6 hours to write the function template `compare`.

This function gets as arguments two parameters with the same type, and returns an int, as described below:

- 0 : The two parameters are equals.
- -1: The first parameter is lower than the second one.
- 1: The first parameter is greater than the second one.

This function must run properly for `const char *` (by calling explicitly the correct overload.)

Your function template must compile with the following code:

```
class toto
{
  private:
    toto &operator=(const toto&) {return *this;}
    toto(const toto &){}
  public:
    bool operator==(const toto&) const {return true;}
    bool operator>(const toto&) const {return false;}
    bool operator<(const toto&) const {return false;}
    toto(){}
};
```

The output must be :

```
toto a, b;
compare(a, b) return 0
compare(1, 2) return -1
compare<const char*>("chaineZ", "chaineA42") return 1
const char *s1 = "42", *s2 = "lulz";
compare(s1, s2) return -1
```

Unlike what you can believe,"chainZ" and "chainA42" are not const char* but respectively a const char[8] and a const char[10].
In order to use the right overload, it is necessary either to static_cast, or to call explicitly compare<const char*> as shown in this example.

Chapter IV

Exercise 2

	Exercise : 02	points : 3
min, the return		
Turn-in directory: `(cpp_d15)/ex02`		
Compiler: `g++`	Compilation flags: `-W -Wall -Werror -std=c++03`	
Makefile: `No`	Rules: `n/a`	
Files to turn in : `ex02.hpp`		
Remarks : `n/a`		
Forbidden functions : `None`		

```
Deoxyribonucleic acid (DNA) is the support of the genetic
information for all known living organisms. Eukaryotic
organisms have their DNA localized in the kernel of each
cells of their organism. Prokaryotes organisms have their
DNA free in the cells' cytoplasm.
 DNA consists of two chains (or strands) that are twisted
together like a twisted ladder and called the double helix, which
diameter is 2nm (10^-9m). Each strand is a sequence of
nucleobases which consists of a sugar (deoxyribose),
a phosphate group and a nitrogenous base.
There are four type of nitrogenous base: Adenine (A),
Cytosine (C), Guanine (G), Thymine (T). These bases
complement each others, two by two: Adenine with
Thymine and Guanine with Cytosine. This complementarity
allows the binding of the two strands.
```

In order to satisfy your reproductive instinct, you need to become rich. You picked up computer science accordingly, in order to fulfill your ambitions. To reach this goal you have to code the following functions:

- A function template `min` that accepts two parameters of the same type and that returns the smallest of the two arguments. This function must displays “template min” on the standard output (without quotes.) If the two parameters are equals, return the first one.

- A function `min` that accepts two integer as parameters and that returns the smallest of the two arguments. This function must display “non-template min” on the standard output (without quotes.) If the two parameters are equals, return the first one.

- A function template `templateMin` that accepts an array and its size as parameters and that returns the smallest value of this array. This function must call the function template `min`.

- A function `nonTemplateMin` that accepts an array of integers and its size and that returns the smallest value of this array. This function must call the non function template `min`.

Thus, if correctly used, the following code:

```
1       int tab[2] = {3, 0};
2       int minimum = templateMin(tab, 2);
3       cout << "templateMin(tab, 2) = " << minimum << endl;
4       minimum = nonTemplateMin(tab, 2);
5       cout << "nonTemplateMin(tab, 2) = " << minimum << endl;
```

should display :

```
1       template min
2       templateMin(tab, 2) = 0
3       non-template min
4       nonTemplateMin(tab, 2) = 0
```

Chapter V

Exercise 3

	Exercise : 03	points : 4
foreach		
Turn-in directory: `(cpp_d15)/ex03`		
Compiler: `g++`	Compilation flags: `-W -Wall -Werror -std=c++03`	
Makefile: `No`	Rules: `n/a`	
Files to turn in : `ex03.hpp`		
Remarks : `n/a`		
Forbidden functions : `None`		

```
In 1221, Theodore Komnenos Doukas, the ruler of Epirote, took
  Serres to conquest the Kingdom of Thessalonica. He pursued
 the expansion of his kingdom in the following years, with a
 crucial move in 1225 when he took Corfu and Durres from the
 Venetians, Adriople from the Byzantines of Nicea, and Xanthi and
 Didymoteicho from the Latins. Theodore Komnenos Doukas
 arranged his coronation as emperor the same year.
 He got defeated in 1230 at the battle of Klokotnista by the
Kingdom of Bulgaria. He was captured during this battle by the
tsar Ivan Asen II who later gouged his eyes for his involvement in a
conspiracy.
```

To avoid a similar fate than Theodeore's, you must code the function template `foreach`.

This function allows to skim through an array by calling a function for each element of this array. The function accepts as argument the address of the beginning of the array, a reference on function and the size of the array. The reference on function corresponds to the following prototype: `void func(const type& elem);`

Moreover you must provide the function `print` that is passed to the function `foreach` and that displays each elements, one per line, whatever their type.

We must be able to use your work as shown below:

```
#include "ex03.hpp"

int main(void) {
   int tab[] = { 11, 3, 89, 42 };
   foreach(tab, print<int>, 4);
   std::string tab2[] = { "j'", "aime", "les", "templates", "!" };
   foreach(tab2, print, 5);
   return 0;
}
```

Which displays:

```
(koala@arbre)$./a.out | cat -e
11$
3$
89$
42$
j'$
aime$
les$
templates$
!$
```

Chapter VI

Exercise 4

	Exercise : 04	points : 4
Test		
Turn-in directory: `(cpp_d15)/ex04`		
Compiler: `g++`	Compilation flags: `-W -Wall -Werror -std=c++03`	
Makefile: `No`	Rules: `n/a`	
Files to turn in : `ex04.cpp, ex04.hpp`		
Remarks : `n/a`		
Forbidden functions : `None`		

```
  En 1985, Harold Kroto, James R. Health, Sean O'Brien and Richard
 Malley prepared a new allotropic form for the carbon, the molecule
 C60 which is based on 60 atoms of carbon. The structure form a
 regular polyhedron shaped with hexagonal and pentagonal facets.
Each atom of carbon is linked with three others. This form is known
as Buckminsterfullerene or Buckyball and is named after the American
architect and inventor Richard Buckminster Fuller who created several
geodesic domes, which forms are similar to the C60.

  More Generally, the fullrenes, within which the C60, are a new carbon
  family. Their surfaces is non equilateral and is composed of a
  combination of hexagons and pentagons, just like the facets of a football.
  This disposition provides a a closed structure similar to a carbon cage.
  However, the synthesis process to obtain these molecules in macroscopic
  quantities was only discover in 1990 by Huffman and Kramer from the
  Heidelberg University. Nanotubes have been identified six year later from
a synthesis by-product of the fullrenes.
```

We won't do physics anymore for today and fortunately, the upcoming exercise is way more easy that the creation of the fullrenes.

Write the function `equal` that is templated with a type `T` and that accepts two parameters of type reference on `const T`. This function must return `true` if the two parameters are equals, and `false` if not.

Write a class `Tester` that is templated on a type `T`. This class provides a member function `equal` that accepts two parameters of the type reference on `const T`. This member function returns `true` if the two parameters are equals , and `false` if not.

You don't have to be concerned about making the class `Tester` canonical.

You must only put your declarations in the .hpp. You are not allowed to put neither the body of the function nor the member function outside the .cpp file you turn in.

You have to instantiate your function and class templates for the following types:

- `int`
- `float`
- `double`
- `std::string`

The following code:

```
1   #include <iostream>
2   #include "ex04.hpp"
3
4   int main()
5   {
6      std::cout << "1 == 0 ? " << equal(1, 0) << std::endl;
7      std::cout << "1 == 1 ? " << equal(1, 1) << std::endl;
8
9      Tester<int> iT;
10
11     std::cout << "41 == 42 ? " << iT.equal(41, 42) << std::endl;
12     std::cout << "42 == 42 ? " << iT.equal(42, 42) << std::endl;
13     return 0;
14  }
```

displays :

```
1   (pouill_g@tatooine 497)./ex04 | cat -e
2   1 == 0 ? 0$
3   1 == 1 ? 1$
```

```
4       41 == 42 ? 0$
5       42 == 42 ? 1$
```

Chapter VII

Exercise 5

	Exercise : 05	points : 4
array<T>		
Turn-in directory: `(cpp_d15)/ex05`		
Compiler: `g++`	Compilation flags: `-W -Wall -Werror -std=c++03`	
Makefile: `No`	Rules: `n/a`	
Files to turn in : `ex05.hpp`		
Remarks : `n/a`		
Forbidden functions : `None`		

```
India has over a billion peoples (2000) This makes India the second
most populated country in the world, after China.
However, while China is more or less able to control its population
growth, India's population is still increasing rapidly with an approximative
growth rate of 19 millions people per year, which is the consequence of
a global fertility rate of 2.7 children per women, against 1.7 for China. Thus
India is expected to become the most populated country in the world by
2035.
Demographers are less alarmed by the numbers (Indian fertility collapsed
over the last 50 years) than by the irregular and relatively slow trend.
This is due to the demographic policy which is at the same time brutal and
incoherent (while China in the meantime focused on the easy and sometime
brutal, but applicable,  single child policy.)
India focuses its politic more on the individual responsibility (with informati
center on contraception.) Moreover India is a democracy, so this policy got
ups and down, while China where the single children policy remains the same
since its creation (with small privileges for rural area.)
```

Write a class template `array` that contains elements of type `T` and that allows the following behaviors:

- Constructor with no parameters. Creates an empty array.
- Construction with an `unsigned int n` as parameter. This creates an array of n elements, initialized by default.
- Construction by copy and assignment operator. In both cases, modifying one of the two arrays after copy/assignment won't affect anything in the other array.
- Elements are accessible through the `operator[]` method. This allows the following behaviors:

```
1 array<int> a(1);
2   a[0] = 42;
3   const array<int> b = a;
4   std::cout << b[0];
```

  Think about the consequences of the `const`, as well as potential overloads.

Some additional guidelines:

- When accessing an element with the operator [], if this element is out of the limits, the array is resized. If the array is `const` and that consequently the element can't be created, a`std::exception` is thrown.
- The class possesses a member function `size()` that returns the number of elements in the array, without changing it.
- You MUST use the operator `new[]` for your allocation. You must not do preventive allocation. The memory management must be as accurate as possible. The code must never access the non allocated memory. These particular instructions will be tested!
- The class `array` must possess a member function `dump()` which displays the content of the l'`array`, using the format presented in the example.
- The array class must possess a member function `convertTo`. This one is templated on a type `U`. It returns a `array<U>` of the same size that the original one, without changing the original one. Each element of the new array is a conversion of the matching element from the original array. The conversion is realized from a function passed as parameter to the member function `convertTo`. You must deduce the exact prototype of convertTo from the example. The member function `convertTo` is in charge of the array as a one time operation.
- An array of bools has a particular behavior. Its output when the member function `dump()` is called, displays `[true, false]` instead of `[1, 0]`.

Example :

```
int float_to_int(float const& f) {
  return static_cast<int>(f);
}

array<int> a(4);
a[3] = 1;
const array<int> b = a;
b.dump();
array<float> c;
c.dump();
c[2] = 1.1;
c.dump();
a = c.convertTo<int>(&float_to_int);
a.dump();
```

displays :

```
(koala@arbre) ./a.out | cat -e
[0, 0, 0, 1]$
[]$
[0, 0, 1.1]$
[0, 0, 1]$
```

Chapter VIII

Exercise 6

	Exercise : 06	points : 5
Tuples		
Turn-in directory: `(cpp_d15)/ex06`		
Compiler: `g++`	Compilation flags: `-W -Wall -Werror -std=c++03`	
Makefile: `No`	Rules: `n/a`	
Files to turn in : `ex06.hpp`		
Remarks : `n/a`		
Forbidden functions : `None`		

```
It is very rare and always accidental that human waste are disposed
outside of the plane during a flight. Airplanes are equipped with
collection tanks that are emptied on the ground by specials
vehicles.

The toilet flushing process of airplanes use an aspiration mechanism
that not only saves water, but also ensures a proper fluid flow.
Indeed, gravity is not enough in a flying plane to ensure a correct
flow within the pipes.

Trains, on their side, releases toilets waste on the rails. This is why it is
forbidden to use toilets when a train is at stop in a station. However,
recent trains are equipped with tanks, just like airplanes.
```

If you don't want to be a human waste collector for the Roissy airport, you must code better and faster.

How many times did you wished that your functions returned not one but two values?

Your dream will come true thanks to this exercise. You must create a structure template Tuple with the following use:

```
1       Tuple<int, std::string> t;
2       t.a = 42;
3       t.b = std::string("Boeuf aux oignons");
```

This time, a and b are public.

If the second parameter is not provided, then it will be identical to the first one:

```
1       Tuple<int> t;
2       t.a = 42;
3       t.b = 21;
```

Your Tuple must provide a member function `toString`, that returns a `std::string` and works as described in the following example. This member function does not modify the instance.

```
1       Tuple<int, std::string> t;
2       t.a = 42;
3       t.b = std::string("Boeuf aux oignons");
4       std::cout << t.toString() << std::endl;
```

The expected output is:

```
1       (koala@arbre)$ ./a.out | cat -e
2       [TUPLE [int:42] [string:"Boeuf aux oignons"]]$
```

You must manage the display for ints, floats and `std::string`s as described below:

- For a `int` : `[int:42]`
- For a `float` : `[float:3.4f]` (use the default display of a `float` with a `ostream`, without taking care of the precision, and don't forget the 'f')
- For a `std::string` : `[string:''test'']`
- For anything else : `[???]`

Thus the following code:

```
1       Tuple<float, char> t;
2       t.a = 1.1f;
3       t.b = 'x';
4       std::cout << t.toString() << std::endl;
```

Generates the following output:

```
1        (koala@arbre)$ ./a.out | cat -e
2         [TUPLE [float:1.1f] [???]]$
```

Chapter IX

Epilogue

```
Canting arms are heraldic bearings that represent the bearer's
name in a visual pun or rebus. The term cant came into the English
language from Anglo-Norman cant, meaning song or singing, from
the Latin cantare and English cognates include canticle, chant, accent,
incantation and recant.

A famous examples of canting arms are those of the late Queen Elizabeth,
the Queen Mother, who was born Elizabeth Bowes-Lyon. Her arms contain
in sinister (i.e. on the bearer's left, viewer's right) the bows and blue lio
that make up the arms of the Bowes and Lyon families.
```

Bonus exercise: Create heraldic bearings that represents the various Epitech communities: Koalas, Asteks, Bocal, Admin as well as the various laboratories. Your arms can be Canting arms or not. Blazoning is expected, a drawing would be appreciated.