B3 - C++ Pool

B-PAV-242

Day 15

Templates

42.0

Day 15

binary name:	no binary
group size:	1
repository name:	cpp_d15
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,...).

General Setpoints

READ THESE CAREFULLY
You will have no possible excuse if you end up with a 0 because you didn't follow one of these.

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.

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

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

Remember: 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 questions asked:

*alloc
*printf
free

Any use of the `friend` keyword will result in a -42

You are not allowed to use any library other than the C++ standard library.

It must be possible to include each of your header files independently from the others. Headers must include all their dependencies.

All your header files will be included in the correction `main`.

None of your files must contain a `main` function

THINK. Please.

THINK

T.H.I.N.K.! For Pony!

To avoid compilation problems during automated tests, please include all necessary files within your headers.

Please note that 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.

This subject may be modified up to one hour before turn-in time!

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 classes you turn in, create a file in that directory named `tests-CLASS-NAME.cpp` containing all the tests needed to cover all of the class' 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 - Old friends

	Exercise: 00	points : 2
Swap, min, max, add		
Turn-in directory: cpp_d15/ex00		
Compiler: g++	Compilation flags: -W -Wall -Wextra -Werror -std=c++14	
Makefile: No	Rules: n/a	
Files to turn in: ex00.hpp		
Notes: None		
Forbidden functions: ‘using namespace‘ keyword		

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:

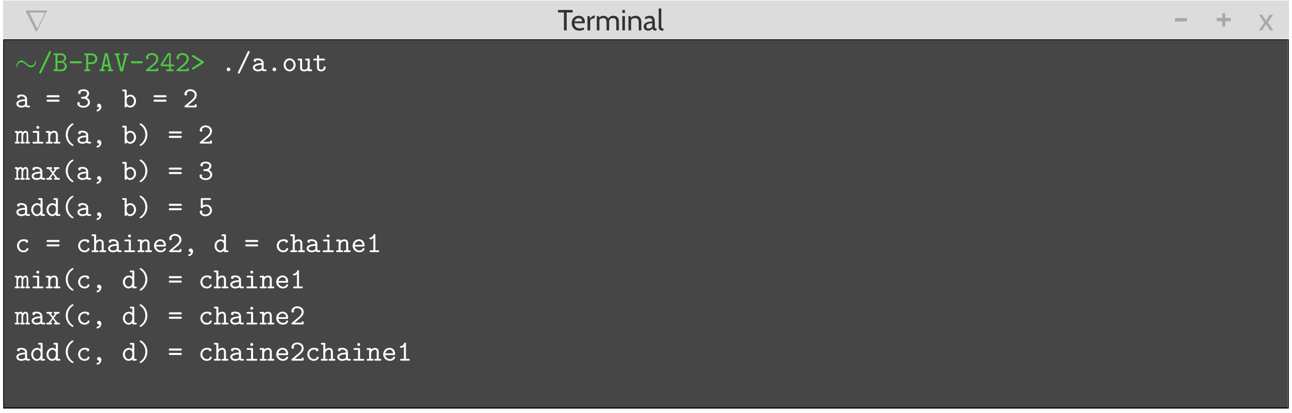
- swap: swaps the value of its two parameters. Does not return anything.
- min: returns the smallest of its two parameters. If the two parameters are equal, returns the second.
- max: returns the biggest of its two parameters. If the two parameters are equal, returns the second.
- add: returns the result of the addition of its two parameters.

These templates should generate functions that can be called with any type of parameter, as long as they have the same type and support all comparison operators.

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

```
#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;
}
```

main.cpp



```
~/B-PAV-242> ./a.out
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
```

Exercise 1 - Compare

	Exercise: 01	points : 2
Compare		
Turn-in directory: cpp_d15/ex01		
Compiler: g++	Compilation flags: -W -Wall -Wextra -Werror -std=c++14	
Makefile: No	Rules: n/a	
Files to turn in: ex01.hpp		
Notes: None		
Forbidden functions: ‘using namespace‘ keyword		

Emmanuelle's recipe of the **Petits Lu** cookies, taken from **Trish Deseine**'s excellent book, *"Je veux du Chocolat"*:

```
 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.
```

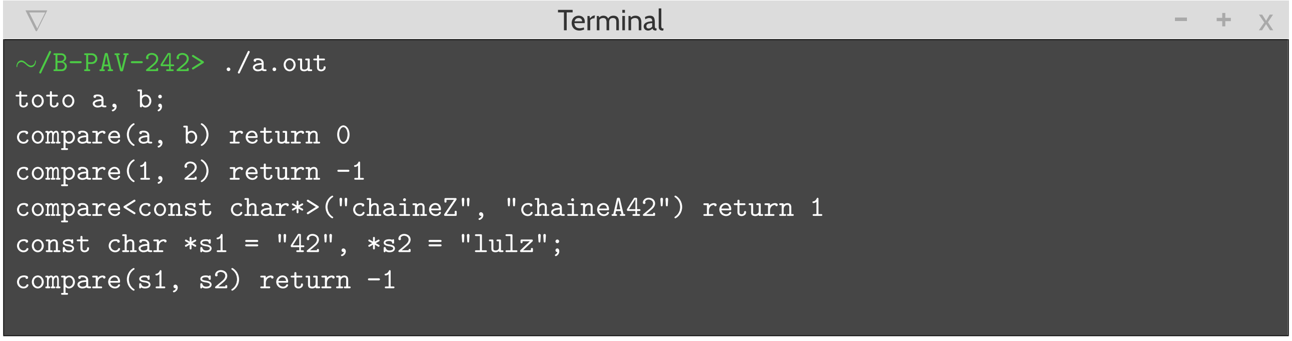
Use those 6 hours to write the `compare` function template. It takes two parameters of the same type, and returns an int equal to:

- 0 if the two parameters are equal
- -1 if the first parameter is smaller than the second
- 1 if the first parameter is bigger than the second

You must also provide an overload of this function for `const char *` parameters.

Here is a sample class and its expected output:

```
class toto
{
  public:
    toto() = default;

  public:
    toto &operator=(const toto&) = delete;
    toto(const toto &) = delete;

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



```
~/B-PAV-242> ./a.out
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 might think, *"chainZ"* and *"chainA42"* are not `const char *` but respectively a `const char[8]` and a `const char[10]`. To use the right overload, you will have to either `static_cast` them into `const char *` are explicitly call `compare<const char *>` as shown in this example.

Exercise 2 - The return of min

Exercise: 02	points : 3
The return of 'min'	
Turn-in directory: cpp_d15/ex02	
Compiler: g++	Compilation flags: -W -Wall -Wextra -Werror -std=c++14
Makefile: No	Rules: n/a
Files to turn in: ex02.hpp	
Notes: None	
Forbidden functions: 'using namespace' keyword	

```
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.
```

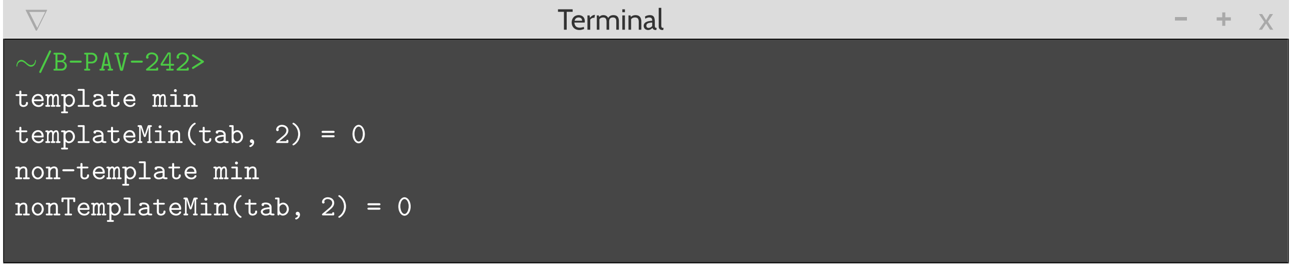
If you ever want to satisfy your reproductive instincts, you need to become **rich**. Obviously, you picked computer science to fulfill your ambitions. The first step towards becoming rich, for you, is to implement the following functions:

- A `min` function template taking two parameters of the same type and returning the smallest of the two. It must print *"template min"* (without the quotes). If the two parameters are equal, the function returns the first one.
- A `min` function taking two integers as parameters and returning the smallest of the two. It must print *"non-template min"* (without the quotes). If the two parameters are equal, the function returns the first one.
- A `templateMin` function template taking an array and its size as parameters and returning the smallest value in the array. This function must call the `min` function template.
- A `nonTemplateMin` function taking an array of integers and its size as parameters and returning the smallest value in the array. This function must call the `min` function.

If you did everything right, the following code should produce the associated output:

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

```
    minimum = nonTemplateMin(tab, 2);
    cout << "nonTemplateMin(tab,␣2)␣=␣" << minimum << endl;
```



```
~/B-PAV-242>
template min
templateMin(tab, 2) = 0
non-template min
nonTemplateMin(tab, 2) = 0
```

EXERCISE 3 - ANOTHER ITERATION

KOALA	Exercise: 03	points : 4
'foreach'		
Turn-in directory: `cpp_d15/ex03`		
Compiler: `g++`	Compilation flags: `-W -Wall -Wextra -Werror -std=c++14`	
Makefile: `No`	Rules: `n/a`	
Files to turn in: `ex03.hpp`		
Notes: `None`		
Forbidden functions: `'using namespace' keyword`		

> 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.

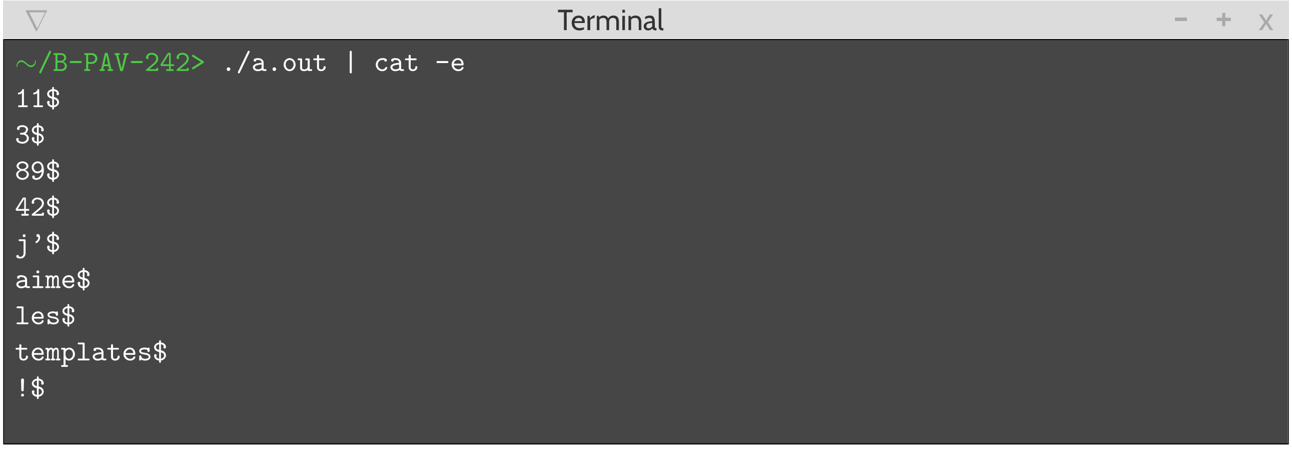
To avoid a fate similar to **Theodore**'s, you must implement the `foreach` function template. It will let you walk over an array and call a function for each element in it. It takes as parameters the array's base address, a reference to a function, and the size of the array. The function reference must match the following prototype: `void func(const T &elem)`.

Moreover, you must provide the `print` function template that can be passed to `foreach` and displays each element on a separate line.

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

```cpp
#include "ex03.hpp"

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

main.cpp



```
~/B-PAV-242> ./a.out | cat -e
11$
3$
89$
42$
j'$
aime$
les$
templates$
!$
```

Exercise 4 - Testing

	Exercise: 04	points : 4
Test		
Turn-in directory: cpp_d15/ex04		
Compiler: g++	Compilation flags: -W -Wall -Wextra -Werror -std=c++14	
Makefile: No	Rules: n/a	
Files to turn in: ex04.cpp, ex04.hpp		
Notes: None		
Forbidden functions: ‘using namespace‘ keyword		

```
 In 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
eidelberg University. Nanotubes have been identified six year later from
a synthesis by-product of the fullrenes.
```

That's it for physics today and, fortunately, the upcoming exercise is much easier than creating fullrenes.

Write the `equal` function template that takes as parameters two constant references to a given type. It must return `true` if the two parameters are equal, and `false` otherwise.

Write a `Tester` class template, taking a single type parameter called `T`. It must provide an `equal` member function taking as parameter two constant references to `T`. This function must return `true` if the two parameters are equal, and `false` otherwise.

Don't worry about making `Tester` canonical.

You must only put your declarations in the `hpp` file. You are not allowed to pt the body of the function nor the member function outside of the `cpp` file you turn in.

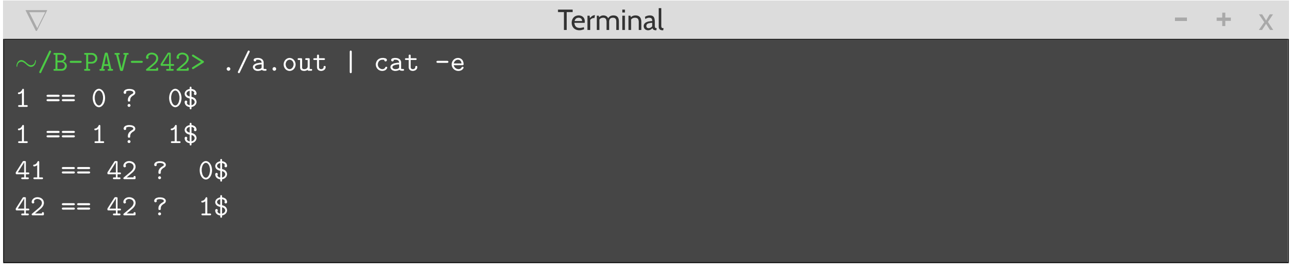
Instantiate your function and class templates for the following types:

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

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

```
#include <iostream>
#include "ex04.hpp"

int main()
{
    std::cout << "1␣==␣0␣?␣" << equal(1, 0) << std::endl;
    std::cout << "1␣==␣1␣?␣" << equal(1, 1) << std::endl;

    Tester<int> iT;

    std::cout << "41␣==␣42␣?␣" << iT.equal(41, 42) << std::endl;
    std::cout << "42␣==␣42␣?␣" << iT.equal(42, 42) << std::endl;
    return 0;
}
```

main.cpp



```
~/B-PAV-242> ./a.out | cat -e
1 == 0 ?  0$
1 == 1 ?  1$
41 == 42 ?  0$
42 == 42 ?  1$
```

EXERCISE 5 - ARRAYS

Exercise: 05	points : 4
‘array<T>‘	
Turn-in directory: `cpp_d15/ex05`	
Compiler: `g++`	Compilation flags: `-W -Wall -Wextra -Werror -std=c++14`
Makefile: `No`	Rules: `n/a`
Files to turn in: `ex05.hpp`	
Notes: `None`	
Forbidden functions: `‘using namespace‘ keyword`	

```
  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
information 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 an `array` class template that contains elements of type `T` and has:

- A default constructor that creates an empty array
- A constructor taking an `unsigned int n` as parameter that creates an array of `n` default-constructed elements
- A copy constructor and assignment operator
- An overload of the square-brackets (`[]`) operator that makes the following possible:

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

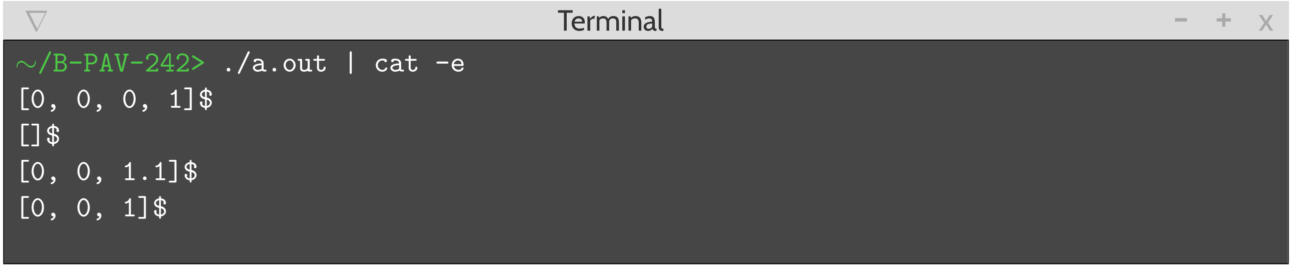
Some additional guidelines:

- When accessing an out-of-bounds element with the square-bracket operator, the array must be resized. If the array is `const` and can therefore not be resized, an `std::exception` must be thrown.
- The class must have a `size()` member function returning the number of elements in the array
- You **MUST** use `operator new[]` to allocate memory. You must not perform any preventive allocation. Memory management must be as accurate as possible

- The class must have a `dump()` member function that displays the content of the `array` using the format found in the sample output
- The class must have a `convertTo` member function template, taking a `U` type parameter. This function returns an `array<U>` with the same size as the original, with each element being a conversion of the matching element from the original array. The conversino is done through a function passed as parameter to `convertTo`. You must deduce the exact prototype of `convertTo` based on the sample code.
- An array of `bool`s is a special case: its output when calling `dump` must be `[true, false]` instead of `[1, 0]`.

Here is a sample piece of code and its expected output:

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

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



```
∼/B-PAV-242> ./a.out | cat -e
[0, 0, 0, 1]$
[]$
[0, 0, 1.1]$
[0, 0, 1]$
```

Exercise 6 - Tuples

Exercise: 06	points : 5
Tuples	
Turn-in directory: cpp_d15/ex06	
Compiler: g++	Compilation flags: -W -Wall -Wextra -Werror -std=c++14
Makefile: No	Rules: n/a
Files to turn in: ex06.hpp	
Notes: None	
Forbidden functions: ‘using namespace‘ keyword	

> 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 often have you wished that your functions could return not just one, but two values?

Your dream will come true thanks to this exercicse. You must create a `Tuple` structure template which allows for the following:

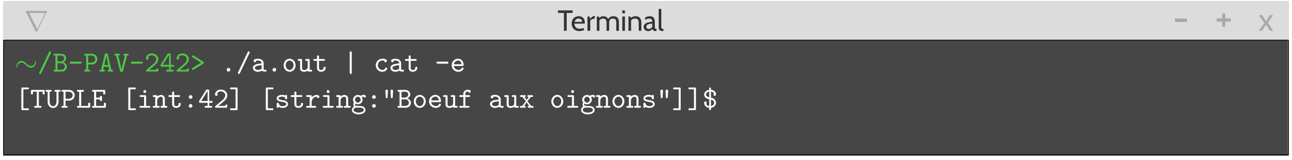
```
Tuple<int, std::string> t;
t.a = 42;
t.b = std::string("Boeuf␣aux␣oignons");
```

If the second type parameter is not provided, it must be identical to the first:

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

Your `Tuple` must provide a `toString` member function returning an `std::string` that matches the following example:

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



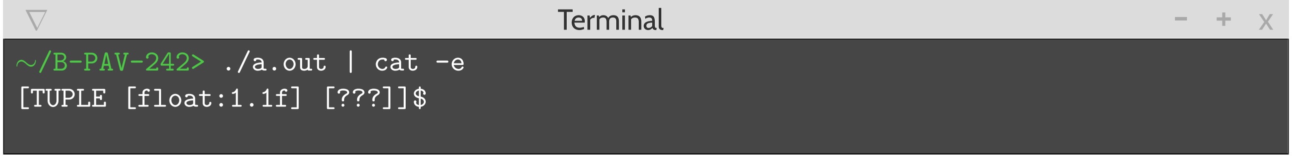
```
∼/B-PAV-242> ./a.out | cat -e
[TUPLE [int:42] [string:"Boeuf aux oignons"]]$
```

This function must be able to print `int`s, `float`s and `std::string`s as described below:

- For an `int`: *[int:42]*
- For a `float`: *[float:3.4f]* (use the default output of `float`s with `ostream`s, without attending to the precision)
- For an `std::string`: *[string:"test"]*
- For anything else: *[???]*

Thus, the following code will produce the following output:

```
Tuple<float, char> t;
t.a = 1.1f;
t.b = ’x’;
std::cout << t.toString() << std::endl;
```



```
∼/B-PAV-242> ./a.out | cat -e
[TUPLE [float:1.1f] [???]]$
```

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 lions that make up the arms of the Bowes and Lyon families.