



# Piscine C++ - d15

## Templates

Uriel "Korfuri" Corfa corfa_u@epitech.eu

*Abstract: Ce document est le sujet du d15 : Templates*

# Table des matières

# Chapitre I

# REMARQUES GÉNÉRALES

- LISEZ LES REMARQUES GÉNÉRALES ATTENTIVEMENT !!!!!
  - Vous n'aurez aucune excuse si vous avez 0 parce que vous avez oublié une consigne générale ...

- REMARQUES GÉNÉRALES :
  - Nous parlons ici de templates. Vous pouvez donc implémenter dans des headers ce qui doit logiquement être implémenté dans un header.
  - Toutes les sorties se font sur la sortie standard et sont terminées par un retour à la ligne sauf si le contraire est precisé explicitement.
  - Les noms de fichiers qui vous sont imposés doivent être respectés À LA LETTRE, de même que les noms de classes et de fonctions membres / méthodes.
  - Rappelez-vous que vous faites du C++ et non plus du C. Par conséquent, les fonctions suivantes sont INTERDITES, et leur utilisation sera sanctionnée par un -42 :
    - `*alloc`
    - `*printf`
    - `free`
    - `open`, `fopen`, etc...
  - De facon générale, les fichers associés à une classe seront toujours `NOM_DE_LA_CLASSE.hh` et `NOM_DE_LA_CLASSE.cpp` (s'il y a lieu).
  - Les répertoires de rendus sont ex00, ex01, ..., exN
  - Toute utilisation de `friend` ou de `dynamic_cast` se soldera par un -42, no questions asked.

- Certains exercices sont corrigés à l'aide de debogueurs memoire, vérifiant ainsi que vos allocations sont conformes à ce qui est attendu. N'espérez pas reussir les exos à l'aide d'un int tab[100000] quand nous attendons un new int[5] au bon moment.
- Lisez attentivement les exemples, ils peuvent requérir des choses que le sujet ne dit pas...
- Lisez ENTIÈREMENT le sujet d'un exercice avant de le commencer !
- RÉFLECHISSEZ. Par pitié.

- COMPILATION DES EXERCICES :
  - La moulinette compile votre code avec les flags : `-W -Wall -Werror`
  - Pour éviter les problèmes de compilation de la moulinette, incluez les fichiers nécéssaires dans vos fichiers include (*.hh).
  - Notez bien qu'aucun de vos fichiers ne doit contenir de fonction `main`. Nous utiliserons notre propre fonction `main` pour compiler et tester votre code.
  - Pour éviter les problèmes de compilation de la moulinette, incluez les fichiers nécessaires dans vos fichiers include (*.hpp).
  - Le repértoire de rendu est : `(DÉPOT SVN - piscine_cpp_d15-promo-login_x)/exN` (N étant bien sur le numéro de l'exercice).

# Chapitre II

# Exercice 0

| | Exercice : 00 | points : 2 |
|---|---|---|
| Swap, min, max, add | | |
| Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d15-promo-login_x)/ex00 | | |
| Compilateur : g++ | Flags de compilation: -W -Wall -Werror | |
| Makefile : Non | Règles : n/a | |
| Fichiers a rendre : ex00.hpp | | |
| Remarques : n/a | | |
| Fonctions Interdites : *alloc - free - *printf | | |

L'ornithorynque :

> L'ornithorynque est un petit mammifere semi-aquatique, qu'on ne trouve qu'a l'est de l'Australie. C'est l'un des quatre monotremes, les seuls mammiferes qui pondent des oeufs au lieu de donner naissance a des petits vivants (les trois autres sont des achidnes).
>
> L'ornithorynque (Ornithorhynchus anatinus) ressemble a un castor: le corps et la queue, larges et plats, couverts de fourrure marron, mais il est pourvu de pieds palmes et d'un grand museau caoutchouteux qui l'a fait designer en anglais par « duck-billed platypus » (« pied plat bec de canard »). Sa queue mesure de 10 a 15 cm. Les males sont habituellement d'un tiers plus gros que les femelles, mais leur taille, entre 40 et 50 cm en moyenne, varie considerablement d'une region a l'autre, sans qu'elle soit liee au climat.

Cet exercice ne traite pas des ornithorynques.

Voud devez en revanche écrire les fonctions template suivantes :

- `swap` : échange les valeurs des deux arguments, ne retourne rien

- `min` : compare les deux arguments, renvoie la valeur la plus petite des deux. Si elles sont égales, renvoyer la seconde.
- `max` : compare les deux arguments, renvoie la valeur la plus grande des deux. Si elles sont égales, renvoyer la seconde.
- `add` : ajoute les deux arguments, renvoie le resultat de l'opération. Le retour se fera par copie et non par référence.

Ces fonctions devront pouvoir être appelées avec tous types d'arguments, à condition que les deux arguments soient du même type et supportent tous les opérateurs de comparaison.

Le code suivant devra compiler :

```
#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;
}
```

La sortie devra être :

```
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
```

# Chapitre III

# Exercice 1

| | Exercice : 01 | points : 2 |
|---|---|---|
| Compare | | |
| Répertoire de rendu: `(DÉPOT SVN - piscine_cpp_d15-promo-login_x)/ex01` | | |
| Compilateur : `g++` | Flags de compilation: `-Wextra -Werror -Wall` | |
| Makefile : `Non` | Règles : `n/a` | |
| Fichiers a rendre : `ex01.hpp` | | |
| Remarques : `n/a` | | |
| Fonctions Interdites : `*alloc - free - *printf` | | |

Recette du gateau aux petits Lu d'Emmanuelle, tiré de l'excellent livre "Je veux du Chocolat" de Trish Deseine :

> `Casser en gros morceaux 3/4 d'un paquet de petits Lu avec 40g de meringue.`
>
> `Faire fondre 100g de chocolat avec 300g de beurre, laisser un peu refroidir, ajouter 2 oeufs légèrement battus, 150g de sucre et 50g de cacao. Verser sur les morceaux de Lu et de meringues, bien mélanger et mettre dans un moule à cake en silicone ou filmé. Tasser un peu et mettre au frais au moins 6 heures.`

Profitez de ces 6 heures pour écrire la fonction template `compare`.

Cette fonction prend en argument 2 paramètres de meme type et retourne un int correspondant à :

- 0 : les deux paramètres sont égaux
- -1 : le premier paramètre est inférieur au second
- 1 : le premier paramètre est supérieur au second

Cette fonction devra fonctionner correctement pour des `const char *` (en appelant explicitement la bonne surcharge).

Votre template de fonction doit compiler avec le code suivant :

```
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(){}
};
```

La sortie doit etre :

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

Contrairement à ce qu'on pourrait croire, "chaineZ" et "chaineA42" ne sont pas des const char* mais, respectivement, un const char[8] et un const char[10]. Pour utiliser la bonne surcharge, il faut donc soit static_caster, soit appeler explicitement compare<const char*> comme dans cet exemple.

# Chapitre IV

# Exercice 2

|  | Exercice : 02 | points : 3 |
|---|---|---|
| min, le retour | | |
| Répertoire de rendu: `(DÉPOT SVN - piscine_cpp_d15-promo-login_x)/ex02` | | |
| Compilateur : `g++` | Flags de compilation: `-Wextra -Werror -Wall` | |
| Makefile : `Non` | Règles : `n/a` | |
| Fichiers a rendre : `ex02.hpp` | | |
| Remarques : `n/a` | | |
| Fonctions Interdites : `Aucune` | | |

```
L'acide désoxyribonucléique (ADN) est le support de
 l'information génétique de tous les êtres vivants. Chez les
 eucaryotes, l'ADN est localisé dans le noyau de chaque cellule
 de l'organisme. Chez les procaryotes, la molécule d'ADN est
 libre dans le cytoplasme des cellules.

 L'ADN est constitué de deux chaînes (ou brins) enroulées l'une
 autour de l'autre en une double hélice de 2 nm (10^-9m) de
 diamètre. Chaque chaîne est composée d'une succession de
 nucléotides qui sont un assemblage de trois molécules : un
 groupement phosphate, un sucre (désoxyribose) et une base
 azotée. Les bases azotées sont au nombre de quatre : adénine
 (notée A), thymine (T), guanine (G), cytosine (C). La
 complémentarité de ces bases, deux à deux, permet
 l'association des deux brins d'ADN : l'adénine est toujours
 appariée à une thymine et la guanine à une cytosine.
```

Afin de satisfaire votre instinct reproducteur, vous devez devenir riche, et vous avez choisi l'informatique dans cette optique. Afin de parvenir à ces fins, vous devez écrire les fonctions suivantes :

- Une fonction template `min` prenant deux paramètres du meme type et retournant le plus petit des deux. Cette fonction doit afficher “template min” sur la sortie standard (sans quotes). Si les deux paramètres sont égaux, renvoyer le premier.
- Une fonction `min` prenant deux entiers en paramètres en retournant le plus petit des deux. Cette fonction doit afficher “non-template min” sur la sortie standard (sans les “). Si les deux paramètres sont égaux, renvoyer le premier.
- Une fonction template `templateMin` prenant un tableau et sa taille en paramètre et retournant la plus petite valeur contenue dans ce tableau. Cette fonction doit appeler la fonction template `min`.
- Une fonction `nonTemplateMin` prenant un tableau d’entiers et sa taille en paramètres et retournant la plus petite valeur de ce tableau. Cette fonction doit appeler la fonction `min` non template.

Ainsi, le code suivant, s'il est correctement utilisé :

```
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;
```

affichera :

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

# Chapitre V

# Exercice 3

| | Exercice : 03 | points : 4 |
|---|---|---|
| foreach | | |
| Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d15-promo-login_x)/ex03 | | |
| Compilateur : g++ | Flags de compilation: -Wextra -Werror -Wall | |
| Makefile : Non | Règles : n/a | |
| Fichiers a rendre : ex03.hpp | | |
| Remarques : n/a | | |
| Fonctions Interdites : Aucune | | |

```
En 1221, eut lieu la prise de Sérès par le despote d'Épire,
 Théodore Ange, qui s'empara ainsi du royaume de Thessalonique.
 Il poursuivit l'expansion de son royaume les années suivantes,
 particulièrement en 1225, avec la prise de Corfou et de
 Durazzo aux Vénitiens, d'Andrinople aux Byzantins de Nicée, et
 de Xantheia et Didymotique aux Latins.

 L'année 1230 marqua le couronnement de Théodore Ange à
 Thessalonique, mais aussi sa défaite contre le royaume valaque
 de Bulgarie, lors de la bataille de Cloconita. Il fut fait
 prisonnier par le tsar Ioan Asan II qui lui fit crever les
 yeux.
```

Pour ne pas subir le même sort que ce malheureux, vous devez écrire la fonction template `foreach`.

Cette fonction permet de parcourir un tableau en appelant une fonction pour chacun des éléments de ce tableau. Elle prend en argument l'adresse de debut du tableau, une référence sur fonction et la taille du tableau. La référence sur fonction correspond au prototype suivant : `void func(const type& elem);`

De plus, vous devrez fournir la fonction `print` qui sera passée à la fonction `foreach` et affichera chaque élément à raison d'un élément par ligne, quelque soit leur type.

On devra pouvoir utiliser votre rendu comme suit :

```
#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;
}
```

Ce qui affichera :

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

# Chapitre VI

# Exercice 4

| | Exercice : 04 | points : 4 |
|---|---|---|
| Tester | | |
| Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d15-promo-login_x)/ex04 | | |
| Compilateur : g++ | Flags de compilation: -Wextra -Werror -Wall | |
| Makefile : Non | Règles : n/a | |
| Fichiers a rendre : ex04.cpp, ex04.hpp | | |
| Remarques : n/a | | |
| Fonctions Interdites : Aucune | | |

En 1985, trois chercheurs, Richard Smalley, Robert F. Curl (de l'Université Rice de Houston) et Harold W. Kroto (Université de Sussex) découvraient une nouvelle forme allotropique du carbone, la molécule C60 constituée de 60 atomes de carbone répartis sur les sommets d'un polyèdre régulier formé de facettes hexagonales et pentagonales. Chaque atome de carbone a une liaison avec trois autres. Cette forme est connue sous le nom de Buckminsterfullerene ou Buckyball et doit son nom à l'architecte et inventeur américain Richard Buckminster Fuller qui a créé plusieurs dômes géodésiques dont la forme est analogue au C60.

Plus généralement, les fullerènes dont fait partie le C60, sont une nouvelle famille de composés du carbone. Non équilatéraux, leur surface se compose d'une combinaison d'hexagones et de pentagones à l'instar des facettes d'un ballon de football. Cette disposition leur confère des structures toujours fermées en forme de cage de carbone. Il fallut néanmoins attendre 1990, pour que Huffman et Kramer de l'Université de Heidelberg, mettent au point un procédé de synthèse permettant l'obtention de ces molécules en quantités

```
macroscopiques. Les nanotubes ont été identifiés six années
plus tard dans un sous-produit de synthèse des fullerènes.
```

Nous ne ferons pas de physique aujourd'hui, et fort heureusement, l'exercice qui vient est plus trivial que la fabrication de fullerènes.

Écrire la fonction `equal` templatée sur un type `T` et prenant deux paramètres de type référence sur `const T`. Cette fonction devra renvoyer `true` si les deux paramètres sont égaux, `false` dans le cas contraire.

Écrire une classe `Tester` templatée sur un type `T`. Cette classe fournira une fonction membre `equal` prennant deux paramètres de type référence sur `const T`. Cette fonction membre devra renvoyer `true` si les deux paramètres sont egaux, `false` dans le cas contraire.

Vous n'avez pas à vous soucier de rendre la classe `Tester` canonique.

Vous ne devrez mettre que les déclarations dans le .hpp. Vous n'avez pas le droit de mettre le corps de la fonction ainsi que de la fonction membre en dehors du fichier .cpp à rendre.

Vous instancierez vos templates de fonction et de classe pour les types suivants :

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

Le code suivant :

```
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 }
```

affichera :

```
1       (pouill_g@tatooine 497)./ex04 | cat -e
2       1 == 0 ? 0$
3       1 == 1 ? 1$
4       41 == 42 ? 0$
5       42 == 42 ? 1$
```

# Chapitre VII

# Exercice 5

| | Exercice : 05 | points : 4 |
|---|---|---|
| array<T> | | |
| Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d15-promo-login_x)/ex05 | | |
| Compilateur : g++ | Flags de compilation: -Wextra -Werror -Wall | |
| Makefile : Non | Règles : n/a | |
| Fichiers a rendre : ex05.hpp | | |
| Remarques : n/a | | |
| Fonctions Interdites : Aucune | | |

L'Inde compte plus d'un milliard d'habitants (2000). Cela fait de l'Inde le deuxième pays au monde le plus peuplé, après la Chine.

Toutefois, alors que cette dernière est à peu près parvenue à maîtriser sa croissance démographique, l'Inde connaît toujours une augmentation rapide de sa population. La population indienne augmente d'environ 19 millions d'habitants par an (conséquence d'un taux global de fécondité de 2,7 enfants par femme - contre 1,7 pour la Chine). On peut ainsi s'attendre à ce que l'Inde devienne le pays le plus peuplé du monde aux alentours de 2035.

Les démographes sont ainsi alarmés moins par les chiffres eux-mêmes (la fécondité indienne s'est effondrée en 50 ans) que par la tendance irrégulière et relativement lente. Cela est attribué à une politique démographique à la fois brutale et incohérente (là où la Chine a misé sur la politique - simpliste et brutale parfois, mais applicable - de l'enfant unique). L'Inde a aussi plus axé sa politique sur une responsabilisation individuelle (centres d'information sur la contraception). De plus, l'Inde étant une démocratie, cette

```
politique a eu des hauts et des bas, contrairement à la Chine,
où la politique de l'enfant unique n'a pas varié depuis sa
mise en vigueur (avec seulement des adoucissements pour les
ruraux).
```

Ecrivez une classe template `array`. Elle contiendra des éléments d'un type `T`. Elle permettra les comportements suivants :

- Construction sans paramètres. Ceci crée un array vide.
- Construction avec un `unsigned int n` en paramètre. Ceci crée un array de n éléments, initialisés par défaut.
- Construction par copie et opérateur d'assignation. Dans les deux cas, modifier un des deux arrays apres la copie/assignation n'affectera nullement l'autre array.
- Les éléments seront accessibles par un `operator[]`. Ceci doit permettre les comportements suivants :

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

Pensez aux conséquences du `const`, et à d'éventuelles surcharges.

Quelques consignes supplémentaires :

- Lors de l'accès à un élément avec l'opérateur [], si cet élément est hors limite, le tableau sera redimensionné. Si l'array est `const` et que l'élément ne peut donc être créé, une `std::exception` sera lancée.
- La classe possède une fonction membre `size()` retournant le nombre d'éléments de l'array, sans modifier l'array.
- Vous DEVEZ utiliser l'operateur `new[]` pour votre allocation. Vous ne devez pas faire d'allocation préventive. Votre gestion de la mémoire doit se faire au plus juste. Votre code ne devra en aucun cas accéder à de la mémoire non-allouée. Tous ces points seront testés.
- La classe `array` doit posséder une fonction membre `dump()` affichant le contenu de l'`array`, au format indiqué dans l'exemple.
- La classe array doit posséder une fonction membre `convertTo`. Celle-ci est templatée sur un type `U`. Elle renvoie un `array<U>` de la même taille que l'original, sans modifier l'original. Chaque élément du nouvel array est la conversion de l'élément correspondant de l'array d'origine. La conversion est réalisée à partir d'une fonction passée en paramètre à la fonction membre `convertTo`. Vous devez déduire le prototype exact de convertTo à partir de l'exemple. La fonction membre `convertTo` effectue l'allocation du nouvel array en une fois.

- Un array de bools a un comportement particulier. Sa sortie lors de l'appel de la fonction membre `dump()` affiche `[true, false]` au lieu de `[1, 0]`.

Exemple :

```
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();
```

Affiche :

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

# Chapitre VIII

# Exercice 6

| | Exercice : 06 | points : 5 |
|---|---|---|
| Tuples | | |
| Répertoire de rendu: `(DÉPOT SVN - piscine_cpp_d15-promo-login_x)/ex06` | | |
| Compilateur : `g++` | Flags de compilation: `-Wextra -Werror -Wall` | |
| Makefile : `Non` | Règles : `n/a` | |
| Fichiers a rendre : `ex06.hpp` | | |
| Remarques : `n/a` | | |
| Fonctions Interdites : `Aucune` | | |

> Il est extrêmement rare, et toujours accidentel, que les déjections humaines des avions soient jetées en plein air. Les aéroplanes sont équipés de réservoirs de collecte, vidés lors des arrêts au sol par des véhicules spécialisés.
>
> Le processus de chasse des toilettes d'avion utilise une aspiration, permettant à la fois d'économiser de l'eau, et de s'assurer du bon écoulement des fluides. En effet, dans un avion en mouvement, la gravité ne suffit pas à assurer le bon écoulement dans les tuyaux.
>
> Les trains, quant à eux, écoulent généralement les rejets des toilettes sur les rails. Ce fait est à l'origine de l'interdiction d'utiliser les toilettes durant les arrêts en gare. Cependant les modèles plus récents de trains sont équipés de réservoirs de collecte à l'instar des avions.

Vous ne souhaitez pas finir collecteur d'excréments à l'aéroport de Roissy. Vous devez donc programmer plus vite et mieux.

Combien de fois n'avez vous pas voulu retourner non pas une mais deux valeurs au sein d'une fonction ?

Vous allez pouvoir réaliser ce doux rêve grâce à cet exercice. Vous devez créer la structure template Tuple, permettant l'utilisation suivante :

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

Pour une fois, a et b sont publics.

Si le deuxième paramètre n'est pas fourni, il sera identique au premier, automatiquement :

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

Votre Tuple doit fournir une fonction membre `toString`, retournant un `std::string` et fonctionnant comme dans l'exemple ci-dessous. Cette fonction membre ne modifie pas l'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;
```

La sortie attendue :

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

Vous devez gérer l'affichage pour les int, les floats et les `std::string` de la façon suivante :

- Pour un `int` : `[int:42]`
- Pour un `float` : `[float:3.4f]` (utilisez l'affichage par défaut d'un `float` avec un `ostream`, sans vous soucier de la précision, et n'oubliez pas le 'f')
- Pour un `std::string` : `[string:''test'']`
- Pour n'importe quoi d'autre : `[???]`

Ainsi, le code suivant :

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

Produit la sortie :

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

# Chapitre IX

# Epilogue

> En héraldique, on appelle ‘‘armes parlantes’’, ou, selon le terme plus ancien, ‘‘armes chantantes’’, les armes comportant des meubles ou des figures, qui par leur nom ou leur image, évoquent plus ou moins directement le nom (parfois la fonction) du possesseur de ces armes.
>
> Les armes de la ville de Couffé, Loire-Atlantique, en sont un bon exemple :
>
> > Taillé de sinople à une couffe d’or et de gueules à deux clefs d’or passées en sautoir, une divise en barre ondée d’argent brochant sur la partition; au chef cousu d’azur à la croix cannelée d’argent.

Exercice bonus : réalisez un armoirial représentant les diverses factions présentes au sein d’Epitech : Koalas, Asteks, Bocal, Administration, les divers laboratoires. Vos armes pourront être parlantes ou non. Le blasonnement est attendu, le dessin sera apprécié.