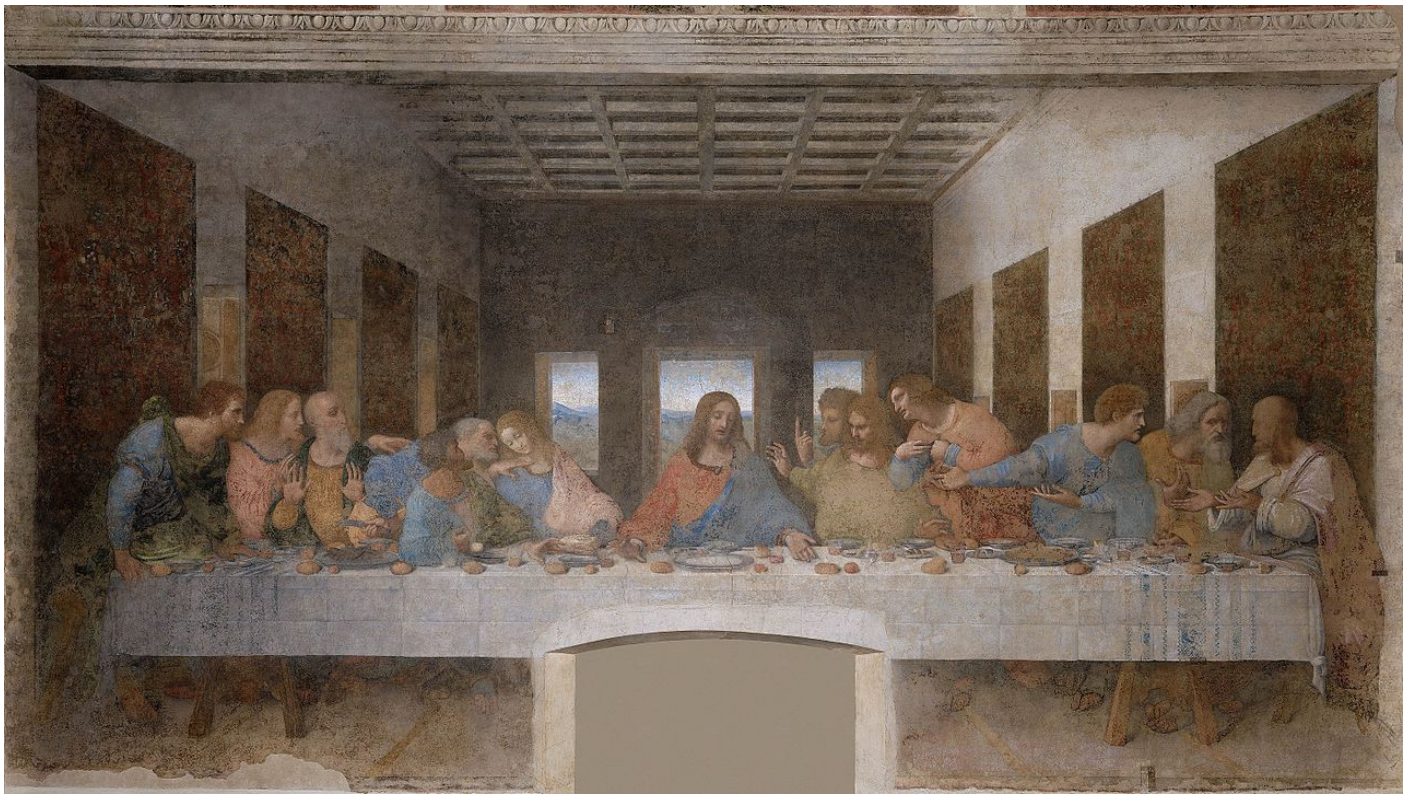
Rapport travaux dirigés 3 - 4
C. Fraisseix et T. Maquinghen



Thème : c++ et art de la Haute Renaissance

"[...]Ce programme de test créera un tableau de Personne"

- Extrait de l'exercice 1

« Ad augusta per angusta »

César Fraisseix ⊗ Tanguy Maquinghen

TD POO

Intro

Disclaimer

Toute référence biblique est issue de la page wikipédia de La Cène. Nous nous sommes servie de cette oeuvre comme contexte à l'exercice.

Exercice 1

> Voir exercice `01-Barthelemy/`.

Exercice 2

> Voir exercice `02-Jacques_leMineur/` pour une version avec la bibliothèque `string`.
>
> ---
>
> Voir exercice `02-Jacques_leMineur(bis)/` pour une version sans la bibliothèque `string`.

Exercice 3

> Voir exercice `03-Andre/`.

Exercice 4

Question 1

Le mot clef `#define` ne réserve pas de mémoire, cette instruction ne fait que remplacer toutes les occurences de `MAX1` par `100` dans le code. `MAX1` n'est donc pas une variable et n'a pas d'adresse.
`static const int` crée une variable globale constante. Elle réserve de la place dans la mémoire et on peut donc accéder à son adresse.
Donc on peut accéder à l'adresse de `MAX2` mais pas à l'adresse de `MAX1` puisque ce dernier n'est pas une variable.

> Voir exercice `04-Judas/`.

Exercice 5

Question 1

`n` est passé par référence, donc si on ne précise pas que `n` est constant, on pourrait le changer dans la fonction et donc changer la valeur de la variable passée en paramètre en dehors de la fonction. `const` empêche cela.

Question 2

`const` appliqué à une fonction membre l'empêche de modifier un attribut de l'instance.

```
cesar@jarvis:~/Documents/school/Université/td/poo/td3/5-Pierre_Simon$ g++ *.cpp
main.cpp:2:21: fatal error: Pesonne.h: No such file or directory
```

```
compilation terminated.
Personne.cpp:39:18: error: non-member function ‘void affiche()’ cannot have cv-qualifier
   void affiche() const {
                  ^
Personne.cpp: In function ‘void affiche()’:
Personne.cpp:40:5: error: invalid use of ‘this’ in non-member function
     this->age = 0;
     ^
```

> Voir exercice `05-Pierre_Simon/`.

Exercice 6

Question 1

A première vue ces trois ecritures fournissent le même résultat. Les deux dernières seront cependant constante (non-modifiable).
De plus, les 3 variables sont côte-à-côte dans la mémoire (à 32 bits d'écart soit la taille d'un entier).

```
cesar@jarvis:~/Documents/school/Université/td/poo/td3/6-Jean$ ./a.out
0xed9c20
0xed9c40
0xed9c60
```

Question 2

Résultat de l'allocation et de la désallocation des 10 pointeurs

```
cesar@jarvis:~/Documents/school/Université/td/poo/td3/6-Jean$ ./a.out
0xef0c20
0xef0c40
0xef0c60
--------
0xef1090 : 1
0xef10b0 : 2
0xef10d0 : 3
0xef10f0 : 4
0xef1110 : 5
0xef1130 : 6
0xef1150 : 7
0xef1170 : 8
0xef1190 : 9
0xef11b0 : 10
--------
désallocation
--------
0xef1090 : 0
0xef10b0 : 15667328
0xef10d0 : 15667360
0xef10f0 : 15667392
0xef1110 : 15667424
0xef1130 : 15667456
0xef1150 : 15667488
0xef1170 : 15667520
0xef1190 : 15667552
```

```
0xef11b0 : 15667584
```

On remarque que les pointeurs pointent vers les mêmes adresses, mais leurs valeurs ont été remplacées par des valeurs probablement arbitraire.

Question 3

Contrairement aux trois premiers pointeurs qui ont pour valeur l'adresse des entiers pointés, les références ont la même adresse et la même valeur que l'entier référencé.
Par conséquent, modifier la référence revient à modifier la variable initiale.

> Voir exercice `06-Jean/`.

Exercice 7

Question 1

Comme vu précédement, `#define copie1(source,dest) source=dest` est une directive de préprocesseur qui va remplacer dans le fichier .cpp toutes les occurences de `#define copie1(source,dest) par source=dest`.
Une fonction `inline` remplace l'appel à la fonction par son code en résolvant les paramètres. Contrairement à un `#define` qui va simplement remplacer le texte sans aucunes résolution des paramètres.
L'utilisation des fonctions inline permet dans certains cas d'économiser des ressources (mémoire et temps de calcul).

Question 2

La copie n'est pas réalisé dans le cas de la fonction inline puisque `source` et `dest` sont passés par valeurs, donc valable uniquement à l'intérieure du bloc fonction `copie2` et non au `main`.

En ce qui concerne les l'utilisation de la récursivité, elle fonctionne avec inline car le remplacement est typé.

> Voir exercice `07-Jesus_deNazareth/`.

Exercice 8

Question 1

Les prototypes sont donnés dans le `main.cpp`.

Question 2

Le compilateur choisi la fonction qui a les types de paramètres les plus compatibles avec les siens.
Ici, il choisira la fonction qui prend des `int` en priorité. Si celle-ci n'existe pas, il choisira celle avec des `float`.
Mais probabalement jamais celle avec des `int[]` car pas assez proche.
Nous avons testé en supprimant la fonction avec des `int` :

```
cesar@jarvis:~/Documents/school/Université/td/poo/td3/8-Thomas$ ./a.out
float
3
float
5.189
```

```
tab
9
9
9
9
9
9
9
9
9
9
float
3
```

Question 3

Il n'y a aucun problème à créer une fonction `Somme()` avec 3 parmètres de même types, possible pour les mêmes raisons qu'évoqués à la réponse 2.

Question 4

Il est possible de créer une fonction `Somme()` avec comme paramètres deux types différents, possible pour les mêmes raisons qu'évoqués à la réponse 2.

> Voir exercice `08-Thomas/`.

Exercice 9

> Voir exercice `09-Jacques_leMajeur`

Exercice 10

> Voir exercice `10-Jacques_leMajeur`

Exercice 11

> Voir exercice `11-Mathieu`

Exercice 12

Question 1

`friend` sur une fonction signifie que cette fonction à accès à tous les membres `private` et `protected` de la classe dans laquelle la fonction est déclarée comme `friend`
`friend` sur une classe signifie qu'elle a accès à tous les membres `private` et `protected` de la classe "hôte".

> Voir exercice `12-Thaddée_dEdesse`

Exercice 13

Question1

Les constructeurs sont implémentés dans la classes car aucun fichier header n'est déclaré.
Les différences avec des constructeurs implémentés à l'extérieur de la classe (header) sont par exemple le

temps de compilation.
En effet le compilateur devra lire entièrement l'implémentation alors qu'il pourrait ne lire que l'interface, de plus à chaque fois que le fichier sera inclus, l'implémentation sera compilé. Ceci peut-être évité en déclarant l'implémentation dans un fichier séparé. Ainsi seule l'interface sera compilé à chaque inclusion.

Question 2

On peut déclarer la fonction dans le fichier où sont déclarées les classes vecteur et matrice, mais il faut impérativement que produit soit déclaré après matrice afin que le compilateur ait déjà connaissance de cette dernière.
Afin que produit puisse accèder aux données membres des objets mat et vect on doit définir des accesseurs en lecture (method get) sur ces objets.
Il est a noter que si les classes Matrice et Vecteur sont implémentées dans des fichiers distincts, il est possible de définir produit comme une fonction amie des deux classes, et donc se passer des accesseurs.

Question 3

On peut définir Matrice comme une classe amie de Vecteur, elle pourra donc accéder aux données membres de Vecteur.

> Voir exercice `13-Simon_leZelote`