



Piscine C++ - d07m

Resistance is Futile

Pierre-Yves “Belga” Lefeuve [lefeuv\\_p@epitech.eu](mailto:lefeuv_p@epitech.eu)

*Abstract: Ce document est le sujet du d07m*

# Table des matières

I	REMARQUES GÉNÉRALES	2
II	Exercice 0	4
III	Exercice 1	8
IV	Exercice 2	12
V	Exercice 3	14
VI	Exercice 4	17
VII	Exercice 5	19

# Chapitre I

## REMARQUES GÉNÉRALES

- REMARQUES GÉNÉRALES :

- Si vous faites la moitié des exercices car vous avez du mal, c'est normal. Par contre, si vous faites la moitié des exercices par flemme et vous tirez à 14h, vous AUREZ des surprises. Ne tentez pas le diable.
- Toute fonction implémentée dans un header ou header non protégé signifie 0 à l'exercice.
- Toutes les classes doivent posséder un constructeur et un destructeur.
- Toutes les sorties se font sur la sortie standard et sont terminées par un retour à la ligne sauf si le contraire est précisé 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`

- De façon générale, les fichiers associés à une classe seront toujours `NOM_DE_LA_CLASSE.h` et `NOM_DE_LA_CLASSE.cpp` (s'il y a lieu).
- Les répertoires de rendus sont `ex00`, `ex01`, ..., `exN`
- Toute utilisation de `friend` se soldera par un -42, `no questions asked` .
- Lisez attentivement les exemples, ils peuvent requérir des choses que le sujet ne dit pas...

- Ces exercices vous demandent de rendre beaucoup de classes, mais la plupart sont TRÈS courtes si vous faites ça intelligemment. Donc, halte à la flemme !
- Lisez ENTIÈREMENT le sujet d'un exercice avant de le commencer !
- REFLÉCHISSEZ. 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 nécessaires 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.
  - Rappelez-vous, on fait du C++ maintenant, donc le compilateur est `g++` !
  - Ce sujet peut être modifié jusqu'à 4h avant le rendu. Rafraichissez-le régulièrement !
  - Le répertoire de rendu est : `(DÉPOT SVN - piscine_cpp_d07m-promo-login_x)/exN` (N étant bien sur le numéro de l'exercice).

# Chapitre II

## Exercice 0

	Exercice : 00	points : 4
Welcome to the Federation! Creation of Starfleet		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d07m-promo-login_x)/ex00		
Compilateur : g++	Flags de compilation: -Wall -Wextra -Werror	
Makefile : Non	Règles : n/a	
Fichiers a rendre : Federation.hh, Federation.cpp, Warpsystem.hh, Warpsystem.cpp		
Remarques : n/a		
Fonctions Interdites : *alloc, free, *printf		

La Fédération des Planètes Unies est une alliance entre des peuples capables de voyager dans l'espace à vitesse de distorsion - ou warp - (revenant à se déplacer dans le subespace) et partageant des valeurs communes.

Étroitement liée à la Fédération, Starfleet est un organisme dont la mission première est de récolter autant d'informations que possible concernant l'Univers (la vie et le reste).

La flotte assure par ailleurs un rôle défensif (ce qui explique que ses navires soient armés), voire offensif lorsque les circonstances l'exigent.

Vous allez donc devoir créer un namespace `Federation` , qui va contenir divers éléments vous permettant de mettre en place la Fédération.

`Starfleet` est également un namespace, contenu dans `Federation` , qui va contenir une classe `Ship` , vous permettant de créer des vaisseaux spatiaux.

Chaque `Ship` possède les attributs suivants :

```
int      _length;
int      _width;
std::string _name;
short    _maxWarp;
```

Qui devront tous être fournis à la construction, et ne pourront pas être modifiés par la suite.

Le prototype du constructeur sera donc le suivant :

```
1 Ship(int length, int width, std::string name, short maxWarp)
```

Lors de la création, chaque vaisseau affichera sur la sortie standard :

```
1 The ship USS [NAME] has been finished. It is [LENGTH] m in length and [WIDTH]
  m in width.
2 It can go to Warp [MAXWARP] !
```

(Bien sur vous devrez remplacer `[NAME]` , `[LENGTH]` , `[WIDTH]` et `[MAXWARP]` par les valeurs appropriées)

Chaque `Ship` nécessite un système compliqué pour naviguer à travers l'espace, vous allez donc devoir le fournir. Comme ce système n'est pas réservé aux `Ships` de la Fédération, vous devrez créer un autre namespace appelé `WarpSystem` .

Ce namespace contiendra une classe `QuantumReactor` . Le `QuantumReactor` contient un seul attribut:

```
bool _stability;
```

qui ne devra pas être fourni à la construction de l'objet, mais sera `True` par défaut.

Vous fournirez également une fonction membre `isStable` permettant de vérifier la stabilité du `QuantumReactor` et une fonction membre `setStability` qui modifiera la stabilité :

```
bool isStable();
void setStability(bool);
```

`WarpSystem` contiendra également une classe `Core` , qui possède un seul attribut :

```
QuantumReactor *_coreReactor;
```

Qui devra être fourni à la création de l'objet. Une fonction membre `checkReactor()` permettra d'accéder au réacteur (elle renverra donc un pointeur sur `QuantumReactor` ).

La classe `Ship` aura également une fonction membre `setupCore` , qui prendra un pointeur sur `Core` en paramètre, et ne renverra rien. Cette fonction membre permettra de stocker un `Core` dans votre `Ship` et affichera :

```
1 USS [NAME]: The core is set.
```

Ainsi qu'une fonction `checkCore` , ne prenant pas de paramètre, et affichant sur la sortie standard :

```
1 USS [NAME]: The core is [STABILITY] at the time.
```

( `STABILITY` sera remplacé par `stable` pour `True` , et par `unstable` pour `False` )

Il sera également possible de créer des objets **Ship** n'appartenant pas à **Starfleet**. Ces objets auront les mêmes attributs et fonctions mais seront construits différemment, la vitesse maximum pour un vaisseau n'appartenant pas à **Starfleet** étant de 1. Un vaisseau indépendant affichera lors de sa création sur la sortie standard :

```
1 The independant ship [NAME] just finished its construction. It is [LENGTH] m
  in length and [WIDTH] m in width.
```

De même les affichages des autres fonctions seront différents comme visible dans l'exemple.

Le code suivant devra compiler et afficher la sortie ci-après :

```
1 int main(void)
2 {
3     Federation::Starfleet::Ship UssKreog(289, 132, "Kreog", 6);
4     Federation::Ship Independant(150, 230, "Greok");
5     WarpSystem::QuantumReactor QR;
6     WarpSystem::QuantumReactor QR2;
7     WarpSystem::Core core(&QR);
8     WarpSystem::Core core2(&QR2);
9
10
11
12
13     UssKreog.setupCore(&core);
14     UssKreog.checkCore();
15     Independant.setupCore(&core2);
16     Independant.checkCore();
17
18     QR.setStability(false);
19     QR2.setStability(false);
20     UssKreog.checkCore();
21     Independant.checkCore();
22     return 0;
23 }
```


Sortie :

```
1 belga@riva ex00$ g++ -W -Wall -Werror *.cpp
2 belga@riva ex00$ ./a.out | cat -e
3 The ship USS Kreog has been finished. It is 289 m in length and 132 m in width
  .$.
4 It can go to Warp 6!$.
5 The independant ship Greok just finished its construction. It is 150 m in
  length and 230 m in width.$.
6 USS Kreog: The core is set.$.
7 USS Kreog: The core is stable at the time.$.
8 Greok: The core is set.$.
9 Greok: The core is stable at the time.$.
10 USS Kreog: The core is unstable at the time.$.
11 Greok: The core is unstable at the time.$.
12 belga@riva ex00$
```



# Chapitre III

## Exercice 1

	Exercice : 01	points : 4
Every ship needs a captain... Except the Borgs.		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d07m-promo-login_x)/ex01		
Compilateur : g++	Flags de compilation: -Wall -Wextra -Werror	
Makefile : Non	Règles : n/a	
Fichiers a rendre : Federation.hh, Federation.cpp, Warpsystem.hh, Warpsystem.cpp, Borg.hh, Borg.cpp		
Remarques : n/a		
Fonctions Interdites : Aucune		

Reprenez les fichiers `Federation` et `Warpsystem` de l'exercice précédent.

L'univers est vaste. S'étendant depuis le quadrant Delta, les Borgs sont une race dangereuse, possédant une technologie incroyable, dûe a leur capacités d'assimilation.

Vous allez donc créer un namespace `Borg`, contenant une classe `Ship`. Les `Ship` Borgs diffèrent de ceux de la `Federation` en plusieurs points.

Tout d'abord ils sont de forme cubique. et n'ont donc pas de longueur et de largeur, mais uniquement une longueur de côté. Ils n'ont pas de nom non plus.

Leurs attributs seront donc :

```
int _side;
short _maxWarp;
```

Les vaisseaux Borgs étant construits sur un modèle unique, leur côté(side) est de 3000m, et leur vitesse maximale de Warp 9. Ces informations n'étant pas fournis a la construction. Lorsqu'un `Ship Borg` est construit, il affiche sur la sortie standard :

```
1 We are the Borgs. Lower your shields and surrender yourselves unconditionally.
2 Your biological characteristics and technologies will be assimilated.
3 Resistance is futile.
```

Un vaisseau borg n'affiche rien lorsqu'il installe un `Core`. Lorsqu'ils le vérifient l'affichage est:

```
1 Everything is in order. // si _stability est true.
```

ou bien

```
1 Critical failure imminent. // si _stability est false.
```

Pour lutter, `StarFleet` a besoin de capitaines d'exception. Vous allez donc créer une classe `Captain` appartenant au namespace `Starfleet` possédant les attributs suivants :

```
std::string _name; //fourni a la construction
int         _age;  //non fourni a la construction
```

Ainsi que les fonctions membres permettant de récupérer le nom, récupérer l'âge et modifier l'âge : `std::string getName(); int getAge(); void setAge(int);`

Vous allez également modifier la classe `Ship` de `Starfleet`, afin de pouvoir lui fournir un capitaine. Vous stockerez donc un pointeur sur `Captain`, et pourrez le modifier avec la fonction membre suivante :

```
1 void promote(Captain*);
```

Qui affichera:

```
1 [CAPTAIN NAME]: I'm glad to be the captain of the USS [SHIP NAME].
```

(Remplacez bien sur les noms par les valeurs appropriées.)

Vous allez créer la classe `Ensign`, qui possèdera un attribut:

```
std::string _name;
```

Vous créerez la classe `Ensign` de façon à ce qu'une enseigne ne puisse être construit que de la façon suivante :

```
1 Ensign(std::string name);
```

Et que les appels suivants soient refusés à la compilation :

```
1 Ensign Chekov;
2 Ensign Chekov = (std::string)''Pavel Andreievich Chekov'';
```

À la construction la sortie sera :

```
1 Ensign [NAME], awaiting orders.
```

Le code suivant devra compiler et afficher la sortie ci-après :


```
1 int main(void)
2 {
3     Federation::Starfleet::Ship UssKreog(289, 132, "Kreog", 6);
4     Federation::Starfleet::Captain James("James T. Kirk");
5     Federation::Starfleet::Ensign Ensign("Pavel Chekov");
6     WarpSystem::QuantumReactor QR;
7     WarpSystem::QuantumReactor QR2;
8     WarpSystem::Core core(&QR);
9     WarpSystem::Core core2(&QR2);
10
11     UssKreog.setupCore(&core);
12     UssKreog.checkCore();
13     UssKreog.promote(&James);
14
15     Borg::Ship Cube;
16     Cube.setupCore(&core2);
17     Cube.checkCore();
18
19     return 0;
20 }
```

Sortie :

```
1 belga@riva ex_0$ g++ -W -Wall -Werror *.cpp
2 belga@riva ex_0$ ./a.out | cat -e
3 The ship USS Kreog has been finished. It is 289 m in length and 132 m in width
   .$
4 It can go to Warp 6!$
5 Ensign Pavel Chekov, awaiting orders.$
6 USS Kreog: The core is set.$
7 USS Kreog: The core is stable at the time.$
8 James T. Kirk: I'm glad to be the captain of the USS Kreog.$
9 We are the Borgs. Lower your shields and surrender yourselves unconditionally.
   $
10 Your biological characteristics and technologies will be assimilated.$
11 Resistance is futile.$
12 Everything is in order.$
```

# Chapitre IV

## Exercice 2

	Exercice : 02	points : 4
Get on moving!		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d07m-promo-login_x)/ex02		
Compilateur : g++	Flags de compilation: -Wall -Wextra -Werror	
Makefile : Non	Règles : n/a	
Fichiers a rendre : Federation.hh, Federation.cpp, Warpsystem.hh, Warpsystem.cpp, Borg.hh, Borg.cpp		
Remarques : n/a		
Fonctions Interdites : Aucune		

Vos Ships ont maintenant besoin de se déplacer. Vous allez donc fournir à vos classes `Ship` les attributs suivants :

```
Destination _location;
Destination _home;
```

`Destination` étant un `enum` que vous trouverez dans le fichier `Destination.hh` .  
`_home` est set à :

```
EARTH // pour les Ships de Federation::Starfleet
VULCAN // pour les Ships de Federation
UNICOMPLEX // pour les Ship de Borg
```

À la construction, `_location = _home` .

Vous fournirez également les fonctions membres suivantes :

```
bool move(int warp, Destination d); // deplace _location a d
bool move(int warp); // deplace _location a _home
bool move(Destination d); // deplace _location a d
bool move(); // deplace _location a _home
```

Les fonctions `move` renvoient `true` si :


- `warp <= _maxWarp`

- `d != _location`
- `QuantumReactor::_stability == true`

et false sinon. Bien entendu, si la fonction ne renvoie pas true, aucun déplacement n'est effectué.

# Chapitre V

## Exercice 3

	Exercice : 03	points : 4
This is war!		
So i guess we need weapons. And shields		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d07m-promo-login_x)/ex03		
Compilateur : g++	Flags de compilation: -Wall -Wextra -Werror	
Makefile : Non	Règles : n/a	
Fichiers a rendre : Federation.hh, Federation.cpp, Warpsystem.hh, Warpsystem.cpp, Borg.hh, Borg.cpp		
Remarques : n/a		
Fonctions Interdites : Aucune		

Les vaisseaux peuvent maintenant se déplacer, ils vont avoir besoin de pouvoir attaquer et se défendre. Vous allez fournir aux vaisseaux de **Starfleet** les attributs suivants :

```
int _shield;
int _photonTorpedo;
```

Ainsi que les getteurs et setteurs :

```
int getShield();
void setShield(int);
int getTorpedo();
void setTorpedo(int);
```

A la construction `_shield` est à 100. Vous modifierez le constructeur de `Starfleet::Ship` de façon à pouvoir écrire les appels suivants :

```
Ship(int length, int width, std::string name, short maxWarp, int torpedo);
Ship();
```

et produire les sorties suivantes :

```
1 The ship USS [name] has been finished. It is [length] m in length and [width]
  m in width. It can go to Warp [maxWarp]! Weapons are set: [Torpedo]
  torpedoes ready.
```

Et si aucune information n'est fournie :

```
1 The ship USS Entreprise has been finished. It is 289 m in length and 132 m in
  width. It can go to Warp 6! Weapons are set: 20 torpedoes ready.
```

L'appel au constructeur sans paramètre fera prendre les valeurs par défaut affichées ci-dessus aux différents attributs.

Vous fournirez également aux vaisseaux de `Starfleet` les fonctions membres :

```
void fire(Borg::Ship*);
void fire(int torpedoes, Borg::Ship*);
```

Chaque appel a la fonction `fire` diminuera de 1 ou de `torpedoes` le nombre de `_photonTorpedo` et affichera :

```
1 [SHIPS NAME]: Firing on target. [TORPEDO] torpedoes remaining.
```

et retire `50 * torpedoes` à l'attribut `_shield` de la cible. S'il n'y a plus de torpilles à tirer :

```
1 [SHIP NAME]: No more torpedo to fire, [CAPTAIN NAME]!
```

Attention : vous ne pouvez évidemment pas tirer plus de torpilles que vous n'en avez. Si vous essayez, le message suivant s'affichera :

```
1 [SHIP NAME]: No enough torpedoes to fire, [CAPTAIN NAME]!
```

Vous ajouterez dans la classe `Federation::Ship` une fonction membre `getCore` ne prenant pas de paramètre et renvoyant un pointeur sur le `Core` contenu dans la classe `Federation::Ship`.

Les vaisseaux Borg ont quant à eux les attributs supplémentaires :

```
int _shield; // vaut 100 lors de la construction.
int _weaponFrequency; // doit etre fourni a la construction
short _repair; // peut etre fourni. Sinon, vaut 3
```

Ainsi que les getteurs et setteurs :



```
int getShield();
void setShield(int);
int getWeaponFrequency();
void setWeaponFrequency(int);
short getRepair();
void setRepair(short);
```

Les appels suivants aux constructeurs de `Borg::Ship` devront être valides :

```
Ship(int wF, short);
Ship(int wF);
```

Vous leur fournirez les fonctions suivantes:

```
void fire(Federation::Starfleet::Ship*); // enleve _weaponFrequency;
                                         a l'attribut _shield de la cible.
void fire(Federation::Ship*); // rend le QuantumReactor de la cible
                               instable.
void repair(); // enleve une charge de _repair (si _repair > 0),
               // remet _shield a 100.
```

Les fonctions `fire` de `Borg::Ship` afficheront la sortie suivante :

```
1 Firing on target with [WEAPONFREQUENCY]GW frequency.
```

(Vous remplacerez évidemment `[WEAPONFREQUENCY]` par la valeur qui convient...

La fonction membre `repair` affichera la sortie suivante (s'il est possible de réparer) :

```
1 Begin shield re-initialisation... Done. Awaiting further instructions.
```


Et sinon :

```
1 Energy cells depleted, shield weakening.
```

Vous ne devriez pas avoir besoin d'un main de test la...

# Chapitre VI

## Exercice 4

	Exercice : 04	points : 4
Commanders, be ready Create your fleet		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d07m-promo-login_x)/ex04		
Compilateur : g++	Flags de compilation: -Wall -Wextra -Werror	
Makefile : Non	Règles : n/a	
Fichiers a rendre : Admiral.hh, Admiral.cpp, BorgQueen.hh, BorgQueen.cpp		
Remarques : n/a		
Fonctions Interdites : Aucune		

Vos flottes peuvent maintenant se déplacer et tirer, vous allez avoir besoin de les commander.

Pour cela, vous allez créer deux classes. Tout d'abord une classe `Admiral` qui appartiendra au namespace `Starfleet` (lui-même dans le namespace `Federation`). Cette classe contiendra l'attribut suivant en privé :

```
std::string _name; // fourni au constructeur
```

Lors de l'appel au constructeur il affichera :

```
1 Admiral [NAME] ready for action.
```

Et aura en public deux pointeurs sur fonction membre : L'un pointera sur la fonction `move(Destination)` de la classe `Ship` contenue dans `Federation::Starfleet` : `movePtr`; L'autre sur la fonction `fire(Borg::Ship*)` de la même classe : `firePtr`; La classe possédera deux fonctions membres dont voici les prototypes :

```
void fire(Federation::Starfleet::Ship*, Borg::Ship*);
bool move(Federation::Starfleet::Ship*, Destination);
```

A l'appel de la fonction membre `fire`, vous afficherez (suivi d'un retour à la ligne) :

```
1 On order from Admiral [NAME]:
```

Cet affichage se fera avant l'appel de `fire` .



Attention, vous ne devez pas appeler directement les fonctions membres `move` ou `fire` de `Ship` .

La classe `BorgQueen` (appartenant au namespace `Borg` ) quant à elle aura trois pointeurs sur fonction membre en `public` :

- `movePtr` sur la fonction membre `move(Destination)` de la classe `Borg::Ship`
- `firePtr` sur la fonction membre `fire(Federation::Starfleet::Ship*)` de la même classe
- `destroyPtr` sur la fonction `fire(Federation::Ship*)`


Avec trois fonctions membres utilisant ces pointeurs :

```
bool move(Borg::Ship*, Destination);
void fire(Borg::Ship*, Federation::Starfleet::Ship*);
void destroy(Borg::Ship*, Federation::Ship*);
```

Les pointeurs de chaque fonction membre seront initialisés dans les constructeurs des classes.

# Chapitre VII

## Exercice 5

	Exercice : 05	points : 1
The kobayashi-maru exam		
Répertoire de rendu: (DÉPOT SVN - piscine_cpp_d07m-promo-login_x)/ex05		
Compilateur : g++	Flags de compilation: -Wall -Wextra -Werror	
Makefile : Non	Règles : n/a	
Fichiers a rendre : Exam.hh, Exam.cpp		
Remarques : n/a		
Fonctions Interdites : Aucune		

Vous devrez rendre la classe Exam de manière à ce que le code suivant compile :

```

1 int main(void)
2 {
3     Exam e = Exam(&Exam::cheat);
4     e.kobayashiMaru = &Exam::start;
5     (e.*e.kobayashiMaru)(3);
6     Exam::cheat = true;
7     if (e.isCheating())
8         (e.*e.kobayashiMaru)(4);
9 }
```

et produise la sortie suivante:

```

1 belga@riva ex_0$ g++ -W -Wall -Werror *.cpp
2 belga@riva ex_0$ ./a.out | cat -e
3 [The exam is starting]$
4 3 Klingon vessels appeared out of nowhere.$
5 they are fully armed and shielded$
6 This exam is hard... you lost again.$
7 [The exam is starting]$
8 4 Klingon vessels appeared out of nowhere.$
```

```
9 they are fully armed and shielded$  
10 What the... someone changed the parameters of the exam !$
```