Piscine C++ - d07m

Resistance is Futile

Pierre-Yves "Belga" Lefeuvre `lefeuv_p@epitech.eu`

Abstract: *This document is the subject for d07m*

Contents

Chapter I

GENERAL REMARKS

- GENERAL REMARKS :

  - If you only complete half of the exercises because you're having trouble, that's fine and somehow expected. However, if you only complete half of the exercises because you're lazy and go back home at 2PM, you WILL have (bad) surprises. Don't take that chance.

  - Any function implemented in a header or unprotected header will land you a 0 to the exercise.

  - All classes must include a constructor and a destructor.

  - Any output will be written on the standard output and will be followed by a newline unless specified otherwise.

  - Required filenames must be STRICTLY respected, as well as names of classes and member functions / methods.

  - Please remember that you've stopped using C and actually started C++. Therefore, the following functions are FORBIDDEN, and using them will mean your work is worth the grade of -42:

    * `*alloc`

    * `*printf`

    * `free`

  - Most of the time, class-related file will always be named NAME_OF_THE_CLASS.hh and NAME_OF_THE_CLASS.cpp (if need be).

  - The directory in which you'll submit your work are ex00, ex01,...,exN

  - Any usage of `friend` will land you a -42, `no questions asked` .

○ Please take some time to read the examples, as they can require elements unspecified by the subject itself.

○ You will have to submit a lot of classes in the following examples, but most of them are VERY short if you write them in a clever way. So, raise your lazyness-shield and get to work !

○ Please read ALL the exercise requirements before actually starting it.

○ THINK. Please.

- COMPILATING YOUR EXERCISES :

○ The moulinette will use the flags `-W -Wall -Werror` to compile your code.

○ To avoid any compilation problem with the moulinette, include any necessary file in your headers (*.hh files).

○ Please do note that no function `main` must exist within your code. We will use our own `main` function to compile and test your code.

○ Remember : we're writing C++ now, so the compiler is g++ !

○ We can add modifications to this subject up to 4 hours before the time of your submission. Please regularly refresh this document !

○ The repositories for your submissions are named as follow: (DÉPOT SVN – `piscine_cpp_d07m-promo-login_x`)/exN (N being the exercise number, of course).

Chapter II

Exercise 0

Welcome to the Federation ! Creation of Starfleet	
Turn-in directory: (DÉPOT SVN - `piscine_cpp_d07m-promo-login_x`)/ex00	
Compiler: `g++`	Compilation flags: `-Wall -Wextra -Werror`
Makefile: `No`	Rules: `n/a`
Files to turn in : `Federation.hh, Federation.cpp, Warpsystem.hh, Warpsystem.cpp`	
Remarks : `n/a`	
Forbidden functions : `*alloc, free, *printf`	

The United Planets Federation is an alliance of people able to travel through space. They all possess the distorsion speed - or warp - technology (allowing them to travel through subspace) and all share common values.

Starfleet is an organisation closely linked to the Federation. Its primary mission is to harvest as much informations as possible about the Universe (and life and everything).
The fleet also has a defensive purpose (which explains why all their vessels prepped and armed), which can turn offensive if necessary.
You will therefore create the `Federation` namespace, which will contain all the element that will allow the Federation to exist.
`Starfleet` is also a namespace, existing within `Federation`. It will contain a class named `Ship`, which will be used to create spaceships.

Each `Ship` will have the following attributes:

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

They will all be given during the `Ship`'s construction, and can never be modified later on.
The constructor will have the following prototype:

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

When created, each `Ship` will display on the standard output:

```
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]!
```

(You will of course replace `[NAME]`, `[LENGTH]`, `[WIDTH]` et `[MAXWARP]` by the appropriate values)

Each Ship requires a complicated system to navigate through space, that you will have to provide. Since this system is not entitled to the Federation's `Ships`, you must create a new namespace called `WarpSystem`.

This namespace will house the class `QuantumReactor`. The `QuantumReactor` has only one attribute:

```
    bool _stability;
```

which will not be provided during the object's construction, but will be `True` by default.

You must also provide a member function `isStable`, which will verify the stability of the `QuantumReactor`, as well as a member function `setStability` which can modify it.

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

`WarpSystem` will also contain a `Core` class, with a single attribute:

```
    QuantumReactor *_coreReactor;
```

It will be provided during the object's construction. A member function `checkReactor()` will allow access to the reactor (it will therefore return a pointer on the `QuantumReactor`).

The `Ship` class will then have a member function `setupCore`, which will take a pointer on a `Core` as its parameter, and won't return anything. This member function will stock a `Core` in your `Ship`, and will display on the standard output :

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

The `Ship` will also have a `checkCore` function, with no parameter, which displays on the standard output :

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

(`STABILITY` must be replaced by `stable` for `True`, and by `unstable` for `False`)

It will also be possible to create `Ship` objects that do not belong to the `Starfleet`. These objects will have the same functions and attributes, but the building process will be different. An independent ship has a maximal speed of 1. On its creation, it displays the following text:

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

The other functions will display some different stuff, as you will see in the example.

The following code must compile and print out what follows:

```
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 }
```

Output :

```
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$
```

Chapter III

Exercise 1

	Exercise : 01	points : 4

Every ship needs a captain...
Except the Borgs.

Turn-in directory: (DÉPOT SVN - piscine_cpp_d07m-promo-login_x)/ex01	
Compiler: g++	Compilation flags: -Wall -Wextra -Werror
Makefile: No	Rules: n/a
Files to turn in : Federation.hh, Federation.cpp, Warpsystem.hh, Warpsystem.cpp, Borg.hh, Borg.cpp	
Remarks : n/a	
Forbidden functions : None	

You will reuse the files `Federation` and `Warpsystem` from the previous exercise.

The universe is a big place. Spreading their influence from the Delta quadrant, the Borgs are a dangerous race, and possess and incredible technology, thanks to their power of assimilation.

You will create a namespace Borg, housing a class `Ship`. The Borg's `Ship`s are different from the Federaration's in many aspects.

First, they have the form of a cube. They thus have to width and height, but a single side length. They have no name either.

They attributes will be:

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

The Borg vessels are built on an unic model, their side is 300 meters long, and their maximum speed is Warp 9. These informations are not given during construction. When a `Borg` `Ship` is built, he displays on the standard ouput:

```
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.
```

A Borg vessel does not display anything when installing a `Core` . Upon its verification however, they will display:

```
1  Everything is in order. // if _stability is true.
```

or

```
1  Critical failure imminent. // if _stability is false.
```

`Starfleet` will need outstanding crewmen and captain to face this threat. You will create a class `Captain` inside the `Starfleet` namespace with the following attributes :

```
    std::string _name; //given during construction
    int         _age; //not given during construction
```

As well as the methods allowing consultation of the name, age, and a way to modify said age: std::string getName(); int getAge(); void setAge(int);

You will also modify the `Starfleet` 's `Ship` class, so that it can accept a captain. You will stock a pointer on a `Captain` , that can be modified using the following method:

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

Which will display:

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

(You will of course replace the names by the appropriate values).

You will create the class `Ensign` , which possess an attribute :

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

There MUST only be one way to build the Ensign class :

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

And the following calls must NOT be compilable :

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

Unpon construction, the Ensign will display :

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

The following code will compile and display :

```
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.$
```

Chapter IV

Exercise 2

	Exercise : 02	points : 4

Get on moving!
Turn-in directory: (DÉPOT SVN - piscine_cpp_d07m-promo-login_x)/ex02

Compiler: g++	Compilation flags: -Wall -Wextra -Werror
Makefile: No	Rules: n/a

Files to turn in : Federation.hh, Federation.cpp, Warpsystem.hh, Warpsystem.cpp, Borg.hh, Borg.cpp
Remarks : n/a
Forbidden functions : None

At some point, your Ships will need to move. You will modify your `Ship` classes with the following attributes:

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

Destination is an `enum` which will be found in the file `Destination.hh` . `_home` is set to :

```
EARTH // for Ships of Federation::Starfleet
VULCAN // for Ships of Federation
UNICOMPLEX // for Ships of Borg
```

During construction, `_location` = `_home` .
You will also add the following methods:

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

The `move` methods return true if :

- warp <= _maxWarp

- d != _location

- QuantumReactor::_stability == true

and false otherwise. Of course, if the method does not return true, the Ship does not move.

Chapter V

Exercise 3

	Exercise : 03	points : 4
This is war! So i guess we need weapons. And shields	
Turn-in directory: (DÉPOT SVN - piscine_cpp_d07m-promo-login_x)/ex03	
Compiler: g++		Compilation flags: -Wall -Wextra -Werror
Makefile: No		Rules: n/a
Files to turn in : Federation.hh, Federation.cpp, Warpsystem.hh, Warpsystem.cpp, Borg.hh, Borg.cpp	
Remarks : n/a	
Forbidden functions : None	

Now that the ships can move, they will need a way to attack and defend themselves. You will provide to `Starfleet` 's Ships these new attributes:

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

As well as getters and setters:

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

During construction, `_shield` is initialized at 100. You will modify `Starfleet::Ship` 's constructor so the following calls are possible:

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

and produce the followins outputs:

```
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.
```

And if no information is given:

```
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.
```

Calling the constructor without parameters will give their default values to all the attributes, as shown above.

You will also implement the following methods within the `Starfleet` 's ships:

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

Each call to the 'fire' function will reduce of 1 or of `torpedoes` the number of `_photonTorpedo` and will display:

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

and removes 50 * `torpedoes` to the target's `_shield` attribute. If the ship doesn't have torpedoes anymore:

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

Of course, you can't fire more torpedoes than your ship currently owns. If you try anyway, you should display the following message :

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

You will add a method `getCore` in the class `Federation::Ship` . It doesn't take any parameter and returns a pointer on the `Federation::Ship` 's `Core` .

The Borg vessels possess the following additional attributes:

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

As well as getters and setters:

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

The following call to the `Borg::Ship` 's constructors must be valid:

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

You will provide the following methods:

```
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.
```

The `Borg::Ship`'s `fire` functions will have the following output:

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

(While obviously replacing [WEAPONFREQUENCY] with the appropriate value...
The method repair will display the following output (if a reparation is possible):

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

Otherwise:

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

You shouldn't really need a main function to test your stuff at this point.

Chapter VI

Exercise 4

	Exercise : 04	points : 4
	Commanders, be ready Create your fleet	
Turn-in directory: (DÉPOT SVN - piscine_cpp_d07m-promo-login_x)/ex04		
Compiler: g++	Compilation flags: -Wall -Wextra -Werror	
Makefile: No	Rules: n/a	
Files to turn in : Admiral.hh, Admiral.cpp, BorgQueen.hh, BorgQueen.cpp		
Remarks : n/a		
Forbidden functions : None		

Now that your fleets can move around and shoot at stuff, you will need a way to command them.

Two classes will be needed to reach this goal. First, an `Admiral` class which belongs to the namespace `Starfleet` (which exists, remember, in the namespace `Federation`). This class will possess the following private attribute:

```
std::string _name; // given at constrution
```

When the constructor is called, it will display:

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

The class will possess two public method pointers: One will point on the method `move(Destination)` of the `Ship` class within `Federation::Starfleet` : `movePtr` ; The other will point on the method `fire(Borg::Ship*)` of the same class: `firePtr` ; There will also be two member functions with the following signatures:

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

Upon calling the method `fire` , you will display the following message, followed by a newline :

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

This should be displayed before calling `fire` .

> ⚠ You must not directly call the methods `move` or `fire` of `Ship` .

The class `BorgQueen` (within the `Borg` namespace) will herself possess three `public` method pointers:

- `movePtr` , pointing on `move(Destination)` from the class `Borg::Ship`

- `firePtr` pointing on `fire(Federation::Starfleet::Ship*)` from the same class

- `destroyPtr` pointing on `fire(Federation::Ship*)`

With three method which will use these pointers:

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

The pointers of each member function will be initialized in the classes' constructors.

Chapter VII

Exercise 5

	Exercise : 05	points : 1
	The kobayashi-maru exam	
Turn-in directory: (DÉPOT SVN - piscine_cpp_d07m-promo-login_x)/ex05		
Compiler: g++	Compilation flags: -Wall -Wextra -Werror	
Makefile: No	Rules: n/a	
Files to turn in : Exam.hh, Exam.cpp		
Remarks : n/a		
Forbidden functions : None		

You must write the Exam class in order for this code to 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 }
```

and output the following :

```
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 !$
```