



STARCRAFT

KREOG-WAR

Par David Giron, Dan Baudry et Pierre-Yves Lefeuvre [Koalab](#)

Abstract: Examen Machine de C++.

Table des matières

I	Consignes	2
II	Exercice 0	4
III	Exercice 1	8
IV	Exercice 2	11
V	Exercice 3	18





Chapitre I

Consignes

* Incoming transmission *

Salutations recrue,

Je suis Gregory Wasteland, responsable du département d'épreuves informatiques au service de l'empire Terran et de son empereur Arcturus Mengsk. Vous avez brillamment passé les tests de sélection précédents et vous allez maintenant pouvoir passer cette dernière épreuve avant de devenir un de nos G-TIPE (Ground Tactical Independant Programmer-Engineer).

Cette épreuve consiste à simuler la programmation de nos équipements militaires de terrain.

Vous devrez placer un fichier `auteur` à la racine de votre dossier `rendu` qui contiendra votre login suivi d'un retour à la ligne.

```
1 >cat auteur
2 vanille.fraise
3 >
```

Cette épreuve est composée de 4 exercices, simulant des missions que vous pourriez être amenés à réaliser en conditions réelles. Soyez donc très concentrés car l'erreur n'est pas permise et la correction s'arrêtera à la première erreur détectée.

En situation réelle, le temps sera votre plus gros ennemi. Vous n'aurez donc que 3 heures pour réaliser cette série de tests.

Dans votre dossier `rendu`, chaque exercice sera placé dans un dossier indépendant. Le nom du dossier correspondra à son numéro.
Ex: `ex0/`, `ex1/`, ...

Certains exercices utilisent le code produit dans l'exercice précédent, vous devrez donc copier les fichiers concernés lorsque cela sera nécessaire.

De plus, le code contenu dans chaque répertoire ne devra pas avoir de fonction `main`. Nous utiliserons notre propre fonction `main` pour compiler et tester vos codes, c'est pourquoi les noms des fichiers et des classes vous sont imposés. Vous devrez donc les respecter à la lettre pour gagner vos points.

Les identifiants que vous serez amenés à écrire auront toujours la forme suivante:

```
unNomDeFonction();  
unAutreNom();  
int unEntier;  
getUneDonneeMembre();
```

Je vous rappelle que le compilateur à utiliser sur votre configuration est `g++` avec les options de compilation `-W -Wall` et `-Werror`.

Pensez à protéger vos fichiers headers.

Notez que toutes les fonctions C (`*alloc`, `free`, `*printf`, etc ...), ainsi que le mot clef `using` en C++, sont illégales dans tout l'empire Terran et leur utilisation sera sanctionnée par l'ordre -42 qui consiste à vous envoyer un Ghost et toute trace de votre existence disparaîtra avec vous. Notez toutefois que votre corps reste la propriété de l'Empire et pourra être utilisé pour diverses recherches scientifiques dont vous n'avez pas à connaître les détails.

Bon courage, l'empereur Mengsk compte sur vous.



Si le header d'un exercice ne stipule "aucune" fonction interdite, cela signifie "aucune fonction interdite en plus de celles du paragraphe précédent."



Les `switch`, les `if` (et/ou `else if`) successifs sont INTERDITS dans la totalité de cet examen. Ces points seront vérifiés à la correction et toute tentative de fraude sera sanctionnée d'un -42 sans appel. Seuls les branchements inférieurs ou égaux à `IF THEN (ELSE IF THEN ELSE)` sont autorisés. Relisez ce paragraphe ATTENTIVEMENT !

Une série de conseils vous aideront tout au long du chemin, servez-vous en!




Lisez l'énoncé des exercices en entier avant de commencer, vous gagnerez du temps !

* End of transmission *

Chapitre II

Exercice 0

	Exercice : 0	points : 5
Implémentation de la classe 'unit'		
Répertoire de rendu: /rendu/ex0		
Compilateur : g++	Flags de compilation: -W -Wall -Werror	
Makefile : Non	Règles : n/a	
Fichiers a rendre : unit.h, unit.cpp		
Remarques : n/a		
Fonctions Interdites : Aucune		

- 0:0
 - Écrivez une classe nommée `unit` de forme canonique, dite de `Coplien` : Constructeur par défaut, par copie, `operator=` et destructeur.
 - Ce standard de programmation, en vigueur dans nos services, vous est imposé pour cette classe. Vous déclarerez la classe et ses fonctions membres dans le fichier `unit.h` puis vous implémenterez ces fonctions dans le fichier `unit.cpp`.



Une classe est dite de forme canonique (de Coplien) si elle présente les 4 fonctions publiques suivantes : constructeur par défaut, constructeur par copie, destructeur (éventuellement virtuel) et opérateur d'affectation (`'='`).



En C on utilise `malloc` et `free`. En C++ on utilise `new` et `delete`.

- 0:1

- Vous allez rajouter les champs protégés suivants à votre classe `unit` :
 - le nom `std::string` `name`
 - le type `std::string` `type`
 - la position en x `int` `posX`
 - la position en y `int` `posY`
 - les dommages `int` `dam`
 - les points de vie actuels `int` `cHP`
 - les points de vie maximum `int` `mHP`

- Il est possible de construire une unité `unit` en précisant (dans cet ordre) son nom, son type, sa position en X, sa position en Y, ses dommages, ses points de vie actuels et ses points de vie maximum. Sinon elles seront initialisées sur leur valeur par défaut à 0, ou "" selon le type.

- 0:2

- Étant donné que ces champs sont protégés, il va maintenant vous falloir coder des accesseurs publics sous cette forme:

— `type getNomDuChamp()` qui renvoie la valeur du champ correspondant. (pour les strings, vous renverrez une référence sur string constante) La syntaxe correspond au préfixe `get` auquel on ajoute l'identifiant du champ dont la première lettre a été capitalisée.

```
ex:      int  getPosX()
         int  getMHP()
         int  getDam()
         ...
```

- 0:3

- Le seul champ à ne pas être utilisé en lecture seule est celui des points de vie courants. Vous allez donc surcharger les opérateurs `+=` et `-=` qui permettront de changer la valeur de ce champ. Vous en profiterez pour vérifier que les points de vie ne puissent monter au dessus des points de vie maximum ni descendre en dessous de 0. Vous adopterez le comportement suivant:

```
1      IF cHP < 0 THEN cHP = 0
2      IF cHP > mHP THEN cHP = mHP
```

- 0:4
 - Les actions suivantes devront être possibles avec votre classe `unit` :
 - Notre chef d'unité souhaite faire l'inventaire du stock d'unités à sa disposition, pour cela il a besoin d'un rapport détaillé sur chaque unité. Une routine est déjà en place pour trier et stocker les unités. Une modification des opérateurs comme `<<` est nécessaire au bon fonctionnement de cette mission. Un code similaire à celui suivant devra compiler et s'exécuter en produisant la sortie correspondante ci-dessous :

```

1  {
2      unit u1;
3      unit u2("Joe", // name
4              "Marine", // type
5              13, 45, // posX, posY
6              6, // dam
7              40, 40); // cHP, mHP
8      unit u3 = u2;
9
10     std::cout << u3.getName() << ", " << u3.getType() << std::endl;
11     std::cout << u2 << std::endl;
12     u2 -= 7;
13     std::cout << u2 << std::endl;
14 }
```

La sortie correspondante sur la console devrait être:

```

1  $>./a.out | cat -e
2  Joe, Marine$
3  Joe is a Marine in 13/45 with 40/40HP damaging at 6$
4  Joe is a Marine in 13/45 with 33/40HP damaging at 6$
```



La forme canonique est-elle suffisante ?



Savez-vous ce qu'est une surcharge ? D'opérateur ;) ? Au pire, Bjarne le sait!



Une simple chaine de caractères est un `'char const * const'` ...



Connaissez-vous la difference entre `'private'` et `'protected'` ?




Connaissez-vous la difference entre `'Private'` et `'Marc Dorcel'` ?



Je vous recommande de sérieusement vérifier cette etape. Si celle-ci est fausse et/ou bancale, inutile de continuer...

Chapitre III

Exercice 1

	Exercice : 1		points : 5
Implémentation d'un marine			
Répertoire de rendu: /rendu/ex1			
Compilateur : g++		Flags de compilation: -W -Wall -Werror	
Makefile : Non		Règles : n/a	
Fichiers a rendre : unit.h, unit.cpp, marine.h, marine.cpp			
Remarques : n/a			
Fonctions Interdites : Aucune			

- 1:1
 - Nos marines sont la première ligne de défense de nos colonies. Les fantastiques armures dans lesquelles ils combattent leur permettent de résister à des situations de combat extrêmes. Ces fiers soldats sont équipés de fusils d'assaut Gauss C-14 'Impaler' qui déclenchent une puissance de feu individuelle impressionnante.
 - L'armure possède son système informatique embarqué que vous allez représenter avec la classe `marine` . Vous déclarerez cette classe dans le fichier `marine.h` et vous implémenterez ses fonctions dans le fichier `marine.cpp` .
 - Cette classe hérite publiquement de la classe `unit` .
 - La classe `marine` est également de forme canonique.
 - La construction d'un marine nécessite obligatoirement le nom du marine, sa position en X et sa position en Y. En tant qu'unité, il conviendra d'initialiser son type, ses dommages, et sa vie.
 - Le type d'un marine est `"Marine"` .
Un marine dispose de 40HP au maximum.
À sa création le marine est en bonne santé, il possède donc tous ses points de vie.

Il inflige 6 points de dommage.

- À l'instanciation de la classe le marine dit:

```
1 Do you want a piece of me, boy ?
```

- À la destruction de la classe le marine 'dit':

```
1 Aaaargh...
```

(avec un grand 'A' et trois petits 'a'!)

- 1:2

- La classe `marine` possède les deux fonctions membres publiques suivantes:

- `void move(int x, int y);`
qui déplace le marine a la position indiquée. Il repond:

```
1 Rock'n'roll !!!
```

- `void stimpack();`
qui augmente les dégats du marine de 1 et diminue ses points de vie de 10.
Le marine répond:

```
1 Ho yeah...
```

- 1:3
 - Les actions suivantes devront être possibles avec votre classe `marine` :


```
1      {
2          marine m2("Jim", 23, 56);
3          marine m3 = m2;
4
5          std::cout << m3 << std::endl;
6          m3 -= 7;
7          std::cout << m3 << std::endl;
8          m3 += 8;
9          std::cout << m3 << std::endl;
10
11         m3.move(6, 8);
12         std::cout << m3 << std::endl;
13
14         m3.stimpack();
15         std::cout << m3 << std::endl;
16     }
```

La sortie correspondante sur la console devrait être:

```
1      $>./a.out | cat -e
2      Do you want a piece of me, boy ?$
3      Do you want a piece of me, boy ?$
4      Jim is a Marine in 23/56 with 40/40HP damaging at 6$
5      Jim is a Marine in 23/56 with 33/40HP damaging at 6$
6      Jim is a Marine in 23/56 with 40/40HP damaging at 6$
7      Rock'n'roll !!!$
8      Jim is a Marine in 6/8 with 40/40HP damaging at 6$
9      Ho yeah...$
10     Jim is a Marine in 6/8 with 30/40HP damaging at 7$
11     Aaaargh...$
12     Aaaargh...$
```

Chapitre IV

Exercice 2

	Exercice : 2	points : 5
Implémentation d'un Véhicule de Construction Spatial (VCS)		
Répertoire de rendu: /rendu/ex2		
Compilateur : g++	Flags de compilation: -W -Wall -Werror	
Makefile : Non	Règles : n/a	
Fichiers a rendre : unit.h, unit.cpp, scv.h, scv.cpp, IScv.h		
Remarques : n/a		
Fonctions Interdites : Aucune		



Oui, les noms des fichiers a rendre sont bons, meme `IScv.h`

- 2:0
 - Concu pour la reconstruction des plate-formes spatiales de Tarsonis, le Véhicule de Construction Spatial, (VCS en francais et SCV en anglais) a prouvé son utilité et sa fiabilité dans le vide spatial.
Rapide et efficace, son utilisation est largement répandue lors des campagnes pour la construction de nos bases avancées.
 - Avant de développer les routines de construction des VCS, vous devrez écrire la classe `scv` . Vous la déclarerez dans le fichier `scv.h` et vous implémenterez ses fonctions dans le fichier `scv.cpp` (soyez très attentifs : VCS en francais et SCV en anglais).

- 2:1

- Le VCS est une unité, votre classe `scv` devra hériter publiquement de votre précédente classe `unit`. Elle est aussi de forme canonique. Elle contient les champs privés suivants:

- `int _gold` (quantité de cristaux transportés)
 - `int _gas` (quantité de gaz transportée)

- À la construction d'un VCS il est possible de préciser son nom, sa position en X et sa position en Y. Sinon elles seront initialisées à 0. (ou "" pour les `std::string`).

Les champs `_gold` et `_gas` sont initialisés à 0. En tant qu'unité, il conviendra d'initialiser son type, ses dommages, et sa vie.

- Le type d'un VCS est `SCV T-280`.

Un VCS dispose de 60HP au maximum.

À sa création le VCS est en bon état, il possède tous ses points de vie.

Il inflige 5 points de dommage.

- Lors de la réification d'un nouveau VCS, il devra afficher:

```
1 SCV ready to go, sir.
```

- Lorsqu'un VCS est détruit il affichera:

```
1 * noise of an exploding SCV *
```

- 2:2

- Le VCS a besoin de routines de récolte des ressources et de réparation (aussi bien les unités mécaniques que les bâtiments). Elles devront être sous la forme des fonctions membres publiques suivantes:

- Prototypes :

- `void gather(int x, int y, int type);`

- `void repair(int x, int y, std::string const &target);`

Pour ces deux opérations, vous devrez mettre la position du VCS à jour. (Et dans tous les cas.)

— pour `gather` : (x et y étant les positions de notre VCS)

— Si type vaut 1 le VCS répondra:

1 Yes sir, I'm gathering gold in x/y.

— Si type vaut 2 le VCS répondra:

1 Yes sir, I'm gathering gas in x/y.

— Sinon le VCS répondra:

1 No way sir.

— Chaque appel réussi a `gather` augmente de 8 la ressource concernée.

— pour `repair`: (`x` et `y` étant les coordonnées de notre VCS et `target` le nom de la cible passé en paramètre à `repair`)

— Le VCS répondra:

1 Yes sir, repairing target in x/y.



Soyez prudent avec la ponctuation.

- 2:3

- Vous allez maintenant implémenter les compétences de construction du VCS.

- Les constructions disponibles pour un VCS sont standardisées, vous allez donc commencer par simuler les constructions possibles via un `enum` contenant les valeurs suivantes:

- `COMMAND_CENTER 1`
- `BARRACK 2`
- `SUPPLY_DEPOT 3`
- `BUNKER 4`

(L'enum se devra d'être dans `scv.h` , dans le scope global)

- Vous déclarerez une interface `IScv` dans le fichier `IScv.h` . Elle contiendra les méthodes pures suivantes:

- A chaque bâtiment correspondra une méthode `protected` respectant le prototype `void buildNomDuBtiment();` . ex: `void buildCommandCenter()`

- L'interface contient également une méthode publique dont le prototype est `void createBuilding(int);` (le paramètre correspond à une valeur de l'enum).

Cette interface sera héritée par votre classe `scv` .

- Dans l'implémentation des méthodes de construction des bâtiments de l'interface, le VCS affichera respectivement pour chaque bâtiment le message suivant :

- `command center :`

```
1 Command center finished sir !
```

- `barrack :`

```
1 Barrack finished sir !
```

- `supply depot :`

```
1 Supply depot finished sir !
```

— bunker :

```
1 Bunker finished sir !
```

— le batiment n'existe pas :

```
1 No such building sir !
```

Le VCS fait appel à la méthode `createBuilding` qui fait elle même appel à une des méthodes précédentes afin de construire le batiment voulu.



Pour résoudre cette question Vous DEVEZ utiliser les pointeurs sur membres. Vous êtes prévenus. Relisez l'en-tête du sujet.

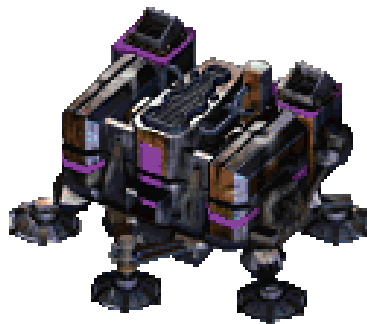
Par exemple, le code suivant :

```
1
2 {
3     scv v1;
4     scv v2("Joe", 6, 3);
5     scv v3 = v2;
6
7     std::cout << v3 << std::endl;
8     v3 -= 3;
9     std::cout << v3 << std::endl;
10    v3 += 5;
11    std::cout << v3 << std::endl;
12
13    v3.gather(2, 3, 1);
14    std::cout << v3 << std::endl;
15    v3.gather(6, 7, 2);
16    std::cout << v3 << std::endl;
17    v3.gather(8, 9, 3);
18    std::cout << v3 << std::endl;
19
20    v3.repair(1, 4, "arclit siege tank");
21    std::cout << v3 << std::endl;
22
23    v3.createBuilding(COMMAND_CENTER);
24    v3.createBuilding(BARRACK);
25    v3.createBuilding(SUPPLY_DEPOT);
26    v3.createBuilding(BUNKER);
27    v3.createBuilding(42);
28 }
```

Génèrera la sortie suivante:


```
1  $>./a.out | cat -e
2  SCV ready to go, sir.$
3  SCV ready to go, sir.$
4  SCV ready to go, sir.$
5  Joe is a SCV T-280 in 6/3 with 60/60HP damaging at 5$
6  Joe is a SCV T-280 in 6/3 with 57/60HP damaging at 5$
7  Joe is a SCV T-280 in 6/3 with 60/60HP damaging at 5$
8  Yes sir, I'm gathering gold in 2/3.$
9  Joe is a SCV T-280 in 2/3 with 60/60HP damaging at 5$
10 Yes sir, I'm gathering gas in 6/7.$
11 Joe is a SCV T-280 in 6/7 with 60/60HP damaging at 5$
12 No way sir.$
13 Joe is a SCV T-280 in 8/9 with 60/60HP damaging at 5$
14 Yes sir, repairing arclit siege tank in 1/4.$
15 Joe is a SCV T-280 in 1/4 with 60/60HP damaging at 5$
16 Command center finished sir !$
17 Barrack finished sir !$
```

```
18 Supply depot finished sir !$  
19 Bunker finished sir !$  
20 No such building sir !$  
21 * noise of an exploding SCV *$  
22 * noise of an exploding SCV *$  
23 * noise of an exploding SCV *$  
24 $>
```



Chapitre V

Exercice 3

	Exercice : 3	points : 5
Implémentation d'une science facility et de ses modules		
Répertoire de rendu: /rendu/ex3		
Compilateur : g++	Flags de compilation: -W -Wall -Werror	
Makefile : Non	Règles : n/a	
Fichiers a rendre : building.h, building.cpp, scienceFacility.h, scienceFacility.cpp, IAddOn.h, physicsLab.h, physicsLab.cpp, covertOps.h, covertOps.cpp		
Remarques : n/a		
Fonctions Interdites : Aucune		



Oui, les noms de fichiers sont bons. Non, IAddOn.h n'est pas mal écrit.

- 3:0
 - La **science facility** est un batiment de recherches avancées sur le terrain. Nos scientifiques, et les G-TIPEs, dont je l'espère vous ferez parti, travaillent vaillamment pour le bien de nos forces. De fantastiques technologies y sont quotidiennement découvertes.
 - De plus, ces laboratoires peuvent être étendus avec un département des services secrets ou un laboratoire de physique afin de nous permettre de déployer les terribles Ghosts et leur assauts nucléaires tactiques ainsi que les titanesques croiseurs de guerre et leur apocalyptique canon Yamato.

Vous allez devoir simuler les recherches disponibles ainsi que les add-on possibles pour une **science facility** .

- 3:1

- Pour cela, vous allez commencer par déclarer une classe `building` qui fonctionne sur le même principe que votre classe `unit`. Vous déclarerez cette classe dans le fichier `building.h` et vous implémenterez ses fonctions dans le fichier `building.cpp`.

Cette classe contiendra les caractéristiques communes à nos bâtiments de guerre, à savoir:

- le nom `std::string name`
- le type (centre de commande, bunker, ...) `std::string type`
- la position en x `int posX`
- la position en y `int posY`
- les points de vie actuels `int cHP`
- les points de vie maximum `int mHP`
- son état (s'il est en vol ou au sol) `bool isFlying_`

- 3:2

- Étant donné que ces champs sont protégés, il va maintenant vous falloir coder des accesseurs publics sous cette forme:

- `type getNomDuChamp()` qui renvoie la valeur du champ correspondant. (pour les strings, vous renverrez une référence sur `string` constante). La syntaxe correspond au préfixe `get` auquel on ajoute l'identifiant du champ dont la première lettre a été capitalisée.

```
ex:      int getPosX();
         int getMHP();
         bool isFlying();
         ...
```

- La classe `building` n'a pas besoin d'être canonique, cependant, elle doit contenir les caractéristiques suivantes:

- Un constructeur par défaut, initialisant les champs à leur valeur par défaut. (à 0, ou "", voir `false` pour le boolean)
- Un constructeur, auquel il est possible de préciser le type, la position en X et la position en Y. Si ce n'est pas le cas elles seront initialisées sur leurs valeurs par défaut. (à 0, ou "", voir `false` pour le boolean)
- Un destructeur

- une surcharge des opérateurs `-=` et `+=` sur le modèle de la classe `unit`
- Une surcharge de l'opérateur `«` qui produit le type de sortie suivant:

```
1 The building is a flying factory in 34/12 with 1250HP.
```

ou encore:

```
1 The building is a landed command center in 2/45 with 1500
  HP.
```

- Nos buildings ne se déplacent qu'en volant d'un point a un autre. Ils ne sont toutefois fonctionnels en terme de recherches et de construction qu'au sol, il faut donc faire attention. Le comportement d'un bâtiment en vol qui entame une recherche ou une construction est indéterminé et ne sera pas testé. Vous allez donc implémenter les routines :

- une fonction membre publique `void fly(int x, int y);` avec les effets suivants :

- ★ si le bâtiment est au sol, elle affiche :

```
1 Flying sequence started, sir ! Moving to x/y.
```

puis fait décoller le bâtiment et le déplace aux coordonnées voulues.

- ★ si le bâtiment est déjà en vol, elle affiche :

```
1 Building's already flying, sir !
```

- une fonction membre publique `void land(void);` avec les effets suivants :

- ★ si le bâtiment est en vol, elle affiche:

```
1 Landing sequence started !
```

puis pose le bâtiment au sol.

- ★ si le bâtiment est déjà au sol, elle affiche :

```
1 Building's not flying, sir !
```

- 3:3

- Vous allez maintenant pouvoir déclarer la classe `scienceFacility` dans le fichier `scienceFacility.h` et implémenter ses fonctions membres dans le fichier `scienceFacility.cpp`
- La classe `scienceFacility` hérite publiquement de votre classe `building`.
- Vous en profiterez pour déclarer les enums suivants (toujours dans le scope global, comme pour le VCS):
 - Les recherches internes à la Science facility:
 - `EMP_BLAST` 1
 - `IRRADIATE` 2
 - `TITAN_REACTOR` 3
 - Les add-ons possibles:
 - `COVERT_OPS` 1
 - `PHYSICS_LAB` 2
- En public, votre classe contiendra :
 - un constructeur par défaut
 - un constructeur stipulant sa position (le X et le Y)
 - un destructeur
 - une fonction membre `void doResearch(int)` prenant en paramètre une des valeurs de l'enum des recherches possibles.
- Le type d'une Science facility est `ScienceFacilityAdvanced`.
La Science facility possède 1500HP au maximum. A sa création, la Science facility est en bon état, elle possède donc tous ses points de vie.

- Pour les 3 recherches possibles, votre classe contiendra un champ privé déterminant si la recherche a été faite et une fonction privée permettant de la faire. Ces fonctions sont appelées par la fonction `doResearch` en fonction du paramètre passé.

La sortie produit selon les cas:

— Pour l'EMP blast

```
1      EMP blast research finished, sir !
```

— Pour l'Irradiate

```
1      Irradiate research finished, sir !
```

— Pour le Titan reactor

```
1      Titan reactor research finished, sir !
```

- Si la recherche a déjà été effectuée:

```
1      This research was already done, sir !
```

- Si la recherche n'existe pas:

```
1      No such research, sir !
```

- Exemple d'utilisation:

```
1      {
2          scienceFacility s1(2, 5);
3          s1.fly(4, 6);
4          s1.land();
5          s1.doResearch(EMP_BLAST);
6          s1.doResearch(IRRADIATE);
7          s1.doResearch(TITAN_REACTOR);
8          s1.doResearch(5);
9          s1.doResearch(EMP_BLAST);
10     }
```

- Sortie:

```
1      $>a.out | cat -e
2      Flying sequence started, sir ! Moving to 4/6.$
3      Landing sequence started !$
4      EMP blast research finished, sir !$
5      Irradiate research finished, sir !$
6      Titan reactor research finished, sir !$
7      No such research, sir !$
8      This research was already done, sir !$
```

- 3:4

- Vous allez maintenant implémenter les add-ons possibles pour une science facility, à savoir les services secrets (`covert ops`) et le laboratoire de physique (`physics lab`).

Ces modules correspondent à une interface qui sera votre première étape. Cette interface se nomme `IAddOn` et vous la déclarerez dans le fichier `IAddOn.h` . Elle contient en public la méthode pure suivante ainsi que tout ce que vous jugerez nécessaire d'ajouter:

- `void doAddOnResearch(int)`

- Cette interface sera concrétisée par les classes `covertOps` et `physicsLab` respectivement déclarées dans `covertOps.h` et `physicsLab.h` . Les enums suivant seront déclarés dans le fichier correspondant: (dans le scope global)

- Covert ops:

- `LOCK_DOWN` 1

- `OCULAR_IMPLANTS` 2

- `CLOAKING_FIELD` 3

- `MOEBIUS_REACTOR` 4

- Physics lab:

- `YAMATO_GUN` 1

- `COLOSSUS_REACTOR` 2

- Ces deux add-ons permettant eux aussi d'effectuer des recherches, l'implémentation de ces deux classes sera très similaire à votre précédente classe `scienceFacility` :
- Pour chacune des recherches possibles, respectivement vos classes contiendront un champ privé déterminant si la recherche a été faite et une fonction privée permettant de la faire. Ces fonctions sont appelées par `doAddOnResearch` en fonction du paramètre.

La sortie produit selon les cas :

— Pour le Lock down

```
1 Lock down research finished, sir !
```

— Pour les Ocular implants

```
1 Ocular implants research finished, sir !
```

— Pour le Cloaking field

```
1 Cloaking field research finished, sir !
```

— Pour le Moebius reactor

```
1 Moebius reactor research finished, sir !
```

— Pour le Yamato gun

```
1 Yamato gun research finished, sir !
```

— Pour le Colossus reactor

```
1 Colossus reactor research finished, sir !
```

— Si la recherche a déjà été effectuée:

```
1 This research was already done, sir !
```

— Si la recherche n'existe pas:

```
1      No such research, sir !
```

- 3:5

- Afin de pouvoir construire un de ces add-ons, vous ajouterez un pointeur public sur `IAddOn` dans votre classe `scienceFacility` qui vous permettra d'agréger un des modules possibles:

```
1      IAddOn* addOn;
```

- Pour cela, vous ajouterez également les deux fonctions membres publiques suivantes à votre classe `scienceFacility` :

— `void buildAddOn(int);`
(le paramètre correspond à une des valeurs de l'enum des modules possibles)

— L'ajout d'un add-on produit la sortie :

```
1      NOM_ADD_ON added to science facility, sir !
```

(Où `NOM_ADD_ON` correspond à l'add-on : `Covert ops` ou `Physics lab`)

— Si un add-on est déjà construit :

```
1      Add on already built, sir !
```

— Un appel à un add-on invalide produit le message suivant :

```
1      No such add on, sir !
```

— `void destroyAddOn(void);`

- La suppression d'un add-on produit la sortie :

```
1 Science facility's add on destroyed, sir !
```

- Un appel de `destroyAddOn()` sur une Science Facility n'en possédant pas produit le message suivant :

```
1 No add on, sir !
```

Exemple:

```
1 {  
2   scienceFacility s1(2, 5);  
3  
4   s1.buildAddOn(COVERT_OPS);  
5   s1.addOn->doAddOnResearch(LOCK_DOWN);  
6   s1.destroyAddOn();  
7   s1.buildAddOn(PHYSICS_LAB);  
8   s1.buildAddOn(PHYSICS_LAB);  
9   s1.addOn->doAddOnResearch(YAMATO_GUN);  
10 }
```

Sortie:

```
1 $>a.out | cat -e  
2 Covert ops added to science facility, sir !$  
3 Lock down research finished, sir !$  
4 Science facility's add on destroyed, sir !$  
5 Physics lab added to science facility, sir !$  
6 Add on already built, sir !$  
7 Yamato gun research finished, sir !$  
8 $>
```