# Travaux dirigés C++ n°3

**Informatique**

*—IMAC 2e année—*

---

## Première classe, suite

Ce TD a pour but de poursuivre l'introduction du concept d'objet en C++. Les étudiants apprendront à définir des accesseurs et des opérateurs.

---

▸ **Exercice 1.** ***Opérateurs setter/getter***

*Objectif de l'exercice :* *Surchages d'opérateurs getter et setter.*
*Révisions :* *surchage et opérateurs*

**A faire** *Nous allons surcharger l'opérateur* [ ], *pour pouvoir faire :*

- `vec[3] = 42;`
- *ou bien* `double x = vec[3];`

*Comme pour l'opérateur* `=`, *l'opérateur* `[ ]` *renvoie une référence sur un double et non un double, c'est pour pouvoir faire* `vec[2] = 42;` *Par ailleurs, nous allons devoir faire deux version : une* `const` *et une "*`pas const`*" :*

1. *Commencez par la version* `const` *qui aura dans* `VectorD.hpp` *le prototype suivant :*
   `const double& operator[](const size_t& i) const;`
   *Il s'agit ici d'un getter, puisqu'on ne peut pas modifier l'objet appelant, on peut juste lire ses données.*
2. *Implantez son code dans* `VectorD.cpp` *(cette méthode doit renvoyer le* $i$*ème élément de notre vecteur).*
3. *Testez dans la fonction* `main.cpp` *avec une instruction du genre* `double x = vec[3];`
4. *Continuez avec la version* `pas const` *qui aura dans* `VectorD.hpp` *le prototype suivant :*
   `double& operator[](const size_t& i);`
   *Il s'agit ici d'un getter pour les objets non* `const` *et d'un setter, puisqu'on peut modifier l'objet appelant.*
5. *Implantez son code dans* `VectorD.cpp` *(cette méthode doit renvoyer le* $i$*ème élément de notre vecteur).*
6. *Testez dans la fonction* `main.cpp` *avec une instruction du genre* `vec[3] = 42;`

▸ **Exercice 2.** ***Protection et encapsulation***

*Objectif de l'exercice :* *Implémenter le concept d'encapsulation et de protection des données.*
*Révisions :* *tp précédent,* `public`, `private`

1. *Rappelez la signification des mots clés* `public` *et* `private`.
2. *Expliquez l'intérêt de protéger les membres et fonctions d'une classe.*
3. *Sur la classe* `VectorD`, *faites en sorte que les membres (attributs) de cette classes deviennent privés, tout en laissant les méthodes publiques.*
4. *Vérifiez qu'il n'est plus possible de faire* `vec.m_data[3] = 42;` *c'est ce qu'on voulait!*
5. *A quoi sert le mot-clé* `inline` *?*
6. *Ajoutez dans le fichier* `VectorD.hpp` *une méthode* `inline` *nommée* `size()` *qui renvoie la taille du vecteur.*

▸ **Exercice 3.** ***Affichage***

*Objectif de l'exercice :* *Surcharger l'opérateur* `<<` *de la fonction* `std::cout`.
*Révisions :* `std::ostream`

*Cet exercice consiste à surchargez la fonction*

```
std::ostream& operator<< (std::ostream& stream, const VectorD& v)
```

*pour qu'elle puisse afficher le contenu d'un* `VectorD` *par l'instruction :*

```
std::cout << vec << std::endl;
```

*où* `vec` *est un* `VectorD`. *À noter que cette fonction n'est pas une méthode de la classe* `VectorD`, *elle doit donc être déclarée en dehors de la classe.*

```
class Plop{
 ...
};

// ici
```

***Remarque :***
*À noter que si la classe* `VectorD` *ne possédait pas de getters pour chaque attribut, l'opérateur* `<<` *devrait être déclarée comme fonction amie de* `VectorD` *afin d'accéder directement à ces attributs (mais ça n'est pas le cas dans cet exercice).*

*Pour définir ce que le flux doit contenir, vous pouvez utiliser l'operateur* `<<` *sur votre flux, comme par exemple :*

```
std::ostream & operator<< (std::ostream & stream, const VectorD & v)
{
  stream << "I love C++\n";
  stream << "I love maths too !!" << std::endl;
  return stream;
}
```

▸ **Exercice 4.** ***Opérateurs***
*Objectif de l'exercice :* *Surchages d'opérateurs usuels.*
*Révisions :* *surchage et opérateurs*

*Certaines opérations sont plus lisibles en utilisant des opérateurs :*

1. *Surchargez l'opérateur + :* `vec3 = vec1 + vec2;`
   *Cet opérateur renvoit-il un "*`VectorD`*" ou bien un "*`VectorD &`*" ?*
2. *Surchargez d'autres opérateurs binaires (*`vec3 = vec1 - vec2;` `vec3 = vec1 * 3;` *...)*
3. *Surchargez le moins unaire :* `vec3 = - vec2;`

*Reflechissez bien aux paramètres et valeurs de retour de ces fonctions.*

▸ **Exercice 5.** ***Entrées/sorties fichier***
*Objectif de l'exercice :* *Voir les mécanismes de d'entrées sorties de fichiers.*

1. *Pour les questions suivantes, vous utiliserez* `fstream`*, avec l'include correspondant.*
2. *Implantez une méthode* `save(const std::string &filename) const` *qui sauvegarde dans le fichier* `filnemane` *la taille et les composantes d'un vecteur.*
3. *Implantez une méthode* `load(const std::string &filename)` *qui lit la taille d'un vecteur dans le fichier* `filnemane` *puis qui charge chacune de ses composantes.*