De l'élimination du mot clé `friend` et de l'utilisation de méthode `const`

Voilà un petit exemple d'utilisation des méthodes `const` Reprenons le code que vous avez écrit lors du TP sur les vecteurs. Voici ici une classe `Vecteur` un peu plus simple que celle que vous aviez définie (elle contient un tableau d'entier de taille trois initialisé à [0,1,2]). A cette classe, sont associés deux opérateur :

`operator +` : qui renvoie la somme de deux vecteurs

`operator <<` : qui affiche le vecteur sur un ostream

Exceptionnellement, la déclaration de la classe et l'implémentation des fonctions sont dans un seul et même fichier `Vecteur.h` :

```
#ifndef VECTEUR_H
#define VECTEUR_H
#include <iostream>
using namespace std;
class Vecteur
{
    protected :
        int tab[3];
    public :
        Vecteur(){ tab[0]=0; tab[1]=1; tab[2]=2;}
        friend ostream & operator<< (ostream &,const Vecteur&);
        friend Vecteur operator+ (const Vecteur&, const Vecteur &);
};
ostream & operator<< (ostream &o,const Vecteur&v)
{
    for(unsigned int i=0;i<3;++i)
        o<<v.tab[i]<<" ";
    cout<<endl;
    return o;
}
Vecteur operator+(const Vecteur& a, const Vecteur& b)
{
    Vecteur c;
    for(unsigned int i=0;i<3;++i)
        c.tab[i]=a.tab[i]+b.tab[i];
    return c;
}
#endif
```

L'utilisation de cette classe se réalise alors avec le fichier `main.cpp` que voici :

```
#include "Vecteur.h"
int main(int, char**,char**)
{
    Vecteur v,v1,v2;
    cout<<v<<v1<<v2;
    v=v1+v2;
    cout<<v<<v1<<v2;
    return 0;
}
```

Comme de bien entendu, l'exécution du main nous donne :

```
0 1 2
0 1 2
```

```
0 1 2
0 2 4
0 1 2
0 1 2
```

Si la déclaration des deux opérateurs comme fonctions `friend` de Vecteur (ce qui permet à ces fonctions d'accéder directement aux attributs de `Vecteur`) est bien pratique, elle met à défaut le principe d'encapsulation cher au concept objet. Je propose donc de retirer les lignes où apparaissent les mots clef `friend` et de faire en sorte que le code recompile de nouveau...

Première étape :

```
#ifndef VECTEUR_H
#define VECTEUR_H
#include <iostream>
using namespace std;
class Vecteur
{
    protected :
        int tab[3] ;
    public :
        Vecteur(){ tab[0]=0 ; tab[1]=1 ; tab[2]=2 ;}
        [redacted]
        [redacted]
};
ostream & operator<< (ostream &o,const Vecteur&v)
{
    for(unsigned int i=0 ;i<3 ;++i)
        o<<v.tab[i]<<" " ;
    cout<<endl ;
    return o ;
}
Vecteur operator+(const Vecteur& a, const Vecteur& b)
{
    Vecteur c ;

    for(unsigned int i=0 ;i<3 ;++i)
        c.tab[i]=a.tab[i]+b.tab[i] ;

    return c ;
}
#endif
```

Dès lors le compilateur râle :

```
In file included from main.cpp:1:
Vecteur.h: In function ‘std::ostream& operator<<(std::ostream&, const
   Vecteur&)’:
Vecteur.h:10: ‘int Vecteur::tab[3]’ is protected
Vecteur.h:18: within this context
Vecteur.h: In function ‘Vecteur operator+(const Vecteur&, const Vecteur&)’:
Vecteur.h:10: ‘int Vecteur::tab[3]’ is protected
Vecteur.h:27: within this context
Vecteur.h:10: ‘int Vecteur::tab[3]’ is protected
Vecteur.h:27: within this context
Vecteur.h:10: ‘int Vecteur::tab[3]’ is protected
Vecteur.h:27: within this context
make: *** [e] Error 1
```

En effet, maintenant que les opérateurs ne sont plus `friend`, ils ne peuvent plus accéder librement aux attributs de `Vecteur`. Rajoutons alors un opérateur d'indexation à `Vecteur` et utilisons celui-ci dans l'opérateur de flux :

```
#ifndef VECTEUR_H
#define VECTEUR_H
#include <iostream>
using namespace std;
```

```
class Vecteur
{
    protected :
        int tab[3] ;
    public :
        Vecteur(){ tab[0]=0 ; tab[1]=1 ; tab[2]=2 ;}
        int operator[](unsigned int index)
        {
            return tab[index] ;
        }
};
ostream & operator<< (ostream &o,const Vecteur&v)
{
    for(unsigned int i=0 ;i<3 ;++i)
        o<<v[i]<<" " ;
    cout<<endl ;
    return o ;
}
Vecteur operator+(const Vecteur& a, const Vecteur& b)
{
    Vecteur c ;
    for(unsigned int i=0 ;i<3 ;++i)
        c.tab[i]=a.tab[i]+b.tab[i] ;
    return c ;
}
#endif
```

Ce qui nous donne à la compilation :

```
In file included from main.cpp:1:
Vecteur.h: In function ‘std::ostream& operator<<(std::ostream&, const
   Vecteur&)’:
Vecteur.h:22: no match for ‘const Vecteur& [unsigned int&]’ operator
Vecteur.h:14: candidates are: int Vecteur::operator[](unsigned int) <near
   match>
Vecteur.h: In function ‘Vecteur operator+(const Vecteur&, const Vecteur&)’:
Vecteur.h:10: ‘int Vecteur::tab[3]’ is protected
Vecteur.h:31: within this context
Vecteur.h:10: ‘int Vecteur::tab[3]’ is protected
Vecteur.h:31: within this context
Vecteur.h:10: ‘int Vecteur::tab[3]’ is protected
Vecteur.h:31: within this context
make: *** [e] Error 1
```

Ce que veut nous signaler le compilateur ici, c'est qu'il ne peut appeler l'opérateur d'indexation sur le `Vecteur` qui lui est passé en argument. En effet, dans le prototype, celui-ci est déclaré comme `const`, c'est à dire qu'il ne sera pas modifié par la fonction. Cela veut dire aussi qu'il ne doit pas être modifié par les méthodes de `Vecteur` utilisées dans l'implémentation de l'opérateur...

C'est bien le cas de notre opérateur de flux ! Mais il faut préciser cela au compilateur en déclarant notre opérateur d'indexation comme étant une méthode `const`... La nouvelle version de `Vecteur.h` est alors : (on notera que l'utilisation de l'opérateur d'indexation a été généralisée à l'opérateur d'addition)

```
#ifndef VECTEUR_H
#define VECTEUR_H
#include <iostream>
using namespace std ;
class Vecteur
{
    protected :
        int tab[3] ;
    public :
        Vecteur(){ tab[0]=0 ; tab[1]=1 ; tab[2]=2 ;}
        int operator[](unsigned int index) const
        {
            return tab[index] ;
```

```
        }
};
ostream & operator<< (ostream &o,const Vecteur&v)
{
    for(unsigned int i=0;i<3;++i)
        o<<v[i]<<" ";
    cout<<endl;
    return o;
}
Vecteur operator+(const Vecteur& a, const Vecteur& b)
{
    Vecteur c;
    for(unsigned int i=0;i<3;++i)
        c[i]=a[i]+b[i];
    return c;
}
#endif
```

Ce qui devrait nous donner un comilation réussie :

```
In file included from main.cpp:1:
Vecteur.h: In function 'Vecteur operator+(const Vecteur&, const Vecteur&)':
Vecteur.h:31: non-lvalue in assignment
make: *** [e] Error 1
```

Et bien non, ce n'est pas fini. Si l'opérateur d'indexation comme nous l'avons défini permet d'accéder à `a[i]` et `b[i]`, cet opérateur ne nous permet pas de modifier la valeur de `c[i]`... Pour que cela soit possible il faut ajouter un autre opérateur d'indexation (dit en *écriture* alors que le précédent est dit de *lecture seule*). Rajoutons cet opérateur et nous obtenons :

```
#ifndef VECTEUR_H
#define VECTEUR_H
#include <iostream>
using namespace std;
class Vecteur
{
    protected :
        int tab[3];
    public :
        Vecteur(){ tab[0]=0; tab[1]=1; tab[2]=2;}
        int operator[](unsigned int index) const
        {
            return tab[index];
        }
        int & operator[](unsigned int index)
        {
            return tab[index];
        }
};
ostream & operator<< (ostream &o,const Vecteur&v)
{
    for(unsigned int i=0;i<3;++i)
        o<<v[i]<<" ";
    cout<<endl;
    return o;
}
Vecteur operator+(const Vecteur& a, const Vecteur& b)
{
    Vecteur c;
    for(unsigned int i=0;i<3;++i)
        c[i]=a[i]+b[i];
    return c;
}
#endif
```

Ce qui compile et nous donne à l'exécution :

```
0 1 2
0 1 2
0 1 2
0 2 4
0 1 2
0 1 2
```