MAM4 - 2017 partiel 1

Exercice 1 (taille)

On considère le programme suivant

```
vector<int> v;
v[0];
cout << v[0];
```

Est il correct ? Proposer éventuellement une correction.

Le bout de code ne produit pas d'erreur à la compilation mais à l'exécution : Erreur de segmentation (core dumped).
En effet, le vecteur v est initialisé avec le constructeur par défaut.
Il a une taille de 0 et l'espace mémoire v[0] n'existe pas. On peut corriger le problème de deux façons :

1. Utiliser un constructeur qui donne une taille non nulle :

   ```
   vector<int> v(1) ; // créer un vecteur de taille un
   v[0] = 2 ;
   cout << v[0] ;
   ```

2. Utiliser la méthode push_back() pour ajouter le nombre à la fin du vecteur

   ```
   vector<int> v ; // créer un vecteur de taille un
   v.push_back(2);
   cout << v[0] ;
   ```

Remarque : Pour créer une variable avec le constructeur par défaut, on ne met pas de paranthèses :

```
vector<int> v(1) ; // ok
vector<int> v() ; // pas d'erreur mais ne créer pas la variable
```

Exercice 2 («)

En général lorsque l'on souhaire implanter l'affichage pour une classe A, l'opérateur « n'est pas une fonction membre. Expliquer pourquoi. en donnant un exemple.

Si `operator<<` est une fonction membre, alors elle prend un premier argument implicite du même type que la classe. Pour l'utiliser on devrait écrire :
`a <<cout;`
au lieu de :
`cout<<a;`

Et pour afficher plusieurs objets à la suite on devrait écrire :
`a3<<(a2<<(a1<<cout));`
au lieu de :
`cout<<a1<<a2<<a3;`

Déclarer `operator<<` en ami ou en externe permet d'avoir un meilleur ordre des arguments.

Exercice 3 (default bug)

On considère le programme suivant

```
class A {
    A() {cout << "0";} // constructeur 0
    A(int i) {cout << "1";} // constructeur 1
};
```

```
int main() // on veut déclarer 2 objets avec le constructeur 0
{          //et 2 autres avec le constructeur 1
    A a;
    A b();
    A c(1);
    A d = 2;
}
```

a. Ecrire ce qui est affiché

si on ajoute `public:` dans la classe A
Le programme affiche : `011`

```
A a; // affiche 0 car utilise le constructeur A()
A b(); // n'affiche rien car n'appelle pas de constructeur (appel d'une fonction b)
A c(1); // affiche 1 car utilise le constructeur A(int)
A d = 2 ; // affiche 1 car convertit 2 grâce au constructeur A(int)
```

b. Est-ce bien ce que l'on attend ? Corriger le programme si nécessaire.

Pour corriger et avoir l'affichage `0011` il faut :

```
A a; // affiche 0 car utilise le constructeur A()
A b = A(); //affiche 0 car utilise le constructeur A()
A c(1); // affiche 1 car utilise le constructeur A(int)
A d = 2 ; // affiche 1 car convertit 2 grâce au constructeur A(int)
```

Exercice 4 (membre, ami, externe)

On rappelle une partie de la classe MVector.

```
class MVector {
private:
    vector<double> v;
public:
    class bad_dim{};

    MVector(int n = 0, double x = 0.0); //constructors

    int dim() const {return v.size();}

    double& operator[](int i) {return v[i];}
    double operator[](int i) const {return v[i];}

    friend MVector operator+(const MVector& mv1, const MVector& mv2);
};
```

a. *Ecrire le constructeur de conversion qui convertit un* ***double*** *en un MVector de taille 1.*

```
MVector(double val) : v(1,val) {}
```

Ou bien en moins propre

```
MVector(double val) {
    v.push_back(val);
}
```

b. *L'opérateur + n'est pas une fonction membre. Expliquer simplement pourquoi en donnant un exemple.*

Si operator+ est une fonction membre, son premier argument est implicite et ne peut pas être convertit automatique en Mvector.

Par exemple, puisqu'on a le constructeur de conversion de `double`, on peut écrire :

```
double d = 3.2 ;
Mvector mv (1,5.6) ;
Mvector sum = mv + d; // ok
// Mais on ne peut pas faire :
Mvector sum = d + mv;
// Si operator+ est une fonction membre
```

On la déclare donc amie ou bien externe, mais pas membre.

c. *Pourquoi peut-on aussi déclarer l'oéprateur + comme une fonction externe ?*

On peut déclarer `operator+` externe car il n'y a pas besoin d'accéder aux données membres private.

On peut se débrouiller avec les fonctions publiques `int dim()` et `operator[](int) const` :

```
MVector operator+ (MVector mv1, MVector mv2){
    // si pas bonne dimension, erreur
    if (mv1.dim()!=mv2.dim()){
        throw MVector::bad_dim();
    }
    // sinon on créer un vecteur de même taille
    MVector res (mv1.dim());
    // on le complète
    for (int i=0; i<mv1.dim(); i++){
        res[i] = mv1[i] + mv2[i];
    }
    return res;
}
```

d. *Ecrire l'opérateur de conversion qui convertit un MVector (ayant un coefficient unique) en un* ***double****. On emmetra une exception* ***bad_dim*** *si la taille du vecteur n'est pas 1.*

```
explicit operator double (){
    // si pas bonne dimension, erreur
    if (dim()!=1){
        throw bad_dim();
    }
    // sinon renvoie premier (et unique) élément
    return v[0];
    }
```

On ajoute `explicit` devant l'opérateur de conversion car sinon, la somme d'un `double` et d'un `MVector` est ambiguë : le compilateur ne sait pas si il faut :

1. Convertir `MVector` en `double` avec l'opérateur de conversion et sommer deux `double`
2. Sommer deux `MVector` en convertissant le `double` en `Mvector` avec le constructeur de conversion

Exercice 5 (squarematrix)

On rappelle une partie de la classe SquareMatrix.

```
class SquareMatrix {
public:
    SquareMatrix(int n = 0, double x = 0.0) : Matrix(n,n,x) {}
```

```
    SquareMatrix(const Matrix& mat); // conversion constructor
    friend SquareMatrix operator+(const SquareMatrix& m, const SquareMatrix& n);
};
```

On suppos que les variables `mat` *et* `sqmat` *sont respectivement de type* `Matrix` *et* `SquareMatrix`*. Pourquoi l'expression* `mat + sqmat` *est-elle ambiguë ? Proposer une correction.*

Le compileur ne sait pas s'il faut :

1. Convertir `sqmat` en `Matrix` et utiliser `operator+(Matrix,Matrix)` ce qui est possible car `SquareMatrix` hérite de `Matrix`.
2. Convertir `mat` en `SquareMatrix` et utiliser `operator+(SquareMatrix,SquareMatrix)` ce qui est possible grâce au constructeur de conversion `SquareMatrix(Matrix)`.

On peut résoudre le problème en ajoutant `explicit` devant `SquareMatrix(Matrix)`.