Programmation Orienté Objet

2022-2023

Travaux Pratiques 4v2

La différence avec la version précédente : un exercice en plus, une question en moins et des aides supplémentaires : le but des exercices de la version 1 restent inchangés.

Site du cours : `https://defelice.up8.site/poo.html`
`g++ mon_fichier.cpp -std=c++17 -Wall -Wextra -g -fsanitize=address -o mon_programme`
Les exercices marqués d'un @ sont à faire dans un second temps.

Dans tous les exercices de ce TP (et des tp suivants) il faut si possible penser à déclarer les méthodes constantes.

Exercice 1. *Echange*

On souhaite construire une fonction `echange` qui échange les contenus de deux variables `int`.

1. Construire la fonction `echange`. Exemple d'utilisation :

```
int a=3;
int b=2;
echange(a,b); // et pas echange(&a,&b);
// ici a==2 et b==3.
```

2. Peut-on donner un sens à : `echange(3,2)` ?

Exercice 2. *Vecteur du plan*

Implanter la classe `vecteur2d_t` qui représente les vecteurs du plan (avec des float) La classe, outre le constructeur doit surcharger les opérateurs

— + : pour la somme
— * : pour le produit par un nombre
— || : pour le produit scalaire entre deux vecteurs.
@ ~~&& pour le produit vectoriel~~ (n'existe pas en 2d)

Exemples :

```
class vecteur2d_t {...};

vecteur2d_t o{0.,0.}, a{3,4};
vecteur2d_t c=o+a;

c.afficher();
c=c*4.5;
// a= c || a; erreur
a.afficher();
float h= a || c;
```

Rappel des opérations :

— Somme de deux vecteurs : $\begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} a+c \\ b+d \end{pmatrix}$
— Produit par un nombre : $\begin{pmatrix} a \\ b \end{pmatrix} \times c = \begin{pmatrix} a \times c \\ b \times c \end{pmatrix}$
— Produit scalaire : $\begin{pmatrix} a \\ b \end{pmatrix} . \begin{pmatrix} c \\ d \end{pmatrix} = a \times c + b \times d$

Exercice 3. *Tableau*

Implanter une classe `tableau20_t` qui implante des tableaux d'entiers de 20 cases. Voici un exemple d'utilisation de la classe.

```
tableau20_t tab1;// creer un tableau de taille  20
                 // initialise chaque case avec la valeur 0

tab1[4]=2; // place 2 dans la cinquième case du tableau

int var1=tab1.nbNul() // renvoie le nombre de cases qui contiennent
         //0 (ici 19)

int var2=tab1[4];
   // ici var2==2
tab1.affiche() // affiche le contenu du tableau sur l'entrée
               // standard
tableau20_t tab2;
tab2=tab1;
tab1=tab2+tab1; //  + fait la somme des deux tableaux
               // case par case. renvoie un nouveau tableau
```

Exercice 4. *Somme 2*

Reprendre la classe `intreliee_t` (voir exercice Somme du tp3) et modifiez-la pour en faire une classe `intreliee2_t` qui fait exactement la même chose que la classe précédente mais en prenant en compte les opérations d'affectation `=` et les constructions par copie. (ce que la classe `intreliee_t` ne prenait pas forcément en compte).

Exercice 5. *Acces*

Créer la classe `acces_t` qui mémorise un nombre entier, et compte les acces à cette valeur depuis la dernière modification.

```
acces_t n{4};
n.nbGet(); // renvoie 0
n.get(); // renvoie 4
n.nbGet(); // renvoie 1
n.nbGet(); // renvoie 1
n.get(); // renvoie 4
n.nbGet(); // renvoie 2
n.set(2);
n.nbGet(); // renvoie 0
n.get(); // renvoie 2
n.nbGet(); // renvoie 1
```

```
acces_t const c{5};
c.nGet(); // renvoie 0
c.get(); // renvoie 5
c.nGet(); // renvoie 1
c.nGet(); // renvoie 1
c.get(); // renvoie 5
c.get(); // renvoie 5
c.nGet(); // renvoie 3
//c.set(4);  erreur de compilation c est constant
```