

**Bonjour à tous et joyeux Noël. J'ai commencé les exercices en C++. Je ne dis pas qu'ils sont tous corrects. Mais si ça peut vous aider, et bien, tant mieux... Le reste sera mis plus tard sur le site... bon blocus et l'année prochaine.**

Si erreur sur un code : me contacter à l'adresse mail suivant :  
guillaumefallon@hotmail.com

## **Exercices en C++**

### **Exercice 1**

Donner la version procédurale et orientée objet des projets suivants :

- a) gérer l'ensemble des personnes étudiant ou travaillant à la HELHO
- b) gérer l'ensemble des voitures d'un parking

### **Exercice 2**

Placer du plus général au plus spécifique les concepts suivants :

- A. humain, footballeur, avant-centre, sportif, skieur, spécialiste du slalom géant
- B. guitare, instrument de musique, trompette, instrument à vent, instrument à corde, violon, saxophone, voix.

### **Exercice 3**

Quelles erreurs seront détectées par un compilateur C++ dans ce fichier source qui est accepté par un compilateur C ?

```
main(){
    int a=10,b=20,c ;
    c=g(a,b) ;
    printf("valeur de g(%d,%d)=%d\n",a,b,c) ;
}
g(int x,int y){
    return (x*x + 2*x*y +y*y) ;
}
```

### **Exercice 4**

Créer une fonction content qui retourne la longueur d'un intervalle ou la surface d'un rectangle ou le volume d'un parallélépipède suivant qu'on passe à la fonction 2, 4 ou 6 paramètres.

Exemples d'appel de la fonction content : content(3.0,8.0,-4.0,6.0) retourne 50,  
content(3.0,8.0) retourne 5 et content (3.0,8.0,-4.0,6.0,6.0,5.0) retourne 1.

### **Exercice 5**

Créer une fonction polynome qui permet d'évaluer un polynôme de degré inférieur ou égal à 3. Exemples d'appel de la fonction polynome : polynome(x,7,6) correspondant

au polynôme  $7x+6$  (si  $x=5 \Rightarrow$  la fonction retourne 41), `polynome(x,7,6,5)`  
correspondant au polynôme  $7x^2+6x+5$  (si  $x$  vaut 1  $\Rightarrow$  la fonction retourne 18).

### **Exercice 6**

Soient les déclarations suivantes :

```
int fct(int) ;           //fonction 1
int fct(float) ;        //fonction 2
void fct(int,float);    //fonction 3
void fct(float,int);    //fonction 4
```

```
int n, p;
float x, y ;
char c ;
double z ;
```

Les appels suivants sont-ils corrects et, si oui, quelles seront les fonctions effectivement appelées et les conversions éventuellement mises en place ?

- a) `fct(n) ;`
- b) `fct(x) ;`
- c) `fct(n,x);`
- d) `fct(x,n);`
- e) `fct©;`
- f) `fct(n,p);`
- g) `fct(n,c);`
- h) `fct(n,z);`
- i) `fct(z,z);`

### **Exercice 7**

Créer des fonctions maximum et minimum permettant de renvoyer le maximum ou le minimum de 2 ou 3 nombres de n'importe quel type de données.

### **Exercice 8**

Ecrire une fonction en ligne qui permet de calculer le carré de 2 nombres.

### **Exercice 9**

Ecrire le programme suivant en ne faisant appel qu'aux nouvelles possibilités d'entrées-sorties de C++:

```
#include <stdio.h>
void main(){
    int n; float x;
    printf("donnez un entier et un float\n ");
    scanf("%d %f",&n,&x);
    printf("Le produit de %d par %f est %f\n",n,x,n*x);
}
```

### **Exercice 10**

Soit le modèle de structure suivant:

```

struct essai{
    int n ; float x ;
};

```

Ecrire une fonction nommée `raz` permettant de remettre à 0 les 2 champs d'une structure de ce type, transmise en argument :

- par adresse
- par référence.

Dans les 2 cas, écrire un petit programme d'essai de la fonction.

### **Exercice 11**

Ecrire plus simplement en C++, en utilisant les spécificités de ce langage les instructions suivantes :

```

double *adtab ; int nval ;
...
printf(« Combien de valeurs ? ) ;
scanf(« %d », &nval) ;
adtab=(double *)malloc(sizeof(double)*nval) ;

```

### **Exercice 12**

Ecrire un programme allouant des emplacements pour des tableaux d'entiers dont la taille est fournie en donnée. Les allocations se poursuivront jusqu'à ce que l'on aboutisse à un débordement de mémoire.

### **Exercice 13**

Ecrire un programme en C++ qui affiche le code ASCII des lettres majuscules et minuscules.

### **Exercice 14**

Quels sont les résultats de l'exécution du programme suivant :

```

#include <iostream.h>
int main(){
    char c='A';
    cout<<c++<< " "<<int(c)<<endl ;
    cout<<c++<< " "<<int(c)<<endl ;
    cout<<c++<< " "<<int(c)<<endl ;
    return 0;
}

```

### **Exercice 15**

Créer une fonction `echange` qui accepte 2 arguments de type entier et échange les valeurs de ces 2 arguments.

Définir 2 fonctions supplémentaires, elles aussi appelées `echange`, l'une échange les valeurs de 2 arguments de type double et l'autre d'arguments de type char.

Créer un programme principal permettant de tester ces fonctions.

Exercice 3

```

#include <iostream.h>
#include <stdio.h> //ne pas oublié de mettre cet include

g(int x, int y);
main(){ //avertissement, mettre void
    int a=10,b=20,c;
    c=g(a,b); //obligation de définir une fonction avant main
    printf("valeur de g(%d,%d)=%d\n",a,b,c);
}
g(int x, int y){
    return(x*x+2*x*y+y*y);
}

```

#### Exercice 4

```

#include <iostream.h>
#include <math.h>

// exercices pour comprendre les fonctions
double fonction (double x, double y);
double fonction (double x,double y, double w, double z);
double fonction (double x,double y, double w, double z,double o,double p);

double fonction (double x, double y)
{
    return (y-x);
}

double fonction (double x,double y, double w, double z)
{
    return fonction(3.0,8.0)*(z-w);
}

double fonction(double x,double y, double w, double z,double o, double p)
{
    return fonction(3.0,8.0,-4.0,6.0)*(o-p);
}

int main()
{
    double intervalle=fonction(3.0,8.0);
    double surface=fonction(3.0,8.0,-4.0,6.0);
    double volume=fonction(3.0,8.0,-4.0,6.0,6.0,5.0);
    cout<<intervalle<<endl;
    cout<<surface<<endl;
    cout<<volume<<endl;
}

```

#### Exercice 5

```

#include <iostream.h>

int polygone(int x, int y, int w);
int polygone(int x, int y, int w, int z);

```

```

int polygone(int x, int y, int w){
    int p= y*x+w;
    return p;
}

int polygone(int x, int y, int w, int z){
    int q= y*x*x+w*x+z;
    return q;
}

void main(){
    int y;
    cout <<"Entrez la valeur de X :"<<endl;
    cin>>y;

    cout<<polygone(y,7,6)<<endl;

    cout <<"Entrez la valeur de X :"<<endl;
    cin>>y;

    cout<<polygone(y,7,6,5)<<endl;
    cout<<"-----"<<endl;
}

```

## Exercice 6

```

#include <iostream.h>
int fct();
int fct(int);
int fct(float);
void fct(int,float);
void fct(float,int);

int n,p;
float x,y;
char c;
double z;

void main(){
    fct(n);
    fct(x);
    fct(n,x);
    fct(c);// c'est ok car c'est comme si on avait mis int fct();
    fct(n,p); // pas bon car p est un entier
    fct(n,c); // il n' y a aucun char en paramètre
    fct(z,z); // il n' y a pas deux float en paramètres
    fct(n,z);

    // rem : double est un long float
}

```

## Exercice 7

```
#include <iostream.h>

float maximum(int x,char y, float z);
float minimum(int x, char y, float z);

float maximum(int x,char y, float z){
    if((x>y)&&(x>z))
        return x;
    if ((y>x)&&(y>z))
        return y;
    if ((z>x)&&(z>y))
        return z;
}
float minimum(int x,char y, float z){
    if ((x<y)&&(x<z))
        return x;
    if ((y<x)&&(y<z))
        return y;
    if ((z<x)&&(z<y))
        return z;
}

void main(){

    cout<<maximum(100,'b',120.0)<<endl;
    cout<<minimum(100,'b',120.0)<<endl;
}
```

## Exercice 8

```
#include <iostream.h>

/*Le mot clé inline remplace avantageusement l'utilisation de #define du
préprocesseur
pour définir des pseudo-fonctions. Afin de rendre l'exécution plus rapide d'une
fonction et à condition que celle ci soit de courte taille, on peut définir une
fonction avec le mot réservé inline. Le compilateur générera, à chaque appel de
la fonction, le code de celle ci. Les fonctions inline se comportent comme des
fonctions normales et donc, présentent l'avantage de vérifier les types de
leurs arguments, ce que ne fait pas la directive #define.
*/
```

```
inline int carre(int n); // déclaration

void main() {
    cout << carre(10) << endl;
}
```

```
// inline facultatif à la définition, mais préférable
inline int carre(int n) {
    return n * n;
}
```

### Exercice 9

```
#include <iostream.h>
```

```
void main(){
    int n;float x;
    cout<<"donnez un entier et un float\n"<<endl;
    cin>>n>>x; //faire un espace
    //cin>>n;
    //cin>>x
    cout<<"le produit de "<<n<<" par "<<x<<" est "<<x*n<<endl;
}
```

### Exercice 10

```
#include <iostream.h>
```

```
/*void raz(int &a, float &b);
struct essai{
    int n; float x;
};
```

```
void raz(int &c, float &d){
    int a=0;
    c=a;
    float b=0;
    d=b;
}
```

```
void main()
{
    struct essai test;
```

```
test.n=2;
test.x=4;
```

```
cout<<test.n<<endl;
cout<<test.x<<endl;
```

```
raz(test.n,test.x);
```

```
cout<<test.n<<endl;
cout<<test.x<<endl;
```

```
}
*/
```

```
struct essai{
    int n; float x;
```

```

};

void raz(int *c, float *d){
    int a=0;
    //c=&a;
    *c=a;
    float b=0;
    //d=&b;
    *d=b;
}

void main()
{
struct essai test;

test.n=2;
test.x=4;

cout<<test.n<<endl;
cout<<test.x<<endl;

raz(&test.n,&test.x);

cout<<test.n<<endl;
cout<<test.x<<endl;
}

```

## Exercices : les classes

### **Exercice 0 :**

Soit la classe voiture avec les méthodes tourne, accélère, allumeBougie, sortPiston, coinceRoue, changeVitesse, injecteEssenceDansCylindre. Quelles sont celles qui font partie de l'interface et celles qui font partie de l'implémentation ?

### **Exercice 1 :**

Réaliser une classe cylindre permettant de travailler avec des cylindres. Elle a comme données membres rayon(float) et hauteur (float). Elle a comme méthodes : un constructeur qui permet une initialisation de ses données membres, la méthode section renvoyant un float (la surface de la base du cylindre) et la méthode volume renvoyant un float (le volume du cylindre). Créer un programme permettant de tester la classe.

### **Exercice 2 :**

Créer une classe parité dont les objets permettent de mémoriser des valeurs entières et qui détermine si le total de ces valeurs est pair ou impair. Cette classe comprendra un constructeur qui permet d'initialiser les données membres (un entier et un pointeur sur des entiers), une méthode qui permet d'ajouter une valeur et une méthode test qui teste si le total des valeurs est pair (retourne 1) ou impair (retourne 0). Créer un programme permettant de tester la classe.

### **Exercice 3 :**

Créer une classe `file_attente` qui permet de gérer des objets fonctionnant sur le mode "premier entré-dernier sorti". Cette classe comprendra un constructeur permettant d'initialiser les données membres (un entier et un pointeur sur des entiers), une méthode qui permet d'ajouter un élément à la pile et une méthode qui permet de retirer un élément à la pile.

Créer un programme permettant de tester la classe.

### **Exercice 4 :**

Créer 3 fichiers pour l'énoncé suivant : (h, cpp et principal).

Créer une classe `cercle` contenant une donnée membre privée `r` (le rayon) et 3 méthodes

publiques :

- un constructeur qui initialise les données membres et qui donne comme valeur par défaut à `r` 1

- `surface` qui renvoie la surface d'un cercle et qui a comme paramètre le rayon (entier)

- `glsurface` qui a comme paramètre le côté d'un carré et qui renvoie la surface d'un carré, elle utilise une fonction globale (externe à la classe), nommée `surface`, qui renvoie la surface d'un carré et qui a comme paramètre un entier (le côté du carré).

Créer un programme appelant les différentes méthodes et affichant les différentes surfaces.

### **Exercice 5 :**

Quels seront les résultats fournis par l'exécution du programme suivant ?

```
class demo{
    int x,y ;
    public :
    demo(int abs=1, int ord=0).
        demo(demo&) ;
        ~demo() ;
};
demo::demo(int abs, int ord){
    x=abs;y=ord;
    cout<<"Constructeur I          ";<<x<<" "<<y<<"\n" ;
}
demo ::demo(demo &d){
    cout<<"Constructeur II (recopie) ";<<d.x<<" "<<d.y<<"\n" ;
    x=d.x ;y=d.y ;
}
demo ::~demo(){
    cout<<"destruction          ";<<x<<" "<<y<<"\n" ;
}
void fonction(demo d,demo *add){
    cout<<"Entrée fonction\n" ;
    delete add ;
    cout<<"Sortie fonction\n" ;
}
void main(){
    cout<<"Début main\n";
    demo a ;
    demo b=2 ;
    demo c=a;
    demo *adr=new demo(3,3);
    fonction(a,adr);
    demo d=demo(4,4) ;
}
```

```

    c=demo(5,5);
    cout<<"Fin main\n " ;
}

```

### **Exercice 6 :**

- a) Créer une classe point pour représenter des points dans un espace à 3 dimensions (x, y, z). Inclure un constructeur par défaut, un constructeur initialisant les données membres, un constructeur par copie, une fonction oppose qui transforme un point en son opposé, une fonction norme qui retourne la distance du point par rapport à l'origine (0, 0, 0) et une fonction affiche. Créer un programme permettant de tester les différents constructeurs et fonctions.
- b) Modifier les constructeurs pour qu'ils acceptent des lignes d'initialisation
- c) Remplacer la fonction affiche en utilisant 3 fonctions d'accès c'est-à-dire des fonctions qui retournent les valeurs de x, de y et de z.
- d) Modifier le constructeur pour qu'il initialise , par défaut, le point avec les coordonnées (1,2,3).

### **Exercice 7 :**

Réaliser une classe Point avec ses 3 coordonnées dans l'espace x, y, z et que l'on peut initialiser de 3 manières différentes (selon les valeurs initiales connues des 3 coordonnées, on connaît soit x, soit x et y, soit x, y et z). Intégrer dans la classe une méthode déplacer qui est surchargée 3 fois, dépendant également desquelles des trois valeurs des translations sont connues.

Créer 2 objets de la classe Point et utiliser la méthode déplacer.

Modifier la classe pour qu'elle utilise un seul constructeur et une seule fonction déplacer mais qui réalise la même chose qu'avant.

#### Exercice 1

```

#include <iostream.h>
#include <math.h>
#define PI 3.141592653589793

class cylindre{
    float rayon;
    float hauteur;

public:
    cylindre(float rn, float hr);
    float section();
    float volume();
    //~cylindre();

};

cylindre::cylindre(float rn,float hr){
    rayon=rn;

```

```

        hauteur=hr;
    }

float cylindre::section(){

float result;
return result=2*PI*rayon*hauteur;

}

float cylindre::volume(){
    float result2;
    return result2=PI*rayon*rayon*hauteur;
}
//cylindre::~cylindre();

void main(){

    cylindre a(3,8);
    cout<<a.section()<<endl;
    cout<<a.volume()<<endl;
}

```

## Exercice 2

```

#include <iostream.h>
#include <string.h>

class parite{
    int size;
    int *pointeur;

public:
    parite(int *point,int taille);
    void ajout();
    int test();
    void affiche();
};

parite::parite(int *point,int taille){
    size=taille;
    pointeur=point;
    pointeur=new int [10];
}

void parite::ajout(){
    int i;
    for (i=0;i<size;i++){
        cout<<"Veuillez ajouter un valeur "<<endl;

        cin>>pointeur[i];
    }
}

void parite::affiche(){
    cout<< "voici le tableau d'entier à tester :"<<endl;
}

```

```

        int i;
        for (i=0;i<size;i++){
            cout<<pointeur[i]<<endl;
        }
    }

int parite::test(){
    int i,tab;
    tab=0;
    for (i=0;i<size;i++)
    {
        tab=tab+pointeur[i];
    }
    cout<<" la somme des entiers est : ";
    cout<<tab<<endl;
    return (tab%2);
}

void main (){
    int *tab=0;
    parite a(tab,4);
    a.ajout();
    a.affiche();
    int y=a.test();
    if (y==0) cout<<"Resultat de ma somme pair"<<endl;
    else cout<<"Resultat de ma somme impair"<<endl;
}

```

### Exercice 3

```

#include <iostream.h>
#define MAX 5
#include <stdlib.h>

class file_attente{
    int size;
    int *pointeur;
public:
    file_attente(int sz, int *point);
    void ajout();
    void affiche();
    void suppression();
    ~file_attente();
};

file_attente::file_attente(int sz, int *point){
    size=sz;
    pointeur=point;
    pointeur= new int[size];

//    for(int i=0;i<size;i++)
//        pointeur[i]=point[i];

```

```

    }

void file_attente::ajout(){
    int i;
    for (i=0;i<size;i++){
        cin>>pointeur[i];
    }
}

void file_attente::affiche(){

    int i;
    for (i=0;i<size;i++){
        cout<<pointeur[i]<<endl;
    }
}

void file_attente::suppression(){

    delete [] pointeur;
}

file_attente::~file_attente(){}

void main()

{
    int tab[MAX];
    for(int i=0;i<MAX;i++) tab[i]=i;
    file_attente a(3,tab);
    a.ajout();
    a.affiche();
    a.suppression();
    a.affiche();
}

```

#### Exercice 4

```
#include <iostream.h>
```

```
int surface(int a);
```

```
class cercle{
```

```
    float r;
public:
```

```

        cercle (float rayon);
        float surface(float rayon);
        int glsurface(int cote);
};

#include "entete.h"

cercle::cercle(float rayon){
    rayon=r;
}

int surface(int a){ // attention GLOBALE

int result1=a*a;
return result1;

}

float cercle::surface(float rayon){
    float pi=3.14159265;
    float s;
    s=pi*rayon*rayon;
    return s;
}
int cercle::glsurface(int cote)
{

    cout<<::surface(4)<<endl; // appel de ma fonction globale !!
    int result;
    result=cote*cote;
    return result;

}

#include <iostream.h>
#include "entete.h"

void main(){

cercle p(2);
cout <<p.surface(2)<<endl;
cout <<p.glsurface(2)<<endl;
}

```

## Exercice 5

```

#include <iostream.h>

class demo{
    int x,y;
public:
    demo(int abs=1, int ord=0);
    demo(demo&);

```

```

        ~demo();
};

demo::demo(int abs, int ord){
    x=abs;
    y=ord;
    cout<<"Constructeur I : "<<x<<" "<<y<<"\n";
}

demo::demo(demo &d){
    cout<<"Constrcuteur II(recopie)"<<d.x<<" "<<d.y<<"\n";
    x=d.x;
    y=d.y;
}
demo::~demo(){
    cout<<"destruction"<<x<<" "<<y<<"\n";
}

void fonction(demo d, demo*add){
    cout<<"Entrée fonction\n";
    delete add;
    cout<<"Sortie fonction\n";
}

void main()
{
    cout<<"Debut main\n";
    demo a;
    demo b=2;
    demo c=a;
    demo *adr=new demo(3,3);
    fonction(a,adr);
//    demo d=demo(4,4);
    demo d(4,4);
    c=demo(5,5);
    cout<<"Fin main\n";
}

```

// NOMBRE ERREUR 27 pour commencer

### Exercice 6

```

#include <iostream.h>
#include <math.h>

class point{
    int x;
    int y;
    int z;
public:
    point();
    point(int dim1,int dim2, int dim3);
    point (point&);
    void oppose();

```

```

        double norme();
        void affiche();
        int affiche1();
        int affiche2();
        int affiche3();
};

point::point(){
    cout<<"Constructeur par défaut"<<endl;
    //x=0;y=0;z=0;
    x=1,y=2,y=3;
}

point::point(int dim1,int dim2, int dim3){
//b) point::point(int dim1, int dim2, int dim3):x(dim1),y(dim2),z(dim3){
    x=dim1;
    y=dim2;
    z=dim3;
}

point::point(point& d){
    cout<<"constructeur par copie";
    x=d.x;
    y=d.y;
    z=d.z;
}

void point::oppose(){
    x=-x;
    y=-y;
    z=-z;
    cout<<x<<" "<<y<<" "<<z<<endl;
}

double point::norme(){

    // pour la distance de 1 point par rapport à son origine, nous n'avons pas
    besoin des 3 points

    double result;
    result =sqrt(pow(x,2)+pow(y,2));
    return result;
}

void point::affiche(){
    cout<<x<<endl;
    cout<<y<<endl;
    cout<<z<<endl;
}

int point::affiche1(){
    return x;
}

```

```

int point::affiche2(){
    return y;
}
int point::affiche3(){
    return z;
}

void main(){
    point a(5,6,7);
    //point b(a);
    a.affiche();
    a.oppose();
    a.norme();
    a.affiche();
    cout<<a.affiche1()<<endl;
    cout<<a.affiche2()<<endl;
    cout<<a.affiche3()<<endl;

    //b.affiche();
}

```

Exercice 7

A toi d e jouer ;-)

## Exercices : les données et les fonctions membres statiques

### **Exercice 1 :**

Parmi les attributs suivants de la classe Renault Kangoo, la version avec toutes les options possibles, séparez ceux que vous déclareriez comme statiques des autres : vitesse, nombre de passagers, vitesse maximale, nombre de vitesses, capacité du réservoir, âge, puissance, prix, couleur, nombre de portières.

### **Exercice 2 :**

Créer une classe point ne contenant qu'un constructeur sans arguments, un destructeur et un membre donnée privé représentant un numéro de point (le premier créé portera le numéro 1, le suivant le numéro 2, et ainsi de suite ...). Le constructeur affichera le numéro du point créé et le destructeur affichera le numéro du point détruit.

Ecrire un petit programme d'utilisation créant dynamiquement un tableau de 4 points et le détruisant.

### **Exercice 3 :**

Créer la classe point suivante et comprenant :

- 2 données membres privées de type float (les coordonnées d'un point)
- un constructeur permettant d'initialiser les données membres

- une méthode deplace ayant 2 paramètres de type float (le déplacement suivant x et le déplacement suivant y) et qui modifie la coordonnée du point suite au déplacement
- une méthode affiche qui affiche les coordonnées du point.

Comment pourrait-on adapter la classe point pour qu'elle dispose d'une fonction membre nombre fournissant le nombre de points existant à un instant donné ?

### **Exercice 4 :**

Elaborer une classe dont tous les objets accèdent à une seule et même pile. Les méthodes de la classe devront permettre à n'importe lequel de ses objets de verrouiller l'accès à la pile afin d'en disposer autant que nécessaire avant d'en libérer l'accès.

#### Exercice 2

```
#include <iostream.h>
```

```
class point{
    int static nombre;
public:
    point();
    ~point();
};

int point::nombre=0;

point::point(){
    nombre++;
    cout<<"point n°"<<nombre<<" construit"<<endl;
}

point::~~point(){
    nombre--;
    cout<<"point n°"<<nombre<<" détruit"<<endl;
}

int main(){
    point *tableau=new point[4];

    delete [] tableau;
return 0;
}
```

#### Exercice 3

```
#include <iostream.h>
```

```
class point{
```

```

        float absc;
        float ordo;
        static int nbre;
public:
    point(float x, float y);
    void deplacement(float depl_x, float depl_y);
    void affiche();
    static void nombre();
};

int point::nbre=0;

point::point(float x, float y){
    absc=x;
    ordo=y;
}

void point::deplacement(float depl_x, float depl_y){

    absc=depl_x;
    ordo=depl_y;
    ++nbre;
}

void point::affiche(){
    cout <<absc<<endl;
    cout <<ordo<<endl;
}

void point::nombre() {
    cout<<"nombre de point = "<<nbre<<endl;
}

void main(){
    point::nombre();
    point a(0,0);
    a.deplacement(1.0,0);
    point::nombre();
    a.affiche();
    a.deplacement(2.2,3.3);
    point::nombre();
    a.deplacement(2.3,3.4);
    a.deplacement(2.5,3.6);
    point::nombre();
    a.affiche();
}

```

Exercice 4

```

#include <iostream>
using namespace std;

static int numero=1;
// verrouille=0;
//static int tab[100];

class pile
{
    int nobjet;
public:
    pile();
    ~pile();
    int activation();
    void desactivation(int a);
};

pile::pile()
{
    nobjet=numero;
    // tab[numero]=1;
    numero++;
    cout<<"objet "<<nobjet<<" construit"<<endl;
}

pile::~~pile()
{
    // tab[nobjet]=0;
    cout<<"objet "<<nobjet<<" detruit"<<endl;
}

int pile::activation()
{
    // verrouille=1;
    cout<<"pile verrouillee par objet "<<nobjet<<endl;
    return nobjet;
}

void pile::desactivation(int a)
{
    if (a==nobjet)
    {
        // verrouille=0;
        cout<<"pile deverrouillee par objet "<<nobjet<<endl;
    }
    else
    {
        cout<<"pile non-deverrouillee par objet "<<nobjet<<endl;
    }
}

int main()
{
    int a;
    pile p1,p2;
}

```

```

    a=p1.activation();
    cout<<"a : "<<a<<endl;
    p2.desactivation(a);
    p1.desactivation(a);
    return 0;
}

```

## Exercices en C++ sur la surcharge des opérateurs

### Exercice 1

Définir une classe permettant d'additionner, de soustraire, de multiplier, de diviser et de donner l'opposé (-) des fractions (rationnels comprenant numérateur et dénominateur) en utilisant la surcharge de ces opérateurs. Créer un programme permettant de tester cette classe.

### Exercice 2

Soit la classe vecteur3d qui contient les coordonnées du vecteur (float), un constructeur initialisant les données membres avec des valeurs par défaut à 0. Il faut définir l'opérateur + pour qu'il fournisse la somme de deux vecteurs et l'opérateur binaire \* pour qu'il fournisse le produit scalaire de deux vecteurs. Créer un programme permettant de tester cette classe.

### Exercice 3

Créer une classe permettant d'additionner (surcharge de +), de soustraire (surcharge de -), de lire et d'afficher des dates simples au format jj/mm (ne pas prendre en compte les années bissextiles).

### Exercice 4

Implémenter dans la classe de l'exercice 3 :

- une surcharge de >> permettant d'entrer le jour et le mois
- une surcharge de << permettant d'afficher le jour et le mois (2 chiffres pour chaque)
- une surcharge de ++ qui incrémente la date d'un jour
- une surcharge de -- qui décrémente la date d'un jour
- une surcharge de == permettant de comparer 2 dates

### Exercice 5

Implémenter dans la classe de l'exercice 2 :

- une surcharge de l'opérateur d'affectation
- une surcharge de +=
- une surcharge de ==
- une surcharge de - qui décrémente les coordonnées du vecteur de 1
- une surcharge de >> qui permet d'entrer les composantes du vecteur
- une surcharge de << qui permet d'afficher les composantes du vecteur.
- une surcharge de l'opérateur d'indexation qui retourne la coordonnée suivant x ou suivant y ou suivant z selon l'indice passé en paramètre

## Exercice 6

Implémenter dans la classe de l'exercice 1 :

- une surcharge des opérateurs de pré et post incrémentation
- une surcharge de l'opérateur de conversion en float
- une surcharge de l'opérateur d'exponentiation sous la forme rationnel operator  $^{(rationnel\&, unsigned \&)}$  . Exemple : si  $x=2/5$  alors  $x^4$  retourne le rationnel 16/625
- une surcharge de l'opérateur d'exponentiation sous la forme rationnel operator  $^{(rationnel\&, int \&)}$  . Exemple : si  $x=2/5$  alors  $x^{-4}$  retourne le rationnel 625/16
- une surcharge de l'opérateur << qui permet d'afficher le numérateur et le dénominateur

## Exercice 7

Créer une classe nommée histo permettant de manipuler des « histogrammes ». Un histogramme est obtenu à partir d'un ensemble de valeurs  $x(i)$ , en définissant  $n$  tranches (intervalles) contiguës (souvent de même amplitude) et en comptabilisant le nombre de valeurs  $x(i)$  appartenant à chacune de ces tranches.

Prévoir :

- un constructeur dont les arguments précisent la borne minimale et maximale des valeurs à prendre en compte et les nombres d'intervalles de même amplitude.
- Un opérateur << défini tel que  $h<<xx$  ajoute la valeur  $x$  à l'histogramme  $h$  c'est-à-dire qu'elle incrémente de 1 le compteur relatif à la tranche à laquelle appartient  $x$ .
- Un opérateur [] défini de telle manière que  $h[i]$  représente le nombre de valeurs répertoriées dans la tranche  $i$ .

## Exercice 8

Créer une classe matrice qui permet de gérer des matrices à 2 dimensions. Elle comprend les données membres privées : nombre de ligne, nombre de colonne et une gestion dynamique pour stocker les éléments (doubles) de la matrice.

Elle comprend un constructeur et un destructeur.

Elle comprend la surcharge de l'opérateur <<, de l'opérateur(), de l'opérateur new et de l'opérateur delete .

L'opérateur () devra être utilisé pour stocker les éléments dans la matrice.

Créer un programme permettant de tester la classe matrice.

## Exercice 9

Écrire une classe vecteur comportant :

- en membres donnée privés : trois composantes de type double
- un constructeur qui initialise les valeurs des composantes
- une surcharge de \* permettant de multiplier les composantes par une valeur
- une surcharge de == permettant de comparer 2 vecteurs
- une surcharge des opérateurs >> et <<.

Créer un programme permettant de tester la classe vecteur.

## Exercice 1

```
#include <iostream.h>
class operation{

    float num;
    float deno;
public:
    operation(float numerateur=1, float denominateur=1);
    friend operation operator+(operation fct1, operation fct2);
    friend operation operator-(operation fct1, operation fct2);
    friend operation operator* (operation fct1, operation fct2);
    friend operation operator/(operation f1, operation f2);
    void affiche();
};

operation::operation (float numerateur, float denominateur){
    num= numerateur;
    deno=denominateur;
}
operation operator+ (operation fct1, operation fct2){ //ressemble a constructeur par
copie

operation res;
res.deno= fct1.deno*fct2.deno;
res.num=(fct1.num*fct2.deno)+(fct2.num*fct1.deno);
return res;
}

operation operator- (operation fct1, operation fct2){ //ressemble a constructeur par
copie
operation res;
res.deno= fct1.deno*fct2.deno;
res.num=(fct1.num*fct2.deno)-(fct2.num*fct1.deno);
return res;

}
operation operator* (operation fct1, operation fct2){ //ressemble a constructeur par
copie
operation res;
res.deno= fct1.deno*fct2.deno;
res.num=fct1.num*fct2.num;
return res;
}
operation operator/(operation f1, operation f2) {
    operation res;
    if(f2.num == 0) {

        res.num = 0;
        res.deno = 1;
        cout << "division par 0 pas possible" << endl;
    }
    else {
```

```

        res.deno = f1.deno * f2.num;
        res.num = f1.num * f2.deno;
    }
    return res;
}
// opposé ???????
void operation::affiche(){
    cout<<num<<" / "<<deno<<endl;
}
void main(){

    operation a(2,3);
    operation b(1,2);
    operation c;
    c=a+b;
    c.affiche();
    c=a-b;
    c.affiche();
    c=a*b;
    c.affiche();
    c=a/b;
    c.affiche();
}

```

## Exercice 2

```
#include <iostream.h>
```

```

class vecteur3d{

    float x;
    float y;
    float z;
public:
    vecteur3d(float x1=0, float y1=0, float z1=0);
    friend vecteur3d operator+(vecteur3d vect1, vecteur3d vect2);
    friend vecteur3d operator*(vecteur3d vect1, vecteur3d vect2);
    void affiche();
};

vecteur3d::vecteur3d(float x1, float y1, float z1){
    x=x1;
    y=y1;
    z=z1;
}

vecteur3d operator+(vecteur3d vect1, vecteur3d vect2){
    vecteur3d result;
    result.x=vect1.x+vect2.x;
    result.y=vect1.y+vect2.y;
    result.z=vect1.z+vect2.z;
}

```

```

        return result;}

vecteur3d operator*(vecteur3d vect1, vecteur3d vect2){
    vecteur3d result;
    result.x=vect1.x*vect2.x;
    result.y=vect1.y*vect2.y;
    result.z=vect1.z*vect2.z;
    return result;}

void vecteur3d::affiche(){

    cout<<"("<<x<<","<<y<<","<<z<<")";
}

void main(){

    vecteur3d a(2,3,4);
    vecteur3d b(1,2,5);
    vecteur3d c;
    c=a+b;
    c.affiche();
    c=a*b;
    c.affiche();
}

```

### Exercice 3

```

#include <iostream.h>

class date{

    int jj;
    int mm;

public:
    date(int jour=0, int mois=0); // pas oublier d'initialiser sinon ca pose probleme
    friend date operator-(date d1, date d2);
    friend date operator+(date d1,date d2);
    void lire();
    void affiche();
};

date::date(int jour, int mois){
jj=jour;
mm=mois;
}

date operator+(date d1, date d2){
    date result;
    result.jj=d1.jj+d2.jj;
    result.mm=d1.mm+d2.mm;
return result;
}

date operator-(date d1, date d2){

```

```

        date result;
        result.jj=d1.jj-d2.jj;
        result.mm=d1.mm-d2.mm;
return result;
}

void date::lire(){

    cout<<"Entrer le jour : "<<endl;
    cin>>jj;
    cout<<"Entrer la date : "<<endl;
    cin>>mm;

}

void date::affiche(){
    cout<<" le resultat de l'operation sur les dates "<<endl;
    cout<<jj<<" / "<<mm<<endl;
}

void main(){
    date a;
    date b;
    date c;

    a.lire();
    b.lire();

    c=a+b;
    c.affiche();
    c=a-b;
    c.affiche();

}

```

#### Exercice 4

// utilisation des operateurs de flux et d'incrementation et de decrementation et relationnels

```
#include <iostream.h>
```

```
class date{
```

```
    int jj;
    int mm;
```

```
public:
```

```
    date(int jour=0, int mois=0); // pas oublier d'initialiser sinon ca pose probleme
    friend date operator-(date d1, date d2);
    friend date operator+(date d1,date d2);
    friend istream& operator>>(istream& tmp,date& d);
    friend ostream& operator<<(ostream& tmp,date d);
    date operator --(int);
```

```

        date& operator --();
        date operator ++(int);
        date& operator ++();
        void lire();
        void affiche();
};

date::date(int jour, int mois){
    jj=jour;
    mm=mois;
}

date operator+(date d1, date d2){
    date result;
    result.jj=d1.jj+d2.jj;
    result.mm=d1.mm+d2.mm;
return result;
}

date operator-(date d1, date d2){
    date result;
    result.jj=d1.jj-d2.jj;
    result.mm=d1.mm-d2.mm;
return result;
}

istream& operator>>(istream& tmp,date& d){
    cout<<"Entrez la date svp"<<endl;
    tmp>>d.jj;
    tmp>>d.mm;
    return tmp;
}

ostream& operator<<(ostream& tmp,date d){
    cout<<endl;
    cout<<"Voici la date"<<endl;
    tmp<<d.jj;
    cout<<" / ";
    tmp<<d.mm;
return tmp;
}

date date ::operator--(int){// attention au placement du ::
    date tmp=*this;
    tmp.jj--;
    tmp.mm--;
    return tmp;
}

date& date::operator --(){
    jj--;
    mm--;
    return *this;
}

```

```
date date ::operator++(int){// attention au placement du ::
    date tmp=*this;
    tmp.jj++;
    tmp.mm++;
    return tmp;
}
```

```
date& date::operator ++(){
    jj++;
    mm++;
    return *this;
}
```

```
void date::lire(){

    cout<<"Entrer le jour : "<<endl;
    cin>>jj;
    cout<<"Entrer la date : "<<endl;
    cin>>mm;

}
```

```
void date::affiche(){
    cout<<" le resultat de l'operation sur les dates "<<endl;
    cout<<jj<<" / "<<mm<<endl;
}
```

```
void main(){
    date a;
    date b;
    date c;

    a.lire();
    //b.lire();
    cin>>b;

    c=a+b;
    c.affiche();
    c=a-b;
    cout<<c;
    c--;
    --c;
    c.affiche();
    ++c; // on utilisera cette operation
    c++;
    c.affiche();

}
```