

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 $^{(\text{rationnel}\&, \text{unsigned } \&)}$. 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 $^{(\text{rationnel}\&, \text{int } \&)}$. Exemple : si $x=2/5$ alors x^{-4} retourne le rationnel $625/16$
- une surcharge de l'opérateur \ll 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 \ll défini tel que $h \ll x$ 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 \ll , 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 \gg et \ll .

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();
}

```

Exercice2

```
#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();

}

```

// 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();
}

```

Exercice5

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

```
#include <string.h>
```

```
class vecteur3d{
```

```

    /*float taille;
    float *adresse;
*/

```

```

float x; // en public pour mon operateur d'indexation
float y;
float z;
float adr[3];

```

```
public:
```

```

    vecteur3d(float x1=0, float y1=0, float z1=0);
    friend vecteur3d/*valeur de retour */ operator+(vecteur3d vect1, vecteur3d
vect2);

```

```

    friend vecteur3d/*valeur de retour */ operator-(vecteur3d vect1, vecteur3d
vect2);
    friend vecteur3d operator*(vecteur3d vect1, vecteur3d vect2);
    friend ostream& operator<<(ostream& tmp , vecteur3d v);
    friend istream& operator>>(istream& tmp, vecteur3d& v);
    float& operator[] (int i);
    vecteur3d& operator +=(vecteur3d& v);
    friend bool operator==(vecteur3d v1,vecteur3d v2);
    vecteur3d& operator=(vecteur3d v);

    void affiche();
    ~vecteur3d();
};

vecteur3d::vecteur3d(float x1, float y1, float z1){
    x=x1;
    y=y1;
    z=z1;;
    adr[0]=x;
    adr[1]=y;
    adr[2]=z;
}
vecteur3d::~vecteur3d(){
}
float& vecteur3d::operator[] (int i){

    return adr[i];

}

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;}

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& vecteur3d::operator +=(vecteur3d& v){
    x+=v.x;
    y+=v.y;
    z+=v.z;

    return *this;
}

bool operator==(vecteur3d v1,vecteur3d v2){
    if((v1.x==v2.x) && (v1.y==v2.y) && (v1.z==v2.z)) return true;
    else return false;
}

vecteur3d& vecteur3d::operator=(vecteur3d v){
    x=v.x;
    y=v.y;
    z=v.z;

    return *this;
}

ostream& operator<<(ostream& tmp, vecteur3d v){
    tmp<<"X = "<<v.x<<" , Y = "<<v.y<<" , Z = "<<v.z<<endl;
    return tmp;
}

istream& operator>>(istream& tmp, vecteur3d& v){
    //pas de vecteur3d::
    cout<< " Entrez les valeurs de x,y,z"<<endl;
    tmp>>v.x;
    tmp>>v.y;
    tmp>>v.z;

    return tmp;
}

void vecteur3d::affiche(){

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

```

```

void main(){

    //vecteur3d a(2,3,4);
    vecteur3d a;
    vecteur3d b;
    vecteur3d d(1,1,1);
    vecteur3d c;
/*
    a.x=a[0];
    a.y=a[1];
    a.z=a[2];
    for (i=0;i<3;i++){
        cin>>a[i];
    }
    for (i=0;i<3;i++){
        cout<<a[i]<<endl;
    }
*/

    cin>>a;
    cout<<a;
    cin>>b;
    if(a==b){
        cout<<"Valeurs identiques"<<endl;
    }
    else{
        cout<<"Valeurs différentes"<<endl;
    }

    c=b-d;
    //c.affiche();
    cout<<c;
    c=a+b;
    c.affiche();
    c=a*b;
    c.affiche();
    a+=b;
    a.affiche();
    a=b;
    a.affiche();
    vecteur3d k(1,1,2);
    for (float i=0;i<3;i++){
        cout<<k[i]<<endl;
    }

}

```

Exercice6

```
#include <iostream.h>
#include <math.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);
    friend operation operator~(operation f);
    operation operator++(int); //post
    operation& operator++(); //pré
    operator float();
    friend operation operator^(operation op,unsigned &);
    friend operation operator^(operation op, int& ex);
    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;
}

```

```

operation operator~(operation f){
operation res;
res.num=-f.num;
res.deno=f.deno;
return res;
}

```

```

operation& operation::operator ++(){
    num++;
    deno++;
    return *this;
}

```

```

operation operation::operator ++(int){
    operation tmp=*this;
    tmp.num++;
    tmp.deno++;
    return tmp;
}

```

```

operation::operator float(){
float res=num/deno;
return res;
}

```

```

operation operator^(operation op, unsigned& ex){
operation tmp;
tmp.num=pow(op.num,ex);
tmp.deno=pow(op.deno,ex);
return tmp;
}

```

```

operation operator^(operation op, int& ex){
operation tmp;

tmp.num=pow(op.num,ex);
tmp.deno=pow(op.deno,ex);
return tmp;

}

void operation::affiche(){
    cout<<num<<" / "<<deno<<endl;
}

// surcharge de l'opérateur << --> voir les exo précédents

void main(){

    operation a(2,3);
    operation b(2,5);
    operation c;
    c=a+b;
    c.affiche();
    c=a-b;
    c.affiche();
    c=a*b;
    c.affiche();
    c=a/b;
    c.affiche();
    c=~a;
    c.affiche();
    c=a++;
    c.affiche();
    a.affiche();
    c=++a;
    c.affiche();
    a.affiche();

    float h;
    h=float(a);
    cout<<"conversion "<<h<<endl;
    unsigned int g=4;
    c=operator^(b,g);
    c.affiche();
    int j=(-4);
    c=operator^(b,j);
    c.affiche();

}

```

```

//'' Opérateurs de pré-incrémentation (pré-décrémentaion): ++var, --var
//'' Opérateurs de post-incrémentation (post-décrémentaion): var++, var--
//Il existe deux types de conversions de types :
// la conversion de type prédéfini (ou défini préalablement) vers le type
//classe en question. Ceci sera effectué grâce au constructeur de conversion.
// la conversion du type classe vers un type prédéfini (ou défini préalablement).
//Cette conversion sera effectuée par des fonctions de conversion.

```

Exercice 7

Exercice 8

```

#include <iostream.h>
#include<stddef.h> // utile pour size_t
#include<stdlib.h>

```

```

class matrice{

```

```

    int nbres_lg;
    int nbres_cl;
    int *tab;

```

```

public:

```

```

    matrice(int nblg,int nbcl);
    friend ostream& operator<<( ostream& tmp,matrice d);
    int* operator()(matrice& d);
    void * operator new (size_t);
    void operator delete(void *p);
    void affiche();

```

```

};

```

```

matrice::matrice(int nblg,int nbcl){

```

```

    nbres_lg=nblg;
    nbres_cl=nbcl;
    tab=new int[nbres_lg*nbres_cl];
    for (int i=0;i<nbres_lg*nbres_cl;i++) tab[i]=0;
}

```

```

ostream& operator<<(ostream& tmp, matrice d){

```

```

    for (int i=0; i<d.nbres_lg;i++){
        for(int j=0;j<d.nbres_cl;j++){
            tmp<<"tab["<<i<<"]["<<j<<"]="<<d.tab[(2*i*(d.nbres_cl-
1))+j]<<endl;

```

```

//voir cours de Mr Wilfart

```

```

        //sur l'allocation des tableaux
    }
    }
    return tmp;
}

int* matrice::operator ()(matrice& d){

    for (int l=0;l<d.nbres_lg;l++){
        {
            for(int c=0;c<d.nbres_cl;c++){
                {
                    cout<<"entrez la valeur que vous desirez pour la
matrice["<<l<<" "<<c<<"]:"<<endl;
                    cin>>d.tab[(2*l*(d.nbres_cl-1))+c];
                }
            }
        }
        return d.tab;
    }
}

void * matrice::operator new(size_t sz){
    cout<<"appel new"<<endl;
    return malloc(sz);}

void matrice::operator delete(void *p){
    free(p);
    cout<<"appel delete\n";
}

void matrice::affiche(){

    for (int l=0;l<nbres_lg;l++){
        {
            for(int c=0;c<nbres_cl;c++){
                {
                    cout<<tab[(2*l*(nbres_cl-1))+c]<<endl;
                }
            }
        }
    }
}

void main(){

matrice a(2,2);

a(a);
a.affiche();
cout<<a;
}

```

```

matrice *b=new matrice(a); // en paramètre de ou il va chercher sa reference
// c a d d'une autre matrice
cout<<a;
delete b;
}

```

Exercice 9

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

```

class vecteur{
    double x;
    double y;
    double z;

public:
    vecteur(double x1=0, double y1=0, double z1=0); //ATTENTION;-)
    friend vecteur operator*(vecteur v1, vecteur v2);
    friend bool operator==(vecteur v1, vecteur v2);
    friend ostream& operator<<(ostream& tmp,vecteur v);
    friend istream& operator>>(istream& tmp,vecteur& v);
};

vecteur::vecteur(double x1, double y1, double z1){
    x=x1;
    y=y1;
    z=z1;
}

vecteur operator*(vecteur v1, vecteur v2){
    vecteur tmp;
    tmp.x=v1.x*v2.x;
    tmp.y=v1.y*v2.y;
    tmp.z=v1.z*v2.y;

    return tmp;
}

ostream& operator<<(ostream& tmp,vecteur v){
    tmp<<"X = "<<v.x<<"Y = "<<v.y<<"Z = "<<v.z<<endl;
    return tmp;
}

istream& operator>>(istream& tmp,vecteur& v){
    cout<<" Entrez les valeurs de x,y et z"<<endl;
    tmp>>v.x;
    tmp>>v.y;
    tmp>>v.z;
    return tmp;
}

bool operator==(vecteur v1, vecteur v2){

```

```

        if ((v1.x==v2.x)&&(v1.y==v2.y)&&(v1.z==v2.z)) return true;
        else return false;
    }

void main(){
    vecteur a(2,3,4);
    vecteur b(2.2,3.3,4);
    vecteur c;
    c=a*b;
    vecteur d;
    cin>>d;
    cout<<d;
    if (a==d) cout<<"valeurs identiques"<<endl;
    else cout<<"valeurs différentes"<<endl;
}

```

Exercices sur l'héritage

Exercice 1 :

Créer une classe personnel comprenant 3 données membres (nom, position, salaire), un constructeur initialisant les données membres et une fonction affiche_personnel qui affiche les informations sur le personnel.

Créer une classe patron dérivant publiquement de la classe personnel et ayant 2 données membres (bonus_annuel, marque de voiture de fonction) , un constructeur initialisant les données membres et une fonction affiche_patron qui affiche les informations sur le patron (entant que membre du personnel et patron).

Créer un programme permettant de tester ces classes.

Exercice 2 :

a) Modifier les classes définies dans l'exercice 1 en utilisant la fonction affiche dans la classe de base et la classe dérivée.

Modifier le programme pour tester les modifications.

b) Créer dans le programme principal un pointeur sur la classe personnel et un objet de la classe patron. Le pointeur sur la classe de base contient l'adresse de la classe dérivée. Appeler la fonction affiche en utilisant le pointeur sur la classe de base (il faut qu'elle affiche les informations sur la classe dérivée).

Exercice 3:

Quels résultats produira le programme ci-dessous ?

```

#include <iostream.h>
class point{
protected:
    int x,y;
public :
    point (int abs=0,int ord=0){x=abs ;y=ord ;}
    virtual void affiche(){
        cout<< "point de coordonnées " <<x<< " et " <<y<<endl ;
    }
}

```

```

};
class pointcol:public point{
    short couleur;
public:
    pointcol(int abs=0,int ord=0, short cl=1) :point(abs,ord){ couleur=cl ;}
    void affiche(){
        cout<< "point de coordonnées " <<x<< " et " <<y;
        cout<< "et de couleur " <<cl<<endl ;
    }
};
void main(){
    point p(3,5);point *adp=&p;
    pointcol pc(8,6,2);pointcol *adpc=&pc;
    adp->affiche();adpc->affiche();
    adp=adpc ;
    adp->affiche() ;adpc->affiche() ;
}

```

Exercice 4 :

Qu'affiche le programme ? Est-ce correct ? Justifier et corriger si nécessaire.

```

#include <iostream.h>
class c{
public:
    c(){cout<<"c::c()"<<endl;}
    c(const c& ce){cout<<"c::c(const c& ce)"<<endl;}
};
class d:public c{
public:
    d(){cout<<"d::d()"<<endl ;}
    d(const d& dd){cout<<"d::d(const d& dd)"<<endl;}
};
void main(){
    d d1;
    d d2(d1);
}

```

Exercice 5:

Créer une classe animal, puis des classes dérivées à poils et à plumes où vous dérivez les particularités respectives de chaque animal. Dans la classe de base, implémentez sous forme de méthodes virtuelles les fonctions qui concernent l'animal en général mais s'appliquent différemment selon le type d'animal.

Exercice 6 :

Créer une classe nombre formée d'une unique variable entière et d'une méthode affiche_nombre, puis définir 3 classes dérivées afin d'afficher la variable aux formats hexadécimal, octal et décimal à l'aide d'une méthode affiche_nombre.

Exercice 7 :

- a) Créer une classe Date ayant 3 paramètres jour, mois, année. Elle contient un constructeur initialisant les données membres. Elle contient les fonctions getDay(), getMonth(), getYear(), setDay(), setMonth(), setYear() et DateToString()
Le constructeur attend en paramètre l'année, le mois puis le numéro de jour et les affecte dans l'espace de données privé de l'objet.
Les méthodes accesseurs permettent respectivement de récupérer le jour, le mois

ou l'année sous forme d'un entier (méthodes getxxx) et de changer ces mêmes propriétés (méthodes setxxx). La méthode DateToString retourne une chaîne de caractères représentant la date de l'objet sur lequel a été invoquée cette méthode au format américain.

b) Nous voyons que le constructeur et la méthode DateToString travaillent sur le format américain. Comme cette classe est intégrée dans les bibliothèques d'un environnement de développement, il n'est pas possible de la modifier. Quelle technique nous offre la programmation orientée objet pour pouvoir gérer nos dates au format français tout en utilisant cette classe existante qui offre en outre un ensemble d'autres services ? Écrire le code correspondant pour gérer un constructeur et une méthode retournant une chaîne au format français.

Exercice 8

Créer une classe Etudiant ayant les mêmes caractéristiques qu'une Personne plus une liste de matières suivies et un ensemble de notes. On doit pouvoir obtenir la liste des matières, affecter une note, calculer la moyenne.

Créer une classe Enseignant ayant les mêmes caractéristiques qu'une Personne plus une liste de matières enseignées, un nombre d'heures par matière et un tarif horaire. On doit pouvoir obtenir les matières, faire un calcul du nombre d'heures total et déterminer le coût de l'enseignant.

Les fonctions membres de Etudiant et de Enseignant doivent pouvoir manipuler directement les données de la classe Personne. Modifier en conséquence cette classe.

Créer un tableau d'Etudiant et un tableau d'Enseignant. Les initialiser. Créer un tableau de pointeurs de Personne et affecter selon votre choix à ces personnes soit un étudiant soit un enseignant. Afficher l'ensemble des caractéristiques de ces personnes "polymorphées". Les détruire.

Exercice 1

```
#include <iostream.h>
#include <string.h>
class personnel{
    char *nom;
    char *position;
    int salaire;
public:
    personnel(char *nm, char *pos, int sal);
    void affiche_personnel();
};
class patron:public personnel{/*class sousClasse: droit superClasse{ };*/
    int bonus_annuel;
    char* marque_voiture;
public :
    //constructeur
    //sousClasse(liste des paramètres (avec ceux de la super classe); //déclaration
    //sousClasse::sousClasse(liste de paramètres) : superClasse(paramètres pour
superClasse){ } //définition
    patron(int b_a, char* m_v,char *nm, char *pos, int sal);
```

```

        void affiche_patron();
    };
    personnel::personnel(char *nm, char *pos, int sal){
    cout<<"constructeur"<<endl;
    salaire=sal;
    nom=nm;
    position=pos;
    }
    void personnel::affiche_personnel(){
        cout<<"Nom : "<<nom<<endl;
        cout<<"Position : "<<position<<endl;
        cout<<"Salaire : "<<salaire<<endl;
    }
    patron::patron(int b_a, char* m_v,char *nm, char *pos, int sal):personnel(nm, pos,sal)
    {
    bonus_annuel=b_a;
    marque_voiture=m_v;
    }
    void patron::affiche_patron(){
        affiche_personnel(); //pas oublier
        cout<<"bonus annuel = "<<bonus_annuel<<endl;
        cout<<"Marque de voiture = "<<marque_voiture<<endl;
    }
    void main(){
        personnel b("gerald","gérant",1250);
        patron a(500,"BMW","fallon","PDG",2000);
        b.affiche_personnel();
        a.affiche_patron();
    }

```

Exercice 2

```

#include <iostream.h>
#include <string.h>
class personnel{
    char *nom;
    char *position;
    int salaire;
public:
    personnel(char *nm, char *pos, int sal);
    void affiche();
};
class patron:public personnel{/*class sousClasse: droit superClasse{ };*/
    int bonus_annuel;
    char* marque_voiture;
public :
    //constructeur
    //sousClasse(liste des paramètres (avec ceux de la super classe); //déclaration
    //sousClasse::sousClasse(liste de paramètres) : superClasse(paramètres pour
    superClasse){} //définition

```

```

        patron(int b_a, char* m_v, char *nm, char *pos, int sal);
        void affiche();
};
personnel::personnel(char *nm, char *pos, int sal){
cout<<"constructeur"<<endl;
salaire=sal;
nom=nm;
position=pos;
}
void personnel::affiche(){
    cout<<"Nom : "<<nom<<endl;
    cout<<"Position : "<<position<<endl;
    cout<<"Salaire : "<<salaire<<endl;
}
patron::patron(int b_a, char* m_v, char *nm, char *pos, int sal):personnel(nm, pos,sal)
{
bonus_annuel=b_a;
marque_voiture=m_v;
}
void patron::affiche(){
    //cout<<"Nom : "<<nom<<endl;
    //cout<<"Position : "<<position<<endl;
    //cout<<"Salaire : "<<salaire<<endl;
    personnel::affiche();
    // si on ne veut pas utiliser la résolution de portée
    //on mets nos données en mode protected
    cout<<"bonus annuel = "<<bonus_annuel<<endl;
    cout<<"Marque de voiture = "<<marque_voiture<<endl;
}
void main(){
    /*
    personnel b("gerald", "gérant",1250);
    patron a(500,"BMW", "fallon", "PDG",2000);
    b.affiche();
    a.affiche();
    */

    personnel *pe=new personnel("Fallon", "PDG",2000);
    pe->affiche();
    patron *pa=new patron(500,"PORSCHE", "Toujirat", "HautPDG",2250);
    pa->affiche();

}

```

Exercice 3

```

#include <iostream.h>
class point{
protected:
    int x,y;
public :
    point (int abs=0,int ord=0){x=abs ;y=ord ;}
    virtual void affiche(){
        cout<< "point de coordonnées " <<x<< "et " <<y<<endl ;
    }
};
class pointcol:public point{
    short couleur;
public:
    pointcol(int abs=0,int ord=0, short cl=1) :point(abs,ord){couleur=cl ;}
    void affiche(){
        cout<< "point de coordonnées " <<x<< " et " <<y;
        cout<< "et de couleur " <<couleur<<endl ;
    }
};
void main(){
    point p(3,5);point *adp=&p;
    pointcol pc(8,6,2);pointcol *adpc=&pc;
    adp->affiche();adpc->affiche();
    adp=adpc ;
    adp->affiche() ;adpc->affiche() ;
}

/*      Problème: Si une classe de base comprend une méthode avec un nom
identique
//et une même signature que la méthode de la classe dérivée, tout appel à cette
méthode
//par un pointeur de la classe de base fait référence à la méthode de la classe de base
/*      Solution: les fonctions virtuelles (fonctions dont le nom est précédé de
VIRTUAL)
/*      Polymorphisme: les objets peuvent changer de forme au cours de leur
exécution

```

Exercice 4

```

#include <iostream.h>
class c{
public:

```

```

        c(){cout<<"c::c()"<<endl;}
        c(const c& cc){cout<<"c::c(const c& cc)"<<endl;}
};
class d:public c{
    public:
        d(){cout<<"d::d()"<<endl ;}
        d(const d& dd){cout<<"d::d(const d& dd)"<<endl;}
};
void main(){
    d d1;
    d d2(d1);
}

```

Exercice 5

```

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

```

```

class animal{
protected:
    char nom[20];
public:
    animal(char n[20]="");
    virtual void manger();
};

class poils:public animal{
    int pattes;
public:
    poils(int pt=0,char n[20]="");
    void manger();
};

class plumes:public animal{
    int pattes;
public:
    plumes(int pt=0, char n[20]="");
    void manger();
};
animal::animal(char n[20]){
    strcpy(nom,n);
}
void animal::manger(){
    cout<<" l'animal suivant : "<<nom<<" a besoin de manger pour
survivre"<<endl;
}

poils::poils(int pt,char n[20]):animal(n){

```

```

        pattes=pt;
    }
void poils::manger(){
    cout<<"l'animal suivant : "<<nom<<" est carnivore"<<endl;
}

plumes ::plumes(int pt,char n[20]):animal(n){
pattes=pt;
}
void plumes::manger(){
    cout<<"l'animal suivant : "<<nom<<" n'est pas carnivore et aime les
graines"<<endl;
}

void main(){
    animal p("tortue");
    p.manger();
    poils q(4,"chat");
    q.manger();
    plumes s(2,"pigeon");
    s.manger();
    animal *r;
    r=&s;
    r->manger();
}

```

Exercice 6

```

#include <iostream.h>

class nombre{
protected:
    int entier;
public:

    nombre(int ent);
    virtual void affiche_nombre();
};

class hexa:public nombre{
public:
    hexa(int ent);
    void affiche_nombre();
};

class octal:nombre{
public:
    octal(int ent);

```

```

        void affiche_nombre();
};

class decimal:nombre{
public:
    decimal(int ent);
    void affiche_nombre();

};

nombre::nombre(int ent){
    entier=ent;
}

void nombre::affiche_nombre(){
    cout<<"voici le nombre : "<<entier<<endl;
}

hexa::hexa(int ent):nombre(ent){ }
void hexa::affiche_nombre(){
    cout <<" le nombre en hexa est de : "<<hex<<entier<<endl;
}

octal::octal(int ent):nombre(ent){ }
void octal::affiche_nombre(){
    cout <<" le nombre en octal est de : "<<oct<<entier<<endl;
}

decimal::decimal(int ent):nombre(ent){ }
void decimal::affiche_nombre(){
    cout <<" le nombre en décimal est de : "<<dec<<entier<<endl;
}

void main(){
    nombre a(12);
    a.affiche_nombre();
    decimal b(12);
    b.affiche_nombre();
    octal c(12);
    c.affiche_nombre();
    hexa d(12);
    d.affiche_nombre();
}

```

Exercice 7

```
#include <iostream.h>
#include <string>
#include <stdlib.h>
using namespace std;

class date{
protected:
    int jour;
    int mois;
    int annee;

public:
    date(int jr=0, int ms=0, int an=0);
    void getday();
    void getmonth();
    void getyear();
    void setday();
    void setmonth();
    void setyear();
    virtual string datetostring();
};

class datefr:public date{
public:
    datefr(int jr=0, int ms=0, int an=0);
    string datetostring();
};

datefr::datefr(int jr, int ms, int an):date(jr,ms,an){ }
string datefr::datetostring(){
    char tab1[60]="";
    char tab2[20]="";
    char tab3[20]="";
    cout<<"date en version europeenne"<<endl;
    itoa(jour,tab1,10);
    itoa(mois,tab2,10);
    itoa(annee,tab3,10);
    strcat(tab1,tab2);
    //cout<<tab1<<endl;
    strcat(tab1,tab3);
    //cout<<tab1<<endl;
    return tab1;
}

date::date(int jr, int ms, int an){
    jour=jr;
    mois=ms;
```

```

        annee=an;
    }
    void date::getday(){
        cout<<"Jour :"<<jour;
    }
    void date::getmonth(){
        cout<<"Mois :"<<mois;
    }
    void date::getyear(){
        cout<<"Annee :"<<annee<<endl;
    }
    void date::setday(){
        cout<<"Entrer le jour"<<endl;
        cin>>jour;
    }
    void date::setmonth(){
        cout<<"Entrer le mois"<<endl;
        cin>>mois;
    }
    void date::setyear(){
        cout<<"Entrer l' annee"<<endl;
        cin>>annee;
    }
}

```

```

string date::datetosttring(){
    char tab1[60]="";
    char tab2[20]="";
    char tab3[20]="";
    itoa(annee,tab1,10);
    itoa(mois,tab2,10);
    itoa(jour,tab3,10);
    strcat(tab1,tab2);
    //cout<<tab1<<endl;
    strcat(tab1,tab3);
    //cout<<tab1<<endl;
    return tab1;
}

```

```

void main(){
    date a;
    a.setday();
    a.setmonth();
    a.setyear();
    a.getday();
    a.getmonth();
    a.getyear();
    string tab= a.datetosttring();
    for(int i=0;i<20;i++)

```

```

        cout<<tab[i];
        cout<<endl;
        datefr b(22,9,81);
        string tab1=b.datetostring();
        for(int j=0;j<20;j++)
        cout<<tab1[j];
        cout<<endl;
    }

```

Exercice 8

```

#include <iostream.h>
#include <string.h>
void menu_etudiant();
void menu_enseignant();

class personne {
    char nom[20];
    char prenom[20];
    int age;
public:
    personne(char n[20]="",char p[20]="",int a=0) {
        strcpy(nom,n);
        strcpy(prenom,p);
        age=a;
    }
    void identification() {
        cout<<"Entrez le nom de la personne : ";
        cin>>nom;
        cout<<"Entrez le prenom de la personne : ";
        cin>>prenom;
        cout<<"Entrez l'age de la personne : ";
        cin>>age;
    }
    void affiche() {
        cout<<"Nom : "<<nom<<endl;
        cout<<"Prenom : "<<prenom<<endl;
        cout<<"Age : "<<age<<endl;
    }
};

class etudiant:personne {
    char cours[5];
    int *notes;
public:
    etudiant(char n[20],char p[20],int a,char c[5],int *no):personne(n,p,a){
        strcpy(cours,c);
        notes=new int [5];
    }
    void affect_etudiant() {
        personne::identification();
    }

```

```

        menu_etudiant();
        for(int i=0;i<5;i++) {
            cout<<"Quelle est la matiere ? ";
            cin>>cours[i];
            if(cours[i]=='x') break;
            cout<<"Notes obtenues ? ";
            cin>>notes[i];
        }
    }
    void affiche() {
        float moy=0;
        personne::affiche();
        cout<<"Cours suivis et notes : "<<endl;
        for(int i=0;i<5;i++) {
            if(cours[i]=='p') cout<<"Programmation"<<"
"<<notes[i]<<endl;
            if(cours[i]=='e') cout<<"Electronique"<<" "<<notes[i]<<endl;
            if(cours[i]=='r') cout<<"Reseau"<<" "<<notes[i]<<endl;
            if(cours[i]=='t') cout<<"Technologie des composants
electroniques"<<" "<<notes[i]<<endl;
            if(cours[i]=='x') break;
            moy=moy+notes[i];
        }
        cout<<"Moyenne de l'etudiant : "<<moy/4<<endl;
    }
};
class enseignant:public personne {
    char matieres[5];
    int *nbre_heure;
    int tarif_horraire;
public:
    enseignant(char n[20],char p[20],int a,char mat[5],int *heures,int
tarif):personne(n,p,a) {
        strcpy(matieres,mat);
        nbre_heure=new int [5];
        tarif_horraire=tarif;
    }
    void affect_enseignant() {
        personne::identification();
        menu_enseignant();
        for(int i=0;i<5;i++) {
            cout<<"Une matiere enseignee : ";
            cin>>matieres[i];
            if(matieres[i]=='x') break;
            cout<<"Nombre d'heures pour ce cours : ";
            cin>>nbre_heure[i];
        }
        cout<<"Tarif horraire de cet enseignant : ";
        cin>>tarif_horraire;
    }
}

```

```

void affiche() {
    int total_heure=0,total_salaire=0;
    personne::affiche();
    cout<<"Cours donnees et nombre d'heures : "<<endl;
    for(int i=0;i<5;i++) {
        if(matieres[i]=='p') cout<<"Programmation"<<"
"<<nbre_heure[i]<<endl;
        if(matieres[i]=='e') cout<<"Electronique"<<"
"<<nbre_heure[i]<<endl;
        if(matieres[i]=='r') cout<<"Reseau"<<"
"<<nbre_heure[i]<<endl;
        if(matieres[i]=='t') cout<<"Technologie des composants
electroniques"<<" "<<nbre_heure[i]<<endl;
        if(matieres[i]=='m') cout<<"Laboratoire de micro
controlleur"<<" "<<nbre_heure[i]<<endl;
        if(matieres[i]=='o') cout<<"Micro ordinateur"<<"
"<<nbre_heure[i]<<endl;
        if(matieres[i]=='x') break;
        total_heure=total_heure+nbre_heure[i];
    }
    total_salaire = total_heure * tarif_horraire;
    cout<<"Nombre d'heures donnees par l'enseignant :
"<<total_heure<<endl;
    cout<<"Salaire de l'enseignant : "<<total_salaire<<endl;
}
};
void main() {
    cout<<"Etudiant : "<<endl;
    etudiant etu1("Touijrat","Abdel",29,"pe",0);
    etu1.affect_etudiant();
    etu1.etudiant::affiche();
    cout<<"Enseignant : "<<endl;
    enseignant ens1("Dedecker","Jeff",45,"e",0,13);
    ens1.affect_enseignant();
    ens1.enseignant::affiche();
}
void menu_etudiant() {
    cout<<"Entrez 'p' pour programmation"<<endl;
    cout<<"Entrez 'e' pour electronique"<<endl;
    cout<<"Entrez 't' pour tce"<<endl;
    cout<<"Entrez 'r' pour reseau"<<endl;
    cout<<"Pour arreter, tapez x"<<endl;
}
void menu_enseignant() {
    cout<<"Entrez 'p' pour programmation"<<endl;
    cout<<"Entrez 'e' pour electronique"<<endl;
    cout<<"Entrez 't' pour tce"<<endl;
    cout<<"Entrez 'r' pour reseau"<<endl;
    cout<<"Entrez 'm' pour laboratoire de microcontrolleur"<<endl;
    cout<<"Entrez 'o' pour micro ordinateur"<<endl;
}

```

```
        cout<<"Pour arreter, tapez x"<<endl;
    }
// exo en provenance de Didier
```

Exercices sur l'héritage multiple

Exercice 1

- Créer une classe volante qui a comme donnée membre nombre_ailer et qui a un constructeur, un destructeur et une fonction affiche
- Créer une classe animal qui a comme données membres nombre_pattes et type_pelage et qui a un constructeur, un destructeur et une fonction affiche.
- Créer une classe oiseau qui dérive publiquement des classes volante et animal. Elle a comme donnée membre nombre_œufs et qui a un constructeur, un destructeur et une fonction affiche qui affiche la donnée membre et qui fait appel aux fonctions affiche des classes de base.
- Créer un programme qui crée un objet de type oiseau et teste ses fonctions.
- Faire le diagramme des classes en UML

Exercice 2

- Créer une classe moniteur comprenant :
 - les données membres : type (chaîne de caractères), couleurs (long), x_reso(int) et y_reso(int)
 - un constructeur initialisant les données membres
 - une fonction montrer_moniteur qui affiche les informations sur le moniteur
- Créer une classe carte_mere comprenant :
 - les données membres : processeur(int), vitesse(int) et ram(int)
 - un constructeur initialisant les données membres
 - une fonction montrer_carte qui affiche les informations sur la carte
- Créer une classe ordinateur dérivant publiquement des classes moniteur et carte_mere et qui contient :
 - les données membres : nom(chaînes de caractères), hd(int), lecteur(float)
 - un constructeur initialisant les données membres
 - une fonction montrer_ordinateur qui affiche les informations sur l'ordinateur et appelle les fonctions montrer_moniteur et montrer_carte.
- Créer un programme qui crée un objet ordinateur et qui affiche les informations sur cet objet
- Remplacer les fonctions montrer_moniteur, montrer_carte et montrer_ordinateur par 3 fonctions appelées montrer. Faire les modifications nécessaires au bon fonctionnement du programme.

Exercice 3

Créer une classe poste_de_travail dérivant publiquement de la classe ordinateur (voir exercice 2) Elle contient :

- la donnée membre syst_exploi (chaîne de caractères)

- un constructeur initialisant la donnée membre
- une fonction montrer qui affiche les informations sur le poste de travail en faisant appel à la fonction montrer de la classe ordinateur.

Faire le diagramme des classes de l'exercice 2 et 3

Exercice 4

- a) Créer une classe objetassure comprenant :
 - les données membres : montant(float), type (chaîne de caractères)
 - un constructeur initialisant les données membres
 - une fonction affiche qui affiche les informations sur l'objetassure
 - b) Créer une classe ordinateur dérivant publiquement de la classe objetassure et comprenant :
 - les données membres : ram (int), hd(int), ...
 - un constructeur initialisant les données membres
 - une fonction affiche qui affiche les informations sur l'ordinateur et qui appelle la fonction affiche de la classe objetassure
 - c) Créer une classe bagage dérivant publiquement de la classe objetassuré et comprenant :
 - les données membres : type (chaîne de caractères), poids (float)
 - un constructeur initialisant les données membres
 - une fonction affiche affichant les informations sur le bagage et appelant la fonction affiche de la classe objetassure.
 - d) Créer une classe ordinateurportable dérivant publiquement des classes ordinateur et bagage et comprenant :
 - les données membres poids (float) et epaisseur (float)
 - un constructeur initialisant les données membres
 - une fonction affiche qui affiche les informations sur l'ordinateur portable et qui appelle les fonctions affiche des classes bagage et ordinateur
 - e) Créer un programme qui crée un objet ordinateurportable et qui affiche les informations sur cet objet
- Remarque : l'ordinateur portable doit bénéficier des 2 assurances, l'une contre le vol (en tant qu'ordinateur) et l'autre contre la perte (en tant que bagage).
- f) Faire le diagramme des classes.

Exercice 1

```
#include <iostream.h>
#include <string.h>
#define MAX 20
```

```
class volante{
    int nombre_ailes;
public:
    volante(int na=2);
    ~volante();
    void affiche();
```

```

};

class animal{
    int nombre_pattes;
    char type_pelage[MAX];
public:
    animal(int np=4,char *tp="");
    ~animal();
    void affiche();
};

class oiseau:public volante,public animal{
    int nombre_oeufs;
public:
    oiseau(int no=1,int na=2,int np=4,char *tp="");
    ~oiseau();
    void affiche();
};

volante::volante(int na){
    nombre_ailles=na;
    cout<<"constructeur volante"<<endl;
}

volante::~~volante(){
    cout<<"destructeur volante"<<endl;
}

void volante::affiche(){
    cout<<"nombre d'ailles:"<<nombre_ailles<<endl;
}

animal::animal(int np,char *tp){
    nombre_pattes=np;
    strcpy(type_pelage,tp);
    cout<<"constructeur animal"<<endl;
}

animal::~~animal(){
    cout<<"destructeur animal"<<endl;
}

void animal::affiche(){
    cout<<"nombre de pattes:"<<nombre_pattes<<endl;
    cout<<"type de pelage: "<<type_pelage<<endl;
}

oiseau::oiseau(int no,int na,int np,char *tp):volante(na),animal(np,tp){
    nombre_oeufs=no;
    cout<<"constructeur oiseau"<<endl;
}

```

```

}

oiseau::~oiseau(){
    cout<<"destructeur oiseau"<<endl;
}

void oiseau::affiche(){
    volante::affiche();
    animal::affiche();
    cout<<"nombre d'oeufs: "<<nombre_oeufs<<endl;
}

void main()
{
    int k;
    oiseau o;
    o.affiche();
    cin>>k;
}

```

Exercice 2

//2.a,b,c,d

```

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

```

```

class moniteur{
    char type[5];
    long int couleur;
    int x_reso;
    int y_reso;
public:
    moniteur(char tp[5]="", long int coul=0, int x=0, int y=0);
    void montrer_moniteur();
};

```

```

class carte_mere{
    int processeur;
    int vitesse;
    int ram;
public:
    carte_mere(int pro=0, int vi=0, int rm=0);
    void montrer_carte();
};

```

```

class ordinateur:public moniteur, carte_mere{
    char nom[10];
    int hardisk;
    float lecteur;
public:
    ordinateur(char nm[10], int hd, float lct,char tp[5],long int coul, int x, int y,int
pro, int vi, int rm);
    //attention ne pas oublier
    void montrer_ordinateur();
};

moniteur::moniteur(char tp[5], int long coul, int x, int y){

    strcpy(type,tp);
    couleur=coul;
    x_reso=x;
    y_reso=y;
}

void moniteur::montrer_moniteur(){
    cout<<"type de moniteur: "<<type<<endl;
    cout<<"couleur: "<<couleur<<endl;
    cout<<"x_reso: "<<x_reso<<endl;
    cout<<"y_reso: "<<y_reso<<endl;
}

carte_mere::carte_mere(int pro, int vi, int rm){
    processeur=pro;
    vitesse=vi;
    ram=rm;
}

void carte_mere::montrer_carte(){
    cout<<"processeur: "<<processeur<<endl;
    cout<<"vitesse: "<<vitesse<<endl;
    cout<<"ram: "<<ram<<endl;
}

ordinateur::ordinateur(char nm[10], int hd, float lct,char tp[5],long int coul, int x, int
y,int pro, int vi, int rm): moniteur(tp,coul,x,y),carte_mere(pro,vi,rm){
//je mets public a chak fois... erreur de syntaxe
    strcpy(nom,nm);
    hardisk=hd;
    lecteur=lct;
}

void ordinateur::montrer_ordinateur(){
    moniteur::montrer_moniteur();
    carte_mere::montrer_carte();
    cout<<"nom du PC: "<<nom<<endl;
}

```

```

        cout<<"hd: "<<hardisk<<endl;
        cout<<"lecteur: "<<lecteur<<endl;
    }

void main()
{
int k;
    moniteur a("moniteur",256,800,600);
    a.montrer_moniteur();
    cout<<endl;
    carte_mere b(64,3,1024);
    b.montrer_carte();
    cout<<endl;
    ordinateur c("junior",160,5.4,"moniteur",256,800,600,64,3,1024);
    c.montrer_ordinateur();

cin>>k;
    }

```

```

//2.e
#include <iostream.h>
#include <string.h>

```

```

class moniteur{
    char type[5];
    long int couleur;
    int x_reso;
    int y_reso;
public:
    moniteur(char tp[5]="", long int coul=0, int x=0, int y=0);
    void montrer();
};

```

```

class carte_mere{
    int processeur;
    int vitesse;
    int ram;
public:
    carte_mere(int pro=0, int vi=0, int rm=0);
    void montrer();
};

```

```

class ordinateur:public moniteur, carte_mere{
    char nom[10];

```

```

        int hardisk;
        float lecteur;
public:
    ordinateur(char nm[10], int hd, float lct,char tp[5],long int coul, int x, int y,int
pro, int vi, int rm);
    //attention ne pas oublier
    void montrer();
};

moniteur::moniteur(char tp[5], int long coul, int x, int y){

    strcpy(type,tp);
    couleur=coul;
    x_reso=x;
    y_reso=y;
}

void moniteur::montrer(){
    cout<<"type de moniteur: "<<type<<endl;
    cout<<"couleur: "<<couleur<<endl;
    cout<<"x_reso: "<<x_reso<<endl;
    cout<<"y_reso: "<<y_reso<<endl;
}

carte_mere::carte_mere(int pro, int vi, int rm){
    processeur=pro;
    vitesse=vi;
    ram=rm;
}

void carte_mere::montrer(){
    cout<<"processeur: "<<processeur<<endl;
    cout<<"vitesse: "<<vitesse<<endl;
    cout<<"ram: "<<ram<<endl;
}

ordinateur::ordinateur(char nm[10], int hd, float lct,char tp[5],long int coul, int x, int
y,int pro, int vi, int rm): moniteur(tp,coul,x,y),carte_mere(pro,vi,rm){
//je mets public a chak fois... erreur de syntaxe
    strcpy(nom,nm);
    hardisk=hd;
    lecteur=lct;
}

void ordinateur::montrer(){
    moniteur::montrer();
    carte_mere::montrer();
    cout<<"nom du PC: "<<nom<<endl;
    cout<<"hd: "<<hardisk<<endl;
    cout<<"lecteur: "<<lecteur<<endl;
}

```

```
}
```

```
void main()
```

```
{
```

```
    moniteur a("moniteur",256,800,600);  
    a.montrer();  
    cout<<endl;  
    carte_mere b(64,3,1024);  
    b.montrer();  
    cout<<endl;  
    ordinateur c("junior",160,5.4,"moniteur",256,800,600,64,3,1024);  
    c.montrer();
```

```
}
```

Exercice 3

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

```
#include <string.h>
```

```
class moniteur{
```

```
    char type[5];
```

```
    long int couleur;
```

```
    int x_reso;
```

```
    int y_reso;
```

```
public:
```

```
    moniteur(char tp[5]="", long int coul=0, int x=0, int y=0);
```

```
    void montrer();
```

```
};
```

```
class carte_mere{
```

```
    int processeur;
```

```
    int vitesse;
```

```
    int ram;
```

```
public:
```

```
    carte_mere(int pro=0, int vi=0, int rm=0);
```

```
    void montrer();
```

```
};
```

```
class ordinateur:public moniteur, carte_mere{
```

```
    char nom[10];
```

```
    int hardisk;
```

```
    float lecteur;
```

```
public:
```

```

        ordinateur(char nm[10], int hd, float lct,char tp[5],long int coul, int x, int y,int
pro, int vi, int rm);
        //attention ne pas oublier
        void montrer();
};

```

```

class poste_travail:public ordinateur{
        char syst_exploi[10];
public:

```

```

    poste_travail(char s_e[10],char nm[10], int hd, float lct,char tp[5],long int coul, int x,
int y,int pro, int vi, int rm);
    void montrer();
};

```

```

    poste_travail::poste_travail(char s_e[10],char nm[10], int hd, float lct,char tp[5],long
int coul, int x, int y,int pro, int vi, int rm):ordinateur(nm,hd,lct,tp,coul,x,y,pro,vi,rm){
        strcpy(syst_exploi,s_e);
    }
    void poste_travail::montrer(){
        ordinateur::montrer();
        cout<<"Systeme d'exploitation : "<<syst_exploi<<endl;
    }

```

```

moniteur::moniteur(char tp[5], int long coul, int x, int y){

    strcpy(type,tp);
    couleur=coul;
    x_reso=x;
    y_reso=y;
}

```

```

void moniteur::montrer(){
    cout<<"type de moniteur: "<<type<<endl;
    cout<<"couleur: "<<couleur<<endl;
    cout<<"x_reso: "<<x_reso<<endl;
    cout<<"y_reso: "<<y_reso<<endl;
}

```

```

carte_mere::carte_mere(int pro, int vi, int rm){
    processeur=pro;
    vitesse=vi;
    ram=rm;
}

```

```

void carte_mere::montrer(){
    cout<<"processeur: "<<processeur<<endl;
    cout<<"vitesse: "<<vitesse<<endl;
    cout<<"ram: "<<ram<<endl;
}

```

```

}

ordinateur::ordinateur(char nm[10], int hd, float lct, char tp[5], long int coul, int x, int
y, int pro, int vi, int rm): moniteur(tp, coul, x, y), carte_mere(pro, vi, rm){
//je mets public a chak fois... erreur de syntaxe
    strcpy(nom, nm);
    hardisk=hd;
    lecteur=lct;
}
void ordinateur::montrer(){
    moniteur::montrer();
    carte_mere::montrer();
    cout<<"nom du PC: "<<nom<<endl;
    cout<<"hd: "<<hardisk<<endl;
    cout<<"lecteur: "<<lecteur<<endl;
}
}

```

```

void main()
{

    moniteur a("moniteur", 256, 800, 600);
    a.montrer();
    cout<<endl;
    carte_mere b(64, 3, 1024);
    b.montrer();
    cout<<endl;
    ordinateur c("junior", 160, 5.4, "moniteur", 256, 800, 600, 64, 3, 1024);
    c.montrer();
    cout<<endl;
    poste_travail d("XP", "junior", 160, 5.4, "moniteur", 256, 800, 600, 64, 3, 1024);
    d.montrer();

}

```

Exercice 4

```

#include <iostream.h>
#include <string.h>
#define MAX 20

class objetassure{
    float montant;
    char type[MAX];
public:
    objetassure(float mt=0.0, char tp[MAX]="");
    void affiche();
};

```

```

class ordinateur:public virtual objetassure{
    int ram;
    int hardisk;
public:
    ordinateur(int rm=0,int hd=0,float mt=0.0,char tp[MAX]="");
    void affiche();
};

class bagage:virtual public objetassure{
    char type1[MAX];
    float poids;
public:
    bagage(char tp1[MAX]="",float pd=0,float mt=0.0,char tp[MAX]="");
    void affiche();
};

class ordinateurportable:public ordinateur,public bagage{
    float poids1;
    float epaisseur;
public:
    ordinateurportable(float pd1=0.0,float epaiss=0.0,int rm=0,int hd=0,char
tp1[MAX]="",float pd=0.0);
    //pas obligation de mettre les autres paramètres de objetassure
    //ici je ne le fais pas mais pour afficher il vaudrait mieux ;- )
    void affiche();
};

objetassure::objetassure(float mt,char tp[MAX]){
    montant=mt;
    strcpy(type,tp);
}

void objetassure::affiche(){
    cout<<"montant: "<<montant<<endl;
    cout<<"objet assure type: "<<type<<endl;
}

ordinateur::ordinateur(int rm,int hd,float mt,char tp[MAX]):objetassure(mt,tp){
    ram=rm;
    hardisk=hd;
}

void ordinateur::affiche(){
    objetassure::affiche();
    cout<<"Mémoire type ram : "<<ram<<endl;
    cout<<"Disque Dur: "<<hardisk<<endl;
}

```

```

bagage::bagage(char tp1[MAX],float pd,float mt,char tp[MAX]):objetassure(mt,tp){
    strcpy(type1,tp1);
    poids=pd;
}

void bagage::affiche(){
    objetassure::affiche();
    cout<<"Type de bagage: "<<type1<<endl;
    cout<<"Poids du bagage: "<<poids<<endl;
}

ordinateurportable::ordinateurportable(float pd1,float epaiss,int rm,int hd,char
tp1[MAX],float pd):ordinateur(rm,hd),bagage(tp1,pd){
    poids1=pd1;
    epaisseur=epaiss;
}

void ordinateurportable::affiche(){
    ordinateur::affiche();
    bagage::affiche();
    cout<<"ordinateur portable poids: "<<poids1<<endl;
    cout<<"epaisseur: "<<epaisseur<<endl;
}

void main(){
    int k;
    ordinateurportable orpor(10,10,256,160,"a main",50);//attention a l'affichage
    orpor.affiche();
    cin>>k;
}

```

Exercices sur fonctions amies

Exercice 1 :

Soit une classe vecteur3d ayant 3 données membres privées, de type entier, les composantes du vecteur (x,y,z). Elle a un constructeur permettant d'initialiser les données membres.

Ecrire une fonction indépendante, coincide, amie de la classe vecteur3d, permettant de savoir si 2 vecteurs ont mêmes composantes.

Si v1 et v2 désignent 2 vecteurs de type vecteur3d, écrire le programme qui permet de tester l'égalité de ces 2 vecteurs.

Exercice 2 :

Créer 2 classe (dont les membres données sont privés) :

- l'une nommée vect, permettant de représenter des vecteurs à 3 composantes de type double ; elle comportera un constructeur et une fonction membre affiche.

- L'autre nommée matrice, permettant de représenter des matrices carrées de dimension 3x3 ; elle comportera un constructeur avec un argument (adresse d'un tableau de 3x3 valeurs) qui initialisera la matrice avec les valeurs correspondantes.

Réaliser une fonction indépendante prod permettant de fournir le vecteur correspondant au produit d'une matrice par un vecteur.

Ecrire un programme de test.

Fournir les 2 déclarations de chacune des classes, leurs définitions, la définition de prod et le programme de test.

Exercice 1

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

```
class vecteur3d{
```

```
    int x;
```

```
    int y;
```

```
    int z;
```

```
public:
```

```
    vecteur3d(int a=0,int b=0, int c=0);
```

```
    friend void coincide (vecteur3d p, vecteur3d q);
```

```
};
```

```
vecteur3d::vecteur3d(int a,int b,int c){
```

```
    x=a;
```

```
    y=b;
```

```
    z=c;
```

```
}
```

```
void coincide(vecteur3d p, vecteur3d q){
```

```
    if(p.x==q.x && p.y==q.y && p.z==q.z){
```

```
        cout<<"Les 2 vecteurs sont égaux"<<endl;
```

```
    }
```

```
    else cout<<"FOIREUX"<<endl;
```

```
}
```

```
void main(){
```

```
    vecteur3d v1(3,2,5);
```

```
    vecteur3d v2(3,4,5);
```

```
    coincide(v1,v2);
```

```
}
```

Exercice 2

// marche pas pq ???

```
#include <iostream.h>
#include <string.h>
class matrice;
class vect{
    double x;
    double y;
    double z;
public:
    vect(double a=0,double b=0,double c=0);
    void affiche();
    friend vect prod(matrice b, vect a);
};

class matrice{
double tableau[3][3];
public :
    //friend class a;

    matrice(double tab[3][3]=0);
    friend vect prod(matrice b, vect a);
};

vect::vect(double a,double b, double c){
    x=a;
    y=b;
    z=c;
}

void vect::affiche(){
    cout<<"X= "<<x<<endl;
    cout<<"Y= "<<y<<endl;
    cout<<"Z= "<<z<<endl;
}

matrice::matrice(double tab[3][3]){
    tableau[3][3]=tab[3][3];
    tableau[0][0]=1;
    tableau[0][1]=2;
    tableau[0][2]=3;
    tableau[1][0]=4;
```

```

    tableau[1][1]=5;
    tableau[2][1]=6;
    tableau[0][2]=7;
    tableau[1][2]=8;
    tableau[2][2]=9;
}

vect prod(matrice b, vect a){
    vect c;
    c.x=(b.tableau[0][0]*a.x)+(b.tableau[0][1]*a.y)+(b.tableau[0][2]*a.z);
    c.y=(b.tableau[1][0]*a.x)+(b.tableau[1][1]*a.y)+(b.tableau[1][2]*a.z);
    c.z=(b.tableau[2][0]*a.x)+(b.tableau[2][1]*a.y)+(b.tableau[2][2]*a.z);
    cout<<"le résultat de la multiplication est
:"<<c.x<<" "<<c.y<<" "<<c.z<<endl;
    return c;
}

void main(){
    vect vecteur(1,1,1);
    matrice mat;
    vect res;
    res=prod(mat,vecteur);
}

```

Exercices : entrées et sorties en c++

Exercice 1

```

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

void main(){
    double x;
    double y;
    cout<<"entrez un floatant: "<<endl;
    cin>>x;
    y=pow(x,2);
    cout<<y<<endl;

    cout<<"a la facon point fixe"<<endl;

    for(int i=0;i<11;i++)
    {
        cout<<i<<":";
        // cout.fill(' '); //met des espaces après le ':'
        cout<<y<<endl;
    }
}

```

```

        cout.setf(ios::fixed,ios::floatfield); //ios::fixed=notation"point fixe"

//ios::floatfield=conversion en flottant

//cout.setf=
    cout.precision(i);//ou cout<<setprecision(i) en manipulateur
    cout<<y<<endl;
}

cout<<"*****"<<endl;

cout<<"a la manière scientifique"<<endl;
for(int j=0;j<11;j++)
{
    cout<<j<<":";
    cout.fill(' '); //met des espaces après le ':'
    cout.setf(ios::scientific,ios::floatfield);
//ios::scientific=notation"scientifique"

//ios::floatfield=conversion en flottant

//cout.setf=
    cout.precision(j);
    cout<<y<<endl;
}
}

```

Exercice 2

```

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

#define MAX 20

class produit{
public:
    double prix;
    char libelle[MAX];
    produit(double prix=0,char *libelle="");
    friend bool operator <(produit prod1,produit prod2);
    friend bool operator >(produit prod1,produit prod2);
    friend ostream& operator<<(ostream& o,produit& pr);//si tu comprends pas ,
//va revoir la theorie des operateurs

};

```

```

produit::produit(double prix,char *libelle){
    this->prix=prix;
    strcpy(this->libelle,libelle);
}

bool operator <(produit prod1,produit prod2){
    if (prod1.prix<prod2.prix) return true;
    else return false;
}

bool operator >(produit prod1,produit prod2){
    if (prod1.prix>prod2.prix) return true;
    else return false;
}

ostream& operator<<(ostream& o,produit& pr){
    o.width(35);//largeur de 35 caracteres
    o.fill('.');//si pas on remplit
    o.setf(ios::left,ios::adjustfield);
    o<<"nom: "<<pr.libelle<<" prix: "<<setprecision(3)<<pr.prix<<endl;
    //precision de 3 caracteres apres la virgule
    return o;
}

void main(){
    produit p1(1.0F,"banane");
    produit p2(2.5F,"pomme");

    if(p1>p2) {
        cout<<"PLUS CHER =>\n"<<p1;
        cout<<"MOINS CHER =>\n"<<p2;
    }
    else{
        cout<<"PLUS CHER =>\n"<<p2;
        cout<<"MOINS CHER =>\n"<<p1;
    }
}

```

Exercice 3

```

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

class fraction
{
    int num,den;
public:

```

```

    fraction(int n=0,int d=0);
    fraction saisir(char* g);
    void affiche();
};

fraction::fraction(int n,int d)
{
    num=n;
    den=d;
}

fraction fraction::saisir(char* g)
{
    char x;
    fraction f;

    istream istr(g, 7);
    istr>>f.num>>x>>f.den;
    //on extrait d'abord le num puis un caractere puis le deno
    //cout<<f.num<<endl;
    return f;
}

void fraction::affiche()
{
    cout<<num<<"/"<<den<<endl;
}

int main()
{
    fraction f;
    char* g="324/345";
    f.affiche();

    f=f.saisir(g);
    f.affiche();
    return 0;
}

```

Exercice 4

```

#include <iostream.h>
#include <fstream.h>

void main (){
    int choix;
    char c;
    ofstream f("exo41.txt",ios::out|ios::app);
    while(choix!=0){

```

```

        cout<<"entrez un caractere: ";
        cin>>c;
        f<<c;
        /* cin>>c;
        file.write(&c,1)
        */

        cout<<"voulez-vous encore encoder un caractere? 1 pour oui, 0 pour
non: ";
        cin>>choix;
        f.close();
    }
}

```

Exerice 5

```

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

void main()
{
    char *nom=new char[200];
    char c='a';
    cout<<"entrez nom du fichier\n"<<endl;
    cin>>nom;
    cout<<nom;

    ofstream file;
    file.open(nom,ios::out);

    while(c!='0')
    {
        cin>>c;
        //if(c!='0')
            file.write(&c,1);
    }

    file.close();

    ifstream fichier;

    ofstream voyelle,consonne;

```

```

fichier.open(nom,ios::in);
voyelle.open("voyelle.txt",ios::out);

consonne.open("consonne.txt",ios::out);

        while(!fichier.eof())
        {
            c=fichier.get();

            if(c=='a'||c=='e'||c=='o'||c=='u'||c=='y')
            {
                voyelle.write(&c,1);
            }

            else
            {
                consonne.write(&c,1);
            }
        }

voyelle.close();
consonne.close();
fichier.close();

```

```

fstream pip;
char nm[200]="";
cout<<"entrez le nom du file txt que vs desirez ouvrir\n";
cin>>nm;
pip.open(nm,ios::in);

int k=0;
while(!pip.eof())
{
    c=pip.get();

    cout<<endl;
    cout<<c<<<ios::cur<<" position :"<<k++<<endl;
}

}

```

Pour la généricité : n' hésitez pas à contacter soit moi, soit Abdel, soit Jicé.

Bonne merde A TOUS pour demain