TP-1 POO en C++ : (Initiation à la syntaxe)

- Le but de TP-1 est d'une part un rappel du cours C et d'autre part se familiariser avec la syntaxe C++. Ce BE aborde les points suivants :
- Syntaxe C++ : manipulation des structures complexes, des fonctions, pointeurs, tableaux, chaînes.
- Utilisation basique de la librairie d'entrées/sorties
- Allocation dynamique
- Passage par référence.

Chapitre (1, 2) du Polycopie : Du langage C au C++, Les fonctions.

Exercice 1 : TESTS

Ecrire un programme en C++ qui définit une fonction qui retourne le nombre de jours d'un mois donné (numéro) dans une année donnée : son prototype sera le suivant ***: int nbjour (int mois, int annee).***

- Le mois de février a 28 jours ou 29 les années bissextiles.
- Une année est bissextile si elle est divisible par 4, mais pas si elle est divisible par 100 sauf si elle est divisible par 400.

Exercice 2 : Pointeurs et tableaux à deux dimensions

Ecrire un programme qui lit une matrice A de dimensions N et M au clavier et affiche les données suivantes en utilisant le formalisme pointeur à chaque fois que cela est possible :

a) La matrice A
b) La transposée de A
c) La matrice A interprétée comme tableau unidimensionnel

Exercice 3 : Pointeurs et tableaux à deux dimensions

Ecrire un programme qui lit deux matrices A et B de dimensions N et M respectivement M et P au clavier et qui effectue la multiplication des deux matrices. Le résultat de la multiplication sera affecté à la matrice C, qui sera ensuite affichée. Utiliser le formalisme pointeur à chaque fois que cela est possible.

Exercice 4 : Allocation et libération dynamiques de mémoires

1. Ecrire un programme qui lit 10 phrases d'une longueur maximale de 200 caractères au clavier et qui les mémorise dans un tableau de pointeurs sur char en réservant dynamiquement l'emplacement en mémoire pour les chaînes. Ensuite, l'ordre des phrases est inversé en modifiant les pointeurs et le tableau résultant est affiché.
2. Ecrire un programme qui lit 10 mots au clavier (longueur maximale : 50 caractères) et attribue leurs adresses à un tableau de pointeurs MOT. Effacer les 10 mots un à un, en suivant l'ordre lexicographique et en libérant leur espace en

mémoire. Afficher à chaque fois les mots restants en attendant la confirmation de l'utilisateur (par 'Enter').

Exercice 5 : Fonction et passages de tableaux

1. Ecrire la fonction SOMME_TAB qui calcule la somme des N éléments d'un tableau

TAB du type int. N et TAB sont fournis comme paramètres ; la somme est retournée comme résultat du type long.

2. Ecrire un programme qui lit un tableau A d'une dimension inférieure ou égale à 100 et affiche le tableau et la somme des éléments du tableau.

Exercice 6 : Chaînes

Ecrire un programme qui lit deux chaînes de caractères CH1 et CH2 au clavier et supprime la première occurrence de CH2 dans CH1. Utiliser uniquement des pointeurs, une variable logique TROUVE et la fonction ***strcpy***.

Exemples:

Alphonse	phon	=>	Alse
totalement	t	=>	otalement
abacab	aa	=>	abacab

Exercice 7 : Passage par référence.

Soit le modèle de structure suivant :

```
struct essai {
     int n ;
     double x ;
};
```

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

- Par adresse
- Par référence

Dans les deux cas on écrira un petit programme d'essai de la fonction ; il affichera les valeurs d'une structure de ce type, après appel de ladite fonction.

TP-2 POO en C++ (2ème Séance)

Ce TP aborde les points suivants :
- Définition simple d'une classe, Encapsulation, Allocation dynamique,
- Utilisation basique de la librairie d'entrées/sorties
- Constructeurs et destructeur, Listes d'initialisations, Surcharge de fonctions

Chapitre 3,4 du Polycopie : Les fonctions C++, Classes et Objets

Exercice 1

On souhaite réaliser une classe vecteur 3d permettant de manipuler des vecteurs à trois composantes. On prévoit que sa déclaration se présente ainsi :

```
class Vecteur3d {...double _x,  _y, _z ; ...} ;
```

On souhaite pouvoir déclarer un vecteur soit en fournissant explicitement ses trois composantes, soit en fournissant aucune, auquel cas le vecteur créé possédera trois composantes nulles. Il peut être aussi constant ou non constant ! Définir constructeur, destructeur, fonction d'affichage pour cette classe. Pensez à afficher dans le constructeur et le destructeur (cout<< "Je suis dans le … "<<endl ;)

1) Définir le ou les constructeurs correspondants en utilisant :

1.a Deux fonctions (constructeurs) membre sur-définies,

1.b Une seule fonction membre qui remplace les deux fonctions de 1.a.

1.c Une seule fonction « en ligne » qui remplace 1.b

1.e Une seule fonction avec la liste d'initialisation (corps de la fonction vide ! ! !) qui remplace 1.c.

2) Définir le constructeur de copie pour cette classe.

Exercice 2

Recopier le premier exemple du cours (Pile d'entier) et le préparer. Compléter le programme en ajoutant un fichier principal qui contient le «main». Corriger les erreurs. **Ajouter** une donnée et une fonction permettant de retourner le nombre d'objets piles créées. Exécuter et Valider.

Remarque. Il faut créer un fichier d'en-tête ***(pile.h)***, un fichier de détail ***(pile.cpp)*** et un fichier principal contient le main ***(testpile.cpp)***

Exercice 3

Réaliser une classe «***Point***» permettant de manipuler un point du plan. On prévoira : Un constructeur avec argument les coordonnées en «***Double***» d'un point, valeur par défaut «**0,0**»; Une méthode «***translate()***» effectuant une translation définie par ses deux arguments (double) ; Deux méthodes «***abscisse()*** et ***ordonnée()***» fournissant abscisse et ordonnée d'un objet point, Deux méthodes membres «***rho()*** et ***teta()***» fournissant les coordonnées polaires du point ; Une méthode membre «***rotation()***» qui effectue une rotation dont l'angle est donné en argument. On écrira séparément : (un fichier d'en-tête « ***point.h*** » constituant la déclaration de la classe ; un fichier source «***point.cpp***» constituant la définition des méthodes de cette classe;)

Un programme d'essai main «***mainpoint.cpp***» déclarant un point, affichant ses coordonnées, le déplaçant ou le faisant subir une rotation et l'affichant à nouveau.

TP-3 POO en C++ (3ème Séance)

Ce TP aborde les points suivants :

- Manipulation plus élaborée des classes et des objets
- Membres statiques, Constructeurs de copie, Listes d'initialisations, Surcharge de fonctions, Fonctions Amies

Chapitre 3,4 du Polycopie : Classes et Objets (suite)

Exercice 1

Q1.1 Réaliser une classe «**Complexe**» dans laquelle on définira deux membres privés, la partie réelle et la partie imaginaire. On prévoira des constructeurs et destructeur. On prévoira également les méthodes nécessaires au fonctionnement de cette classe : l'affichage d'un nombre complexe, récupérer la partie réelle ou la partie imaginaire, additionner, soustraire et multiplier deux complexes.

Q1.2 Réaliser une classe polynôme du second degré « **Poly2deg**». Les membres privés (***_a***, ***_b***, ***_c***) sont les coefficients entiers constantes.

On créera les constructeurs nécessaires et les fonctions membres utiles dont la fonction solution qui devra donner la solution de l'équation du second degré, quelle soit réelle où complexe. On utilisera bien sur la classe complexe crée en **Q1.1**.

Ecrire un programme d'application mettant en pratique cette nouvelle classe.

Exercice 2

Q2.1 Réaliser une classe nommée « **Set_ Of_Integer** » permettant de manipuler des ensembles de nombres entiers. On devra pouvoir réaliser sur un tel ensemble les opérations classiques suivantes :

- Lui ajouter un élément, connaître son « cardinal » (nombre d'éléments), savoir si un entier donné lui appartient.
- On conservera les différents éléments de l'ensemble dans un tableau alloué dynamiquement par le constructeur.
- Un argument (dont il existera une valeur par défaut) précisera le nombre maximal d'élément de l'ensemble.

Q2.2 Ecrire un programme utilisant la classe « **Set_Of_Integer** » pour déterminer le nombre d'entiers différents contenus dans un tableau lu en donnée.

Q2.3 Modifier la classe de façon à ce que maintenant l'ensemble d'entiers soit représenté par une **« liste chaînée »**.

Exercice 3

Créer deux classes :

L'une, nommée « **Vecteur3d** » permettant de représenter des vecteurs à 3 composantes de type double ; avec constructeurs, destructeur et une fonction membre d'affichage.

L'autre nommée « **Matrice** » permettant de représenter des matrices carrées de dimension **3x3** ; elle comportera aussi entre autres un constructeur avec argument (adresse d'un tableau **3x3** valeurs) qui initialisera la matrice avec les valeurs correspondantes.

Réaliser deux fonctions amies « **produit ».** Une permettant de fournir le vecteur correspondant au produit d'une matrice par un vecteur. L'autre permettant de fournir la matrice correspondant au produit d'un vecteur par une matrice. Ecrire un programme pour tester les méthodes réalisées.

TP-4 POO en C++ (4 ème Séance)

Ce TP aborde les points suivants :
- Constructeur de copie
- Fonctions Amies
- Surdéfinition d'opérateurs

Chapitre 3, 4, 5 du Polycopie : Classes (fonctions amies), Surcharge d'opérateurs

Exercice 1 : Surdéfinition d'opérateurs de la classe Complexe

On utilisera la classe « Complexe du TP-3 »
Q1.1 Définir les opérateurs « == », « = ! » et l'opérateur « ! »
Q1.2 Définir l'opérateur d'affectation « = »
Q1.3 Définir les opérateur « + », « - », « * »
Q1.4 surcharger les opérateurs << et >> pour que nous puissions afficher et saisir un complexe : (cout<<c1 ; cin>>c2)

Exercice 2 : Surdéfinition d'opérateurs de la classe vecteur3D

On utilisera la classe « Vecteur3d » crée dans le TP-3.
Q2.1 Définir les opérateurs « == et = ! » de manière à ce qu'ils permettent de tester la coïncidence ou le non-coïncidence de deux points (on utiliser des fonctions membres ou fonctions amies).
Q2.2 Définir les opérateurs + pour qu'ils fournissent la somme de deux vecteurs et l'opérateur * pour qu'il fournisse le produit scalaire de deux vecteurs (Il est préférable d'utiliser les fonctions amies).
Q2.3 surcharger les opérateurs << et >> pour que nous puissions afficher et saisir un Vecteur3d : (cout<<v1 ; cin>>v2)

Exercice 3 : Classe String

On désire concevoir une classe chaîne de caractères String de longueur quelconque.

1. Ecrire la déclaration de cette classe dans un fichier ***string1.h***.

Méthodes souhaitées :
- ***longueur()*** : retourne la longueur de la chaîne
- ***nieme()*** : retourne le n ième caractère
- ***affiche()*** : affiche la chaîne
- ***saisie()*** : saisie d'une chaîne d'au plus 1024 caractères
- ***concatene()*** : ajoute une chaîne ou un caractère à la fin de la chaîne
- ***egal()*** : retourne 1 si les chaînes sont égales
- ***minuscule()*** : retourne une nouvelle chaîne formée des éléments de la chaîne d'origine mis en minuscules.

2. Après avoir fait valider cette déclaration, vous devez entreprendre la définition des méthodes dans le fichier ***string1.cpp***.
3. Ensuite utiliser le programme essaiString1.cpp suivant pour valider la classe.

```
#include <iostream.h>
#include "string1.h"

void main() {
    String ch1("essai"), ch2 = ch1, ch3('=' , 80);
    const String ch4("chaîne constante");
    ch1.nieme(1) = 'E'; // le premier caractère de la chaîne
    cout << ch4.nieme(1) << endl;
    ch2.saisie();
    ch2.concatene(" de la classe String");
    ch2.concatene('g');
    if ( ! egal(ch2, "") ) {
     ch2.affiche();
     cout << endl;
    }
    ch2.minuscule().affiche();// est-ce bien raisonnable ???
    cout << endl;
}
```

Attention : des contraintes supplémentaires pourront être déduites du programme d'essai donné.

TP-5 POO en C++ (5ème Séance)

Ce TP aborde les points suivants :

- **Héritage simple**
- **Hiérarchie d'héritage**
- **Héritage multiple**
- Polymorphisme
- Fonction virtuelle pure et classe abstraite

Chapitre 6 du Polycopie : héritage et polymorphisme

Exercice 1 : L'héritage simple

On dispose d'un fichier ***point.h*** contenant la déclaration suivante de la classe **Point**:

```
#include <iostream.h>
class Point {
     public :
          Point(double abs=0.0, double ord=0.0){_x=abs ; _y=ord ;}
          Void affiche(){cout<<"coordonnees"<<_x <<" "<<_y<<"\n" ;}
          Void deplace (double dx, double dy){_x=_x+dx ;_y=_y+dy ;}
     private :
          double _x, _y ;

};
```

Q1.1 Créer une classe **Pointcol**, dérivée de **Point**, comportant :
- Un membre donnée supplémentaire ***cl*** de type ***int*** destiné à contenir la couleur d'un point
- Les fonctions membres suivantes : « ***affiche*** »(redéfinie), qui affiche les coordonnées et la couleur d'un objet de type « **Pointcol »** ; « ***colore*** (***int*** couleur) », qui permet de définir la couleur d'un objet de type « **Pointcol »** ; un **constructeur** définissant la couleur et les coordonnées (non *inline*).

Q1.2 Que fera alors précisément cette instruction : Pointcol P1(2.5, 3.25, 5);

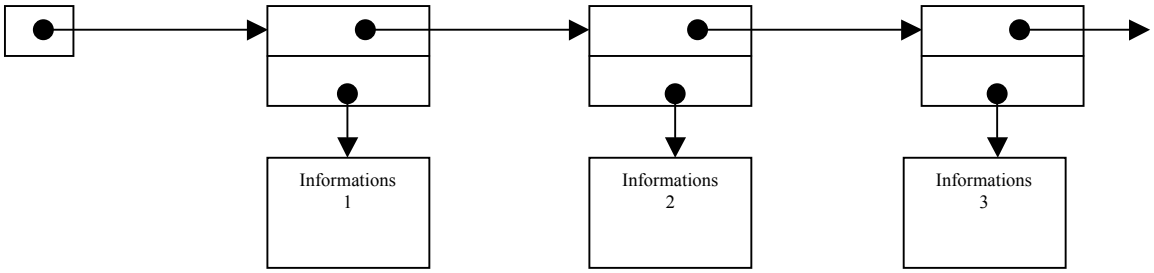
Q1.3 Que faudrait-il faire pour créer une classe **Pointcol** sans héritage mais possédant les mêmes caractéristiques. Quelles différences apparaîtront au niveau des possibilités d'utilisation de ces deux versions.

Exercice 2 : Hiérarchie d'héritage

Q2.1 Recopier l'exemple du cours (Gestion salariés).
Q2.2 Compléter le programme en ajoutant un fichier principal qui contient le «main». Corriger les erreurs.
Q2.3 Ajouter une ou des données et une ou des fonctions permettant de retourner: le nombre de salarié de l'entreprise, nombre de directeurs, employers, commerciaux. Exécuter et Valider.

Exercice 3 : Héritage multiple

On souhaite créer une liste permettant de manipuler des listes chaînées dans lesquelles la nature de l'information associée à chaque nœud de la liste n'est pas connue (par la classe). Une telle liste correspondra au schéma suivant :



La déclaration de la classe liste se présentera ainsi :

```
struct element {
     element *suivant ; void *contenu ;
} ;
class Liste {
     public :   Liste() ;
                ~Liste() ;
                void ajoute(void *) ;
                void *premier() ;
                void *prochain() ;
                int fini() ;
     private :
          element *_debut ;
} ;
```

La fonction « ***ajoute*** » permet d'ajouter un élément pointant sur une information en début de liste dont l'adresse est fournie en argument.
Pour explorer la liste, on a prévu trois fonctions :
- « ***premier*** », qui fournira l'adresse de l'information associée au premier nœud de la liste et qui, en même temps, préparera le processus de parcours de la liste,
- « ***prochain*** », qui fournira l'adresse de l'information associée au « prochain nœud » ; des appels successifs de prochain devront permettre de parcourir la liste (sans qu'il soit nécessaire d'appeler une autre fonction),
- « ***fini*** », qui permettra de savoir si la fin de la liste est atteinte ou non.

Q3.1 Compléter la déclaration précédente de la classe liste et en fournir la définition de manière à ce qu'elle fonctionne comme demandé.

Soit la classe **Point** suivante :

```
class Point {
     public :
        Point (double abs=0.0, double ord=0.0) {_x=abs ; _y=ord ; }
        void affiche() {cout<<"coordonnees" <<_x <<" "<<_y<<"\n" ;}
     private :
        double  _x, _y ;
     };
```

Q3.2 Créer une classe **ListePoints**, dérivée à la fois de Liste et de Point , pour qu'elle puisse permettre de manipuler des listes chaînées de points, c'est-à-dire des listes comparables à celles présentées ci-dessus, et dans lesquelles l'information associée est de type point. On devra pouvoir notamment :

- Ajouter un point en debut d'une telle liste,
- Disposer d'une fonction membre affiche affichant les informations associées à chacun des points de la liste de points

Q3.3 Ecrire un programme pour tester ces classes.

TP-6 POO en C++ (6ème Séance)

Ce TP aborde les points suivants :

- Manipulations sur les flux

Chapitre 7 du Polycopie : Flux

Exercice 1 : Les fichiers

Ecrire un programme qui enregistre (sous forme binaire, et non pas formatée), dans un fichier de nom fourni par l'utilisateur, une suite de nombres entiers fournis sur l'entrée standard.

On conviendra que l'utilisateur fournira la valeur 0 (qui ne sera pas enregistrée dans le fichier) pour préciser qu'il n'a plus d'entier à entrer.

Exercice 2 : Les fichiers

Ecrire un programme qui vous permettra de lister (sur la sortie standard) les entiers contenus dans un fichier tel que celui crée par l'exercice 1.

Exercice 3 : Les fichiers

Ecrire un programme qui vous permettra de retrouver dans le fichier les entiers dont on connaît le « rang ».

On conviendra qu'un rang 0 signifie que l'utilisateur souhaite mettre fin au programme

TP-7 POO en C++ (7ème et 8ème Séance)

Ce TP aborde les points suivants :

- **Surcharge d'opérateur de la classe String !**
- **Héritage simple, Héritage multiple, Polymorphisme**
- **Fonction virtuelle pure et classe abstraite**
- **Manipulations des fonctions des flux (open, read, write, close, seek, tell), avec Héritage multiple et notion classe générique (Template)**
- **Surcharge d'opérateur de la classe String !**

Chapitre 5, 6, 7 du cours

Exercice 1 : Classe String avec Surcharge des opérateurs

On désire modifier la classe String de l'exercice précédent pour y ajouter les opérateurs :

= [] + += << >> == !=

1. Ecrire la nouvelle déclaration de cette classe dans un fichier ***string2.h***.
2. Après avoir fait valider cette déclaration, vous devez entreprendre la définition des méthodes dans un fichier ***string2.cpp***.
3. Ensuite utiliser le programme essaiString2.cpp suivant pour valider la classe

```
#include <iostream.h>
#include "string2.h"
void main() {
    String ch1("essai"), ch2 = ch1, ch3('=', 80);
    const String ch4("chaîne de caractères constante");

    ch1[1] = 'E';    // le premier caractère de la chaîne
    cout << ch4[1] << endl;
    ch1 = "<<<< " + ch2 + " >>>>";
    cout << ch1 << endl;
    cin >> ch2;
    ch2 += " de la classe Strin";
    ch2 += 'g';
    if ( ch2 != "" ) cout << ch2 << endl;
    cout << ch2.minuscule() << endl;
}
```

Exercice 2 : Classe Forme : héritage et polymorphisme

1. Préparation :

On définit une hiérarchie de formes géométriques : le cercle, le triangle, le rectangle et le carré. Pour chaque forme, on veut connaître son périmètre et sa surface. Il possible de déplacer une forme dans le plan, on définira pour cela une classe Coordonnees.

2. **Ecrire les classes :** nécessaires à l'implémentation de cette hiérarchie en vous basant sur l'extrait du « main » ainsi que le résultat de l'exécution fournis ci-dessous.

Extrait du main :

```
void main() {
     Cercle cercle(10,10,4);
      cout <<endl<<cercle<<" surface="<<cercle.surface()<<endl;
      Triangle triangle(20,20,3);
      cout<<endl<<triangle<<" surface="<<triangle.surface()<<endl;
      Rectangle rectangle(30,30,2,5);
      cout << endl <surface() << endl << endl;
      cercle.deplace(50,50);
      cout << "déplacement " << endl;
      cout << cercle << endl << endl;

     *********;   //déclaration du tableau initialisé ci-dessous ???

      tab[0] = &cercle;
     tab[1] = &triangle;
     tab[2] = &rectangle;
     tab[3] = carre;
     float surf=0.0;
     for (int i=0; i<4; i++)  {
          surf += ****************;
     }
     cout << "surface totale : " << surf << endl << endl;

     cout << "périmètre d'une forme tirée au hasard" << endl;
     srand((unsigned int) time(NULL));
     ****** ptr = tab[rand()%4];  // définition de ptr
     cout << **** << " périmètre=" << ******** << endl << endl;
     cout << "destruction de carré alloué dynamiquement" << endl;
     ptr = carre; delete ptr; cout << endl;
}
```

Résultat d'exécution :

```
 - Forme::Forme -- Cercle::Cercle -
Cercle (10,10) r=4 surface=50.2655
- Forme::Forme -- Triangle::Triangle -
Triangle (20,20) c=3 surface=4.5
- Forme::Forme -- Rectangle::Rectangle -
Rectangle (30,30) L=2 l=5 surface=10
- Forme::Forme -- Rectangle::Rectangle -- Carre::Carre -
Carre (100,100) c=2 surface=4

déplacement
Cercle (60,60) r=4

surface totale : 68.7655

périmètre d'une forme tirée au hasard
Carre (100,100) c=2 périmètre=8

destruction de carre alloué dynamiquement
- Carre::~Carre -- Rectangle::~Rectangle -- Forme::~Forme -

- Rectangle::~Rectangle -- Forme::~Forme -
- Triangle::~Triangle -- Forme::~Forme -
- Cercle::~Cercle -- Forme::~Forme -
```

3 .Questions :

1. Où intervient le **polymorphisme** dans le **programme** ci-dessus ?
2. Expliquez l'ordre d'appel des constructeurs et destructeurs.

Exercice 3 : Classe FileOf

1. Objectifs :
   - héritage multiple
   - manipulations de base sur les flux (open, read, write, close, seek, tell)
   - classe générique
2. Préparation :

Ecrire les classes, de la hiérarchie ci dessous, permettant de manipuler des fichiers constitués d'entiers et donnant l'accès direct aux éléments du fichier.

3. Exemple d'utilisation :

```
int main() {
       W_File_of_Int f_out("essai.fic");

       if ( ! f_out ) {
         cerr << "erreur à la création de 'essai.fic'\n";
         return 1;
       }
```

```
for(i=0; i<=10; i++) // Ecriture de 11 entiers dans le fichier
       f_out << i;
cout << f_out.tellp() << "éléments sont écrits dans le fichier.\n";
         // affiche:  11 éléments sont écrits dans le fichier.
f_out.close();
R_File_of_Int f_in("essai.fic");
 int entier;
if ( ! f_in ) {
         cerr << "erreur à la création de essai.fic\n";
         return 1;
}
f_in.seekg(0, ios::end);  // se positionne à la fin du fichier
cout << "Il y a " << f_in.tellg() << " éléments dans le fichier.\n";
          // affiche:  Il y a 11 éléments dans le fichier.
f_in.seekg(0);      // se positionne au début du fichier
while ( 1 ) {        // affichage du contenu du fichier
    f_in >> entier;
    if ( f_in.eof() )
     break;
     cout << entier << " ";
}
f_in.clear(); // ne pas l'oublier ... sortie du while sur erreur eof
cout << endl;
f_in.close();

////////////////////////////////////////////////////////////////
RW_File_of_Int f_io("essai.fic"); //s'il existe, il n'est pas écrasé
if ( ! f_io ) {
    cerr << "erreur à la création de essai.fic\n";
     return 1;
}
f_io.seekp(0, ios::end);   // se positionne à la fin du fichier

cout<<"Il ya déjà"<<f_io.tellp() <<"éléments dans le fichier\n";
            // affiche:  Il y a déjà 11 éléments dans le fichier
for(i=11; i<=19; i++) f_io << i;
f_io.seekp(10);          // se positionne sur le 11 ième entier
while ( 1 ) {            // affichage du contenu du fichier
     f_io >> entier;
     if ( f_io.eof() ) break;
     cout << entier << " ";
}  // affiche:  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
f_io.clear(); // ne pas l'oublier… sortie du while sur erreur eof
f_io.close();
return 0;
}
```

4. Remarques :
   - Une instruction du type f_out.write (...); ne doit pas être compilable.
   - La définition du constructeur de copie n'est pas demandée.
5. Généricité (Template) :
   - Rendre générique ces classes.