Programmation Orientée Objet en C++ (POO en C++)

Travaux Dirigés et Travaux Pratiques

Rahmoune Mohammed

Ce cours est dédié pour les élèves ingénieurs de la première année du cycle ingénieur des Ecoles Nationales des Sciences Appliquées

Table des matières

Série N° 0

Cet exercice est proposé afin de sensibiliser les élèves ingénieurs à utilisé le langage C dans leur réalisation informatique.

Demande client :

Une station d'essence souhaite offrir une application informatique à ses clients qui leurs permettra à mieux gérer leur consommation d'essences.

Pour un client fidèle on suppose :

Dans le premier passage du client, on demande une inscription sur l'application donc nom prénom coordonnées et d'introduire le kilométrage affiché à son compteur et la quantité d'essence souhaité.

Dans les passages qui suivent on doit lui fournir son taux de consommation en lui demandant d'introduire aussi le kilométrage et la quantité d'essence souhaité.

1- D'abord identifier le type d'application et les paramètres essentiels à préparer pour cette application ?
2- Définir le plan d'action
3- Programmer l'application en se basant sur des tableaux
4- Reprogrammer en se basant sur les structures.
5- Améliorer la solution (devoir à réaliser à la maison)

Correction Série N° 0

```
#include <stdio.h>
#include <stdlib.h>
#define N 100


int main()
{
char nom[30],prenom[30];
int i=0;
float km[N],ess[N];
double taux;
```

```
while(1){
system("cls");
if(i==0){
printf("Bonjour,veuiller saisir votre nom et votre prenom\n");
scanf("%s %s",&nom,&prenom);
}else{
printf("Bonjour %s %s c'est votre %d eme visites\n",prenom,nom,i+1);
    }
printf("Veuiller saisir votre kilometrage \n");
scanf("%f",&km[i]);
printf("Veuiller saisir la quantite d'essence\n");
scanf("%f",&ess[i]);
if(i!=0){
taux=ess[i-1]*100/(km[i]-km[i-1]);
printf("Votre taux est: %.2lf\n",taux);
    }
   i++;
system("PAUSE");
    }
   return 0;
}
```

Série N° 1

Exercice I : Namespace

Créer un namespace qui contient une fonction divise (a/b) de type float ou a et b et de type float dans un fichier en tête indépendant et afficher le résultat à l'écran.

Correction

```
#include <cstdlib>
#include <iostream>

using namespace std;

namespace A{ float divise (int a,int b){return (a/b);};}

int main(int argc, char *argv[])
{
   cout<<"ladivision"<<A::divise(1,2)<<endl;
system("PAUSE");
   return EXIT_SUCCESS;
}
```

Exercice II :

1- Réaliser un code en C :

Introduire une structure Rectangle Rec qui lit la largeur et la longueur.
Introduire une fonction extérieure pour calculer l'aire.

2- Réaliser le même code en C++

3- En C : Introduire la fonction dans la structure. Et respecter la définition de la fonction :

float Rec ::aire()

l'accès aux attributs de la structure Rec se fait par un point : Rec.aire() ;

Quel message on obtient ?

4- Réaliser le même programme en C++

5- Conclusion

Correction

Réaliser un code en C :

```
#include <stdio.h>
#include <stdlib.h>

struct Rec {
float x;
float y;
float area(float x,float y);
} rec1;

int main ()
{



printf("Enter x: ");
scanf("%f",&(rec1.));
  printf ("Enter y: ");
scanf("%f",&(rec1.y));
  printf ("l'aire:\n%f\n", rec1.area(rec1.x,rec1.y));

system("PAUSE");
return 0;
}
```

```
float Rec::area(float a,float b)
{
     return (a*b);
     };
```

Réaliser le même programme en C++

```
// pointers to structures
#include <cstdlib>
#include <iostream>
using namespace std;

struct Rec {
float x;
float y;
float area(float x,float y);
} rec1;

int main ()
{


  cout << "Enter x: ";
  cin>>rec1.x;
  cout << "Enter y: ";
  cin >>rec1.y;
  cout << "l'aire:\n"<< rec1.area(rec1.x,rec1.y)<<"\n";

system("PAUSE");
return EXIT_SUCCESS;
}
float Rec::area(float a,float b)
{
return (a*b);
     };
```

Exercice III :

Introduire un entier i qu'on lui associe une référence j, afficher la valeur de i et j, puis modifier la valeur de j et afficher la valeur de i et j.

Correction

```
// reference.cpp : Defines the entry point for the console application.
//

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



int main()
{
        int  i = 18;
        int&j = i;
        cout<< "i est egale "<<i<<endl;
cout << "j est egale "<<j<<endl;
        j=j+2;
        cout << "i est egale "<<i<<endl;
        cout<< "j est egale "<<j<<endl;
        system("PAUSE");
   return 0;


}
```

Exercice IV:

Définir un modèle de structure pouvant contenir une date de type jour, mois, an à partir de trois entiers.
Définissez une variable construite sur ce modèle et réalisez la saisie d'une date dans cette variable.
Définissez un pointeur pouvant contenir une adresse de variable construite sur le modèle de structure de date.
Affectez ce pointeur avec l'adresse de la variable que vous avez définie à l'étape précédente.
Utilisez le pointeur pour écrire le contenu de la variable construite sur le modèle de structure

de date que vous avez affectée.

Correction

```
#include <cstdlib>
#include <iostream>

using namespace std ;
struct date {
int jour;
   int mois;
   int annee;
   };

struct date dat, *ptdate;

main()
{

   /* initialisation du pointeur */
   ptdate = &dat;

   /* saisie de la date */
   cout<<" Entrez une date ( jj mm aaaa ) :";
   cin>> dat.jour>> dat.mois>>dat.annee;

   /* impression de la date */
   cout<< "La date est : \n"<<dat.jour<< dat.mois<< dat.annee<<endl;

   /* impression de la date avec le pointeur */
cout<< "La date est : \n"<<ptdate->jour<<ptdate->mois<<
ptdate->annee<<endl;

system("PAUSE");
return EXIT_SUCCESS;
}
```

Exercice V :

A partir du modèle de structure défini dans l'exercice précédent créez un tableau de cinq structures construites selon ce modèle remplissez ce tableau avec cinq dates.

Définissez un pointeur pouvant contenir une adresse de variable construite sur le modèle de structure de date.
Affectez ce pointeur avec l'adresse de la première variable du tableau.
Utilisez le pointeur pour écrire le contenu des variables du tableau

Correction

```
#include <cstdlib>
#include <iostream>

using namespace std ;
struct date {
int jour;
   int mois;
   int annee;
   };

struct date dat[5], *ptdate;

main()
{

   int i=0;

   /* remplissage du tableau */
   while ( i< 5 ){

      cout<<" Entrez une date ( jj mm aaaa ) :";
cin>> dat[i].jour>> dat[i].mois>> dat[i].annee;
i++;
};

   /* initialisation du pointeur */
   ptdate = &dat[0];
```

```
/* impression du tableau */
for(i=0; i < 5; i++, ptdate++){
/* sans pointeur */
cout<<" Date numero \n"<< i+1 << dat[i].jour<<
dat[i].mois << dat[i].annee<<endl;
/* avec pointeur */
cout<<" Date numero \n"<< i+1 << ptdate->jour<<
ptdate->mois << ptdate->annee <<endl;
   }


system("PAUSE");
return EXIT_SUCCESS;
}
```

Exercice VI

Le cahier de charge d'un magasin est le suivant :

On souhaite réaliser une base de données qui permet de faciliter la gestion du magasin. On stocke d'abord les produits par référence puis chaque référence on lui associe un prix. Le vendeur doit connaître le prix en tapant la référence.

Un produit est représenté par un code produit unique et il est associé à un prix (représenté par une valeur avec point décimal).
La liste des produits est stockée dans un tableau dont la taille est fixée à une valeur quelconque connue que nous appellerons N.
Réalisez une fonction qui correspondant à l'interface suivante :

```
struct ResRech recherch (struct Produit [], int);
```

dans laquelle le premier argument est le tableau de produit et le second argument est un numéro de produit. Cette fonction retourne une structure correspondant au prix et à l'indice dans le tableau de produits.

En cas d'erreur, l'indice dans la structure retournée est égal à moins un.
Ecrivez une fonction principale utilisant cette fonction de recherche

Correction

```
#include <cstdlib>
#include <iostream>


using namespace std;
```

```
struct Produit {
int ref;
   float prix;
} ;

struct ResRech{
float prix;
int indice;
};

extern struct ResRech recherch(struct Produit [], int, int);



int N = 10 ; /* nombre fixe d'elements dans le tableau */

main()
{

  struct Produit tab[] = { {1,87.10},{2,15.20},{3,2.35},{4,70.0},{5,12.40},
          {6,98.0},{7,56.30},{8,35.05},{9,67},{10,40} } ;
int num, i, res ;
  struct ResRech FoundProd;

printf("Les valeurs rentrees dans le tableau sont :\n");
for ( i = 0 ; i < N ; i++ )
printf("%5d   %8f\n",tab[i].ref, tab[i].prix );

printf("\nRentrez le numero de produit recherche : ");
res = scanf("%d",&num) ;
if ( res != 1 ){
printf("Erreur de saisie\n");
return 1 ;
  }
```

```
  FoundProd = recherch( tab, N, num) ;
if(FoundProd.indice !=  -1)
printf("Prix du produit %d : %f a l'indice %d\n",
num,FoundProd.prix,FoundProd.indice);
else
printf("Produit inexistant\n");

system("PAUSE");
return EXIT_SUCCESS;
}
struct ResRech recherch(struct Produit T[], int Size, int Pdt)
{
int rang  ;
struct ResRech resul = { 0.0, -1 };

for( rang =0; rang < Size; rang++){
if(T[rang].ref == Pdt) {
resul.prix = T[rang].prix ;
resul.indice = rang ;
break;
    }   }
   return resul;}
```

Série N° 2

Exercice 1: Classe Point

Dans le plan cartésien, un point est constitué de ses deux coordonnées. Ce TP consiste à la mise en œuvre d'une classe *TPoint* permettant de créer des objets point.

1) **Classe de base**

Ecrivez une classe implantant un point du plan cartésien. Protégez les données dans la partie privée.

Créer un constructeur par défaut initialisant le point à l'origine.

Créer les méthodes pour accéder aux deux coordonnées *GetX GetY SetX* et *SetY*.

Créer une fonction *afficher* et une fonction *saisir* un objet de la classe *TPoint*.

Créer la fonction *MoveTo* permettant de bouger l'objet vers le point passé en paramètres.

2) Donner un nom au point

Ajouter un attribut *nom* à la classe avec ses méthodes d'accès. Les méthodes d'accès doivent permettre de spécifier des chaînes de caractères de longueur variable.

A tout instant, votre classe doit contenir un nom valide.

Ajouter le constructeur à la classe *TPoint* permettant de spécifier les coordonnées et le nom.

3) Constructeurs variés

Outre le constructeur par défaut, les constructeurs suivants vous sont demandés :

- Un constructeur prenant en paramètre les deux coordonnées et le nom.
- Un constructeur recopiant le contenu d'un *TPoint*.

4) Créer plusieurs instances

Créez et manipulez plusieurs instances de points. Créez un tableau de points.

Correction :

TPoint.hpp

```
#ifndef TPOINT_H
#define TPOINT_H
#include <iostream>

class TPoint
{
public :
// Méthodes pour classe de coplien
TPoint();
~TPoint();
TPoint(const TPoint &);
void operator=(const TPoint &);
TPoint (int _x, int _y);

    // Méthodes d'accès
    int getX() const;
int getY() const;
void setX(int );
void setY(int );
friend TPoint operator *(int x, const TPoint &p);
private :
    int x,y;
};#endif
```

TPoint.cpp

```
//---------------------------------------------------------------------------
// TPoint
//---------------------------------------------------------------------------
// Classe point
//---------------------------------------------------------------------------
```

```
#include "TPoint.hpp"

//---------------------------------------------------------------------------
//  TPoint
//---------------------------------------------------------------------------
// Constructeur par défaut
//---------------------------------------------------------------------------
TPoint::TPoint():x(0),y(0)
{}

//---------------------------------------------------------------------------
// ~TPoint
//---------------------------------------------------------------------------
// Destructeur
//---------------------------------------------------------------------------
TPoint::~TPoint()
{}

//---------------------------------------------------------------------------
// TPoint
//---------------------------------------------------------------------------
//  Constructeur de recopie
//---------------------------------------------------------------------------
TPoint::TPoint(const TPoint &p):
x(p.x),y(p.y)
{}

//---------------------------------------------------------------------------
// operator =
//---------------------------------------------------------------------------
// Opérateur d'affectation
//---------------------------------------------------------------------------
void TPoint::operator=(const TPoint &p)
{
```

```
  x=p.x;
  y=p.y;
}

//--------------------------------------------------------------------------
// TPoint
//--------------------------------------------------------------------------
//
//--------------------------------------------------------------------------
TPoint::TPoint(int _x, int _y):
x(_x),y(_y)
{
}

//--------------------------------------------------------------------------
// Méthodes d'accès
//--------------------------------------------------------------------------
void TPoint::setX(int xx)
  {x=xx;}

void TPoint::setY(int yy)
  {y=yy;}

int TPoint::getX() const
  {return x;}

int TPoint::getY() const
{return y;}

//--------------------------------------------------------------------------
// operator *
//--------------------------------------------------------------------------
// Opérateur d'homothétie
//--------------------------------------------------------------------------
```

```
TPoint operator *(int k, const TPoint &p)
{
return TPoint(p.getX()*k,p.getY()*k);
}
```

Exercice 2: Classe Rectangle

Un rectangle peut être défini par deux points du rectangle. Implantez la classe *TRectangle* en utilisant la classe *TPoint*.

Créer trois constructeurs pour le rectangle :

- Constructeur par défaut
- Constructeur prenant en paramètres deux points
- Constructeur prenant en paramètres quatre coordonnées

-Créer les fonctions *afficher* et *saisir* pour un rectangle.

Correction :

TRectangle.hpp

```
#ifndef TRECTANGLE_H
#define TRECTANGLE_H
#include <iostream>
#include "TObjetGraphique.hpp"

using namespace std;
class TRectangle : public TObjetGraphique
{
public:
    // Méthodes pour classe de Coplien
TRectangle();
TRectangle(const TPoint &p,int _long, int _larg);
TRectangle(const TRectangle &);
    ~TRectangle();
    void operator=(const TRectangle &);

    // Méthodes d'accès
```

```
    int getLongueur() const;
    int getLargeur() const;
    void setLongueur(int);
    void setLargeur(int);

    // Méthode points
    TPoint GetPoint1() const;
    TPoint GetPoint2() const;
friend TRectangle operator *(int k, const TRectangle &r);
friend TRectangle operator *(const TRectangle &r, int k);
friend TRectangle operator +(const TPoint &p1, const TPoint &p2);
private:
    int longueur;
    int largeur;
friend ostream &operator<<(ostream &o, TRectangle & R )
{o<<"la largeur est"<<R.getLongueur()<<"la largeur est"<<R.getLargeur()<<endl;
        return o;}
};#endif
```

TRectangle.cpp

```
//---------------------------------------------------------------------------
// TRectangle
//---------------------------------------------------------------------------
// Classe rectangle
//---------------------------------------------------------------------------
#include "TRectangle.hpp"

//---------------------------------------------------------------------------
// TRectangle
//---------------------------------------------------------------------------
// Constructeur par défaut
//---------------------------------------------------------------------------
TRectangle::TRectangle():
TObjetGraphique(),longueur(0),largeur(0)
{}
```

```
//---------------------------------------------------------------------------
// TRectangle
//---------------------------------------------------------------------------
// Constructeur de recopie
//---------------------------------------------------------------------------
TRectangle::TRectangle(const TRectangle &rec):TObjetGraphique(rec),
longueur(rec.longueur),largeur(rec.largeur)
{}

//---------------------------------------------------------------------------
// ~TRectangle
//---------------------------------------------------------------------------
// Destructeur
//---------------------------------------------------------------------------
TRectangle::~TRectangle() { }

//---------------------------------------------------------------------------
// operator=
//---------------------------------------------------------------------------
// Opérateur d'affectation
//---------------------------------------------------------------------------
void TRectangle::operator=(const TRectangle &rec)
{
  TObjetGraphique::operator=(rec);
  largeur=rec.largeur;
  longueur=rec.longueur;
}

//---------------------------------------------------------------------------
// TRectangle
//---------------------------------------------------------------------------
// Constructeur à partir d'un point
//---------------------------------------------------------------------------
```

```
TRectangle::TRectangle(const TPoint &p,int _long, int _larg):
  TObjetGraphique(p),longueur(_long),largeur(_larg)
{
}

//---------------------------------------------------------------------------
// Méthodes d'accès
//---------------------------------------------------------------------------
int TRectangle::getLongueur() const
  {return longueur;}

int TRectangle::getLargeur() const
  {return largeur;}

void TRectangle::setLongueur(int l)
  {longueur = l;}

void TRectangle::setLargeur(int l)
  {largeur = l;}

//---------------------------------------------------------------------------
// GetPoint1
//---------------------------------------------------------------------------
// Renvoie un point origine
//---------------------------------------------------------------------------
TPoint TRectangle::GetPoint1() const
  {return getOrigine();}

//---------------------------------------------------------------------------
// GetPoint2
//---------------------------------------------------------------------------
TPoint TRectangle::GetPoint2() const
{
return TPoint(getOrigine().getX()+longueur,getOrigine().getY()+largeur);
```

```
}

//---------------------------------------------------------------------------
// operator *
//---------------------------------------------------------------------------
// Homotéthie de facteur k
//---------------------------------------------------------------------------
TRectangle operator *(int k, const TRectangle &r)
{
return TRectangle(k*r.getOrigine(),k*r.getLongueur(),k*r.getLargeur());
}

//---------------------------------------------------------------------------
// operator *
//---------------------------------------------------------------------------
// Homotéthie de facteur k
//---------------------------------------------------------------------------
TRectangle operator *(const TRectangle &r, int k)
{
return operator *(k,r);
}

//---------------------------------------------------------------------------
// operator +
//---------------------------------------------------------------------------
//
//---------------------------------------------------------------------------
TRectangle operator +(const TPoint &p1, const TPoint &p2)
{   TRectangle R;
  // R.setLargeur(p2.getX()-p1.getX());
   //R.setLargeur(p2.getY()-p1.getY());
R.longueur=p2.getX();//-p1.getX();
R.largeur=p2.getY();//-p1.getY();
return R;
```

```
  //return TRectangle(p1,p2.getX()-p1.getX(), p2.getY()-p1.getY());
}
```

Main1.cpp

```
#include "TPoint.hpp"
#include "TObjetGraphique.hpp"
#include "TRectangle.hpp"
#include <iostream>
using namespace std;
int main(void)
{
   TRectangle r;

   TRectangle u=r*4;
}
```

Exercice 3: Héritage

1) Classe Point de Coplien

Reprendre la classe TPoint du premier TP. La mettre sous la forme de Coplien (i.e. constructeur sans argument, constructeur de recopie, destructeur et surcharge de l'opérateur d'affectation).

2) Objet Graphique

Construire une classe TObjetGraphique. Tout objet de la classe TObjetGraphique comporte un TPoint nommé origine, un entier représentant le Style de la ligne (attribut StyleLigne) et une chaîne de caractères codant la couleur de l'objet.

Mettre cette classe sous la forme de Coplien et ajouter les méthodes d'accès et de modification des différents attributs.

3) Classe Rectangle

Construire une classe TRectangle qui hérite de la classe TObjetGraphique. Chaque objet de cette nouvelle classe comporte en plus une longueur et une largeur.

Mettre cette classe sous la forme de Coplien et ajouter les méthodes d'accès et de modification des différents attributs. Rajouter une méthode qui calcule les deux sommets opposés du

rectangle et qui renvoie une référence sur un TPoint.

De plus, implémenter les surcharges des opérateurs suivants :

• TPoint = int*TPoint

• Rectangle = Rectangle*int

• Rectangle = int*Rectangle Ces deux opérations représentent l'homothétie

• Rectangle = point1 + point2. Cette opération définit le rectangle de la manière suivante : Le point Origine est le point1. La longueur est égale à (Point2.x – Point1.x) et la largeur est égale à (Point2.y – Point1.y).

Correction :

TObjetGraphique.hpp

```
#ifndef TOBJETGRAPHIQUE_H
#define TOBJETGRAPHIQUE_H

#include <iostream>
#include "TPoint.hpp"

class TObjetGraphique
{
protected:
// Méthodes pour classe de Coplien
TObjetGraphique();
TObjetGraphique(const TObjetGraphique &);
    ~TObjetGraphique();
    void operator=(const TObjetGraphique&);
TObjetGraphique(const TPoint &p);

public:
    // Méthodes d'accès
    char *getCouleurTrait() const;
    void setCouleurTrait(const char *ct);

    int getStyleLigne() const;
    void setStyleLigne(int);
```

```
    TPoint getOrigine() const;
    void setOrigine(const TPoint &);
private:
    TPoint origine;          // Point d'origine
    char *couleurTrait;      // Couleur d'un trait
    int styleLigne;          // style de ligne
};

#endif
```

TObjetGraphique.cpp

```
//---------------------------------------------------------------------------
// Include systèmes
//---------------------------------------------------------------------------
#include <iostream>
using namespace std;

//---------------------------------------------------------------------------
#include "TObjetGraphique.hpp"

//---------------------------------------------------------------------------
// TObjetGraphique
//---------------------------------------------------------------------------
// Constructeur par défaut
//---------------------------------------------------------------------------
TObjetGraphique::TObjetGraphique():
origine(),couleurTrait(0),styleLigne(1)
{
  couleurTrait = new char[strlen("noir")+1];
strcpy(couleurTrait,"noir");
}

//---------------------------------------------------------------------------
// TObjetGraphique
//---------------------------------------------------------------------------
```

```
// Constructeur de recopie
//---------------------------------------------------------------------------
TObjetGraphique::TObjetGraphique(const TObjetGraphique &og):
origine(og.origine) ,couleurTrait(0),styleLigne(og.styleLigne)
{
  couleurTrait = new char[strlen(og.couleurTrait)+1];
strcpy(couleurTrait,og.couleurTrait);
}

//---------------------------------------------------------------------------
// ~TObjetGraphique
//---------------------------------------------------------------------------
// Destructeur
//---------------------------------------------------------------------------
TObjetGraphique::~TObjetGraphique()
{
delete[] couleurTrait;
}

//---------------------------------------------------------------------------
// TObjetGraphique
//---------------------------------------------------------------------------
//
//---------------------------------------------------------------------------
TObjetGraphique::TObjetGraphique(const TPoint &p):
  origine(p),couleurTrait(0),styleLigne(0)
{
  couleurTrait = new char[1];
  *couleurTrait='\0';
}

//---------------------------------------------------------------------------
// operator =
//---------------------------------------------------------------------------
```

```
// Opérateur d'affectation
//---------------------------------------------------------------------------
void TObjetGraphique::operator=(const TObjetGraphique &og)
{
  origine=og.getOrigine();
setCouleurTrait(og.getCouleurTrait());
  styleLigne=og.styleLigne;
}

//---------------------------------------------------------------------------
// Méthodes d'accès
//---------------------------------------------------------------------------
char* TObjetGraphique::getCouleurTrait() const
  {return couleurTrait;}

void TObjetGraphique::setCouleurTrait(const char* ct)
{
  delete [] couleurTrait;
  couleurTrait = new char[strlen(ct)];
strcpy(couleurTrait,ct);
}

int TObjetGraphique::getStyleLigne() const
  {return styleLigne;}

void TObjetGraphique::setStyleLigne(int sl)
  {styleLigne=sl;}

TPoint TObjetGraphique::getOrigine() const
  {return (TPoint(origine));}

void TObjetGraphique::setOrigine(const TPoint& p)
  {origine=p;}
```

Main2.cpp

```
#include <cstdlib>
#include <iostream>
#include "TPoint.hpp"
#include "TObjetGraphique.hpp"
#include "TRectangle.hpp"

using namespace std;

int main(int argc, char *argv[])
{
  TPoint s(2,4);
  TPoint r(s);
  TRectangle R;
  R=s+r;
cout<<R<<endl;
//const TPoint r;
system("PAUSE");
return EXIT_SUCCESS;
}
```

Série N° 3

Exercice 1: Classe vecteur et l'iterateur du vecteur de la librérie standard (STL) du C++

1. Définir une classe Vecteur permettant de conserver les entiers dans un tableau Ajouter un compteur de nombre d'instances créées et proposer les méthodes nécessaires au bon fonctionnement de cette classe.

- Constructeur par défaut : Vecteur(int taille = 0) ;
- Constructeur par copie : Vecteur(const Vecteur& v) ;
- Destructeur : Vecteur() ;
- Opérateur de copie : Vecteur& operator=(const Vecteur& v) ;

2. Ajouter un opérateur pour l'insertion sur le flux en sortie (<<), un opérateur de concaténation (+) et un opérateur d'accès direct ([]).

- flux en sortie : operator <<(ostream &o,const Vecteur &) ;
- concaténation : Vecteur Vecteur : :operator +(const Vecteur &v) ;
- accès direct : proposer une solution pour que l'utilisateur de la classe puisse accéder aux éléments du vecteur de deux manières : en lecture/écriture ou en lecture simple.

3. Définir une classe ItVecteur (classe imbriquée dans Vecteur) définissant un itérateur sur le vecteur. Définir les opérations élémentaires :

- Positionnement au début du vecteur (fourni par le vecteur) : begin()
- Passer à l'élément suivant : ++
- Accéder à l'élément pointé : *
- Comparaison entre deux itérateurs : ==
- Positionnement à la fin du vecteur (fourni par le vecteur) : end()

4. Ecrire la méthode affiche qui imprime le contenu du vecteur (utilisez les itérateurs).

5. Implantez la fonction push_back de manière à obtenir le comportement suivant :

- Vecteur v(10) ; // crée un vecteur de 10 places (0 utilisée)
- v.push_back(5) ;// insère la valeur 5 en tête du vecteur
- v.push_back(3) ;// insère la valeur 3 à la suite (v[0]==5, v[1]==3)

Remarque : Vous devez pour cela modifier l'object Vecteur (attributs et méthodes) en conséquence.

6. modifiez push_back pour qu'elle double la capacité du vecteur lorsque celle-ci est atteinte.

Correction :

Vecteur.h

```
//---------------------------------------------------------------------------
//
// Vecteur.h
//
//---------------------------------------------------------------------------
#ifndef vecteurH
#define vecteurH

#include <ostream>
//---------------------------------------------------------------------------

class Vecteur
{
public:
// Construction / destruction
Vecteur(int capacite =0);
Vecteur(const Vecteur &v);
     ~Vecteur();

// Accesseurs
     int GetTaille() const {return taille;}
int GetCapacite() const {return capacite;}

// Opérateurs
int&operator [](const int &i);
const int &operator [] (const int &i) const;

Vecteur operator +(const Vecteur &v);
```

```
// Méthodes diverses
void push_back(int val);
void affiche();
protected:
    int *tab;   // tableau dynamique d'entiers
    int taille; // taille du tableau
    int capacite; // capacité du tableau

    static int NbInst;  // compteur du nombre d'instances
private:

public:  // deuxième zone public pour séparer la partie Vecteur de ItVecteur
class ItVecteur
{
public:
ItVecteur(Vecteur &v,int n=0);
ItVecteur operator ++();
int operator *();
bool operator ==(const ItVecteur &v);

    // méthodes facultatives
bool operator !=(const ItVecteur &v);
    ItVecteur operator --();
private:
int i;
Vecteur &vect;
};

    ItVecteur begin();
    ItVecteur end();
};

// Opérateur de flux de sortie
std::ostream&operator <<(std::ostream &o,const Vecteur &v);
```

```
#endif
```

Vecteur.cpp

```
//--------------------------------------------------------------------------
//
// Vecteur.cpp
//
//--------------------------------------------------------------------------
#include <iostream>

#include "vecteur.h"
using namespace std;


int Vecteur::NbInst=0;        // Initialise le compteur de sortie (Attribut de classe)
//--------------------------------------------------------------------------

//--------------------------------------------------------------------------
// Vecteur
//--------------------------------------------------------------------------
//  Constructeur (capacité par défaut: 0)
//--------------------------------------------------------------------------
Vecteur::Vecteur(int _capacite):
capacite(_capacite),taille(0)
{
tab = new int[capacite];
   ++NbInst;
}

//--------------------------------------------------------------------------
// Vecteur
//--------------------------------------------------------------------------
//  Constructeur de recopie
//--------------------------------------------------------------------------
Vecteur::Vecteur(const Vecteur &v):
```

```
capacite(v.capacite),taille(v.taille)
{
  // crée le tableau pour la copie
tab = new int[GetCapacite()];
  ++NbInst;

// et copie les éléments
for(int i=0;i<GetTaille();++i)
tab[i]=v[i];
}

//---------------------------------------------------------------------
// ~Vecteur
//---------------------------------------------------------------------
//  Destructeur
//---------------------------------------------------------------------
Vecteur::~Vecteur()
{
  delete [] tab;  // détruit le tableau dynamique
}

//---------------------------------------------------------------------
// operator []
//---------------------------------------------------------------------
// Permet d'accéder à un élément du vecteur (lecture et écriture)
//---------------------------------------------------------------------
int&Vecteur::operator [](const int &i)
{
if(i>=0 && i<GetTaille())
return tab[i];
else
  {
    cerr << "Erreur d'indice" <<endl;   // cerr est le flux d'erreur (comme cout mais pour les
erreurs)
    return tab[0];  // renvoie une référence sur un élément existant
```

```
  }
}

//---------------------------------------------------------------------------
// operator []
//---------------------------------------------------------------------------
// Permet d'accéder à un élément du vecteur (lecture seule)
//---------------------------------------------------------------------------
const int &Vecteur::operator [] (const int &i) const
{
if(i>=0 && i<GetTaille())
return tab[i];
else
  {
    cerr << "Erreur d'indice" <<endl;
    return tab[0];
  }
}

//---------------------------------------------------------------------------
// operator +
//---------------------------------------------------------------------------
// Concatène deux vecteurs
//---------------------------------------------------------------------------
Vecteur Vecteur::operator +(const Vecteur &v)
{
  // Crée un vecteur locale (Ce vecteur sera renvoyé par le return)
  //  Attention: le return ne renvoie pas la variable tmp mais la valeur du vecteur
  Vecteur tmp(GetCapacite()+v.GetCapacite());

  int k=0; // index pour parcourir le vecteur destination

// copie le vecteur *this
for(int i=0;i<GetTaille();++k,++i)
```

```
tmp[k] = tab[i];

// copie le vecteur v
for(int i=0;i<v.GetTaille();++k,++i)
tmp[k] = v.tab[i];

return tmp;
}

//--------------------------------------------------------------------------
// operator<<
//--------------------------------------------------------------------------
// Opérateur de flux de sortie d'un vecteur
//--------------------------------------------------------------------------
ostream&operator <<(ostream &o,const Vecteur &v)
{
for(int i=0;i<v.GetTaille();++i)
    o << v[i] << " ";
return o;
}

//--------------------------------------------------------------------------
// begin
//--------------------------------------------------------------------------
// Renvoie un itérateur sur le premier élément du vecteur
//--------------------------------------------------------------------------
Vecteur::ItVecteur Vecteur::begin()
{
  return ItVecteur(*this);
}

//--------------------------------------------------------------------------
// end
//--------------------------------------------------------------------------
```

```
// Renvoie un itérateur après le dernier élément du vecteur
//---------------------------------------------------------------------------
Vecteur::ItVecteur Vecteur::end()
{
  return ItVecteur(*this,GetTaille());
}

//---------------------------------------------------------------------------
// push_back
//---------------------------------------------------------------------------
// Ajoute un élément à la fin du vecteur
//---------------------------------------------------------------------------
void Vecteur::push_back(int val)
{
  // Si la taille du vecteur dépasse sa capacité
if(GetTaille()>=GetCapacite())
  {
    // alors augmente la capacité du vecteur

    //  crée un vecteur de taille double
capacite = 2*GetCapacite();
int *tmp = new int[GetCapacite()];

// et recopie son contenu
for(int i=0;i<GetTaille();++i)
tmp[i] = tab[i];
delete []tab;
tab = tmp;
  }

tab[GetTaille()]=val;
  ++taille;
}
```

```
//---------------------------------------------------------------------
// affiche
//---------------------------------------------------------------------
// Méthode d'affichage du contenu du vecteur
//---------------------------------------------------------------------
void Vecteur::affiche()
{
  ItVecteur i=begin();
while(!(i==end()))
  {
cout<< *i << '\t';
++i;
  }
}

//**********************************************************************
*
//**********************************************************************
*
//************                                  ***********
//************   ItVecteur                          ***********
//************                                  ***********
//**********************************************************************
*
//**********************************************************************
*

//---------------------------------------------------------------------
// ItVecteur
//---------------------------------------------------------------------
// Constructeur de ItVecteur (imbriquée dans Vecteur)
//  v est le vecteur sur lequel on itère
//  n est la position de l'itérateur  = 0
//---------------------------------------------------------------------
Vecteur::ItVecteur::ItVecteur(Vecteur &v,int n):i(n),vect(v)
 {}
```

```
//---------------------------------------------------------------------------
// operator ++
//---------------------------------------------------------------------------
// Passe au suivant
//---------------------------------------------------------------------------
Vecteur::ItVecteur Vecteur::ItVecteur::operator ++()
{
if(i<vect.GetTaille())   // bloque l'avancement juste après le dernier élément
     ++i;
  return *this;
}

//---------------------------------------------------------------------------
// operator *
//---------------------------------------------------------------------------
// Opérateur de déréférencement
//---------------------------------------------------------------------------
int Vecteur::ItVecteur::operator *()
{
  return vect[i];
}

//---------------------------------------------------------------------------
// operator ==
//---------------------------------------------------------------------------
// Opérateur de comparaison de deux itérateurs
//---------------------------------------------------------------------------
bool Vecteur::ItVecteur::operator ==(const ItVecteur &v)
{
bool eq = true;

// Vérifie que l'on pointe sur le même vecteur
  //  et que les deux index sont les mêmes
```

```
if(&(v.vect)!=&vect || v.i!=i)
eq = false;

  return eq;

// qu'on peut écrire, si on maitrise
//  return&(v.vect)==&vect && v.i==i;
}

//--------------------------------------------------------------------------
// operator !=
//--------------------------------------------------------------------------
//
//--------------------------------------------------------------------------
bool Vecteur::ItVecteur::operator !=(const ItVecteur &v)
  {return !(operator==(v));}

//--------------------------------------------------------------------------
// operator --
//--------------------------------------------------------------------------
//
//--------------------------------------------------------------------------
Vecteur::ItVecteur Vecteur::ItVecteur::operator --()
{
if(i>0)   // bloque l'avancement au premer élément
     --i;
  return *this;
}
```

Principal.cpp

```
//--------------------------------------------------------------------------
// main pour tester le vecteur
//--------------------------------------------------------------------------
#include "vecteur.h"
```

```
#include <iostream>
using namespace std;

//--------------------------------------------------------------------------

int main(int argc, char* argv[])
{
 // Création d'un vecteur
 Vecteur v(6);

v.push_back(0);
 v.push_back(5);
 v.push_back(7);
 v.push_back(6);
v.push_back(2);

 cout << "1er test: affichage d'un vecteur incomplet (taille<capa)" << endl;
v.affiche();     // 1er test: affichage d'un vecteur incomplet (taille<capa)
 cout <<endl;

 cout << "2eme test: dépasse la capacité du vecteur: celui-ci est alors redimensionné" << endl;
v.push_back(1);
 v.push_back(0);

v.affiche();
 cout <<endl;

 cout << "3eme test:  Test de l'opérateur []" << endl;
v[2]=1;
v[3]=2;
v[5]=22;
v[0]=3;

v.affiche();
```

```
 cout <<endl;

 cout << "4eme test:  Test des itérateurs" << endl;
 // Utilisation de l'itérateur
 //   en réalité ce test est déjà réalisé dans la méthode affiche
Vecteur::ItVecteur it = v.begin();
while(!(it==v.end()))
 {
cout<< *it << '\t';
   ++it;
}
 cout <<endl;

 cout << "5eme test: les avancements de l'itérateur devrait être bloqués" << endl;
 cout << "++ ";
 ++it;  // ces avancements de l'itérateur devrait être bloqués
 cout << "++ " <<endl;
 ++it;

 cout << "Affichage:";
 cout << *it; // tente d'outrepasser le dernier élément
 cout << endl << "-- "<<endl << "Affichage:";
 --it;        // un retour en arrière permet de revenir au vrai dernier élément
 cout << *it;

 return 0;
}
//---------------------------------------------------------------------------
```

Série N° 4

Exercice 1: Projet de gestion des objets postaux :

Héritage et polymorphisme

On souhaite écrire une application que pourra utiliser un employé de poste pour calculer le tarif d'affranchissement et trier les objets à expédier. Une classe ObjetPostal caractérisera tous les objets postaux, des spécialisations de cette classe seront définies suivant le type des objets.

1. Définir la classe ObjetPostal. Tous les objets destinés à l'expédition sont munis d'un *nom de destinataire* (string), de l'*adresse du destinataire* (string), du *code postal* (entier), et du *nom de la ville destination* (string). En outre, un booléen indique si l'objet doit être *expédié en recommandé* ou pas.

On interdira la création d'un objet postal dont le nom du destinataire ou son adresse sont incorrectement précisés. Si le nom du destinataire est vide, on lèvera une exception spécifique ExcepNomManquant. Si l'un des composants de l'adresse est vide (ou 0 pour le code postal), on lèvera une exception ExcepAdresse, qui devra porter comme information contextuelle le nom de la personne pour laquelle l'adresse est incomplète.

On prévoira la possibilité de calculer le *prix d'affranchissement* (un réel) des objets postaux. On considérera en outre que la *somme* de deux objets postaux est la somme de leurs prix d'affranchissement.

Enfin, on réalisera la surcharge de l'opérateur << de telle façon qu'il puisse être utilisable pour tout type d'objet postal (attention, ne pas oublier que cet opérateur doit pouvoir prendre en compte n'importe quel flux de sortie).

2. Définir une classe Lettre, spécialisation de ObjetPostal, et qui possède, hormis les données héritées, une donnée de type booléen qui indique si la lettre doit être *expédiée en urgence* ou pas.

Le tarif d'affranchissement d'une lettre se calculera de la façon suivante :

- le prix d'affranchissement normal est 0.5 ¤,

- si la lettre doit être expédiée en recommandé, il y a un surcoût de 2 ¤,

- si la lettre doit être expédiée en urgence, il y a également un surcoût de 3 ¤.

Ne pas oublier de faire en sorte de donner toutes les informations lors de l'affichage à l'écran des caractéristiques d'une lettre.

3. Définir une classe Colis, spécialisation de ObjetPostal, et qui possède, hormis les données héritées, une donnée de type entier qui donne le *poids du colis*, exprimé en grammes.

On interdira la création d'un objet colis dont le poids n'est pas valide (égal à 0). Dans ce cas, on lèvera une exception spécifique ExcepPoids.

Le tarif d'affranchissement d'un colis se calculera de la façon suivante :

- le prix d'affranchissement normal est calculé sur la base de 0.8 ¤ par unité de poids de

100 grammes (par exemple 0.8 ¤ pour 70 grs, 3.2 ¤ pour 318 grs)

- si le colis doit être expédié en recommandé, il y a un surcoût de 4 ¤.

Ne pas oublier de faire en sorte de donner toutes les informations lors de l'affichage à l'écran des caractéristiques d'un colis.

Nous allons maintenant donner à l'employé de poste la possibilité de traiter uniformément un ensemble d'objets postaux.

4. Définir une classe générique Collection qui permet de rassembler, dans un *tableau*, des pointeurs sur objets d'un type quelconque. Les données membres permettront en outre de connaître la taille du tableau (fixée à la création de la collection) et le nombre courant d'éléments dans le tableau.

On munira cette classe d'une méthode d'*insertion* d'un nouvel élément dans le tableau, d'une méthode somme renvoyant en résultat la somme de tous les (objets pointés par les) éléments du tableau, et d'une surcharge de l'opérateur [] pour l'accès aux éléments du tableau.

5. Ecrire un petit programme d'application qui déclare une collection d'objets postaux BoiteALettres pouvant contenir 4 pointeurs sur objets postaux, puis qui réalise les traitements suivants : insertion de 2 lettres et de 2 colis dans cette collection, affichage à l'écran des caractéristiques des objets envoyés en recommandé puis des autres objets, calcul et affichage du prix total d'affranchissement.

Ne pas oublier les interceptions d'exceptions, tout bloc try ne devra être muni que d'un seul bloc catch.

Correction :

ObjetPostal.hpp

```
#ifndef _OBJETPOSTAL_
#define _OBJETPOSTAL_

#include <string>
#include <exception>
#include <iostream>

using namespace std;

class ObjetPostal
```

```
{ protected:
string nomdest, addest;
int code;
  string villedest;

  bool recommande;
public :
//=================================================================
============

class ExcepNomManquant : public exception
  {
    const char* what() const throw()
    {
      return "Il manque le nom du destinataire...";
}
  };



//=================================================================
============

ObjetPostal(string n, string a, int c, string v, bool r=false);
bool get_recom();
virtual float prix() const = 0;   // methode virtuelle pure
friend  ostream& operator<<(ostream &, ObjetPostal &);
float operator+(float p);
};
//=================================================================
============

class ExcepAdresse : public exception
{
private :
    string nom;
```

```
public :
ExcepAdresse(string n)
    {
nom = n;
    }
const char* what() const throw()
{
      string res = "Il manque un element de l'adresse de destination de ";
//+ nom + " ...";
return res.c_str();
}
    ~ExcepAdresse() throw()
    {}
  };
#endif
```

ObjetPostal.cpp

```
#include "ObjetPostal.hpp"


ObjetPostal::ObjetPostal(string n, string a, int c, string v, bool r)
{
if (n == "") throw ExcepNomManquant();
nomdest = n;
if (a == "" || c == 0 || v == "") throw ExcepAdresse(nomdest);
addest = a;
  code = c;
  villedest = v;
recommande = r;
}

bool ObjetPostal::get_recom()
{
  return recommande;
}
```

```
// pour faire la somme des prix, definie seulement ici
// pour tous les objets
float ObjetPostal::operator+(float p)
{
  return prix()+p;
}

// surcharge de << pour tous les objets postaux, definie
// une seule fois pour tous les objets
ostream&operator<<(ostream &o,  ObjetPostal &obj)
{
o<< "Destinataire --> Nom : " << obj.nomdest
<< ", Adresse : " << obj.addest << " " << obj.code
<< " " << obj.villedest <<endl;
  if (obj.recommande) cout << "Objet en recommande \n";
  else cout << "Objet non recommande \n";
  return o;
}
```

Colis.hpp

```
#include "ObjetPostal.hpp"

class Colis : public ObjetPostal
{
  int poids;
public :
Colis(string n, string a, int c, string v, bool r, int p);
float prix() const;
friend ostream & operator<<(ostream &, Colis &) ;
};


class ExcepPoids : public exception
  {
```

```
const char* what() const throw()
{
     return "Poids du colis non precise...";
   }
   };
```

Colis.cpp

```
#include "colis.hpp"

////////////////////////////////////////////////////////

Colis::Colis(string n, string a, int c, string v, bool r, int p) :
ObjetPostal(n, a, c, v, r)
{
if (p == 0) throw ExcepPoids();
poids = p;
}

// calcul du prix des colis
float Colis::prix() const
{
float p = (0.8 * (poids/100));// + ((poids%100)!=0 ? 0.8 : 0);
if (recommande) p+=4;
return p;
}

// fct d'affichage specifique
ostream& operator<<(ostream &o, Colis &C)
{
  o << "Cet objet est une lettre..." <<endl;
  o<<*(ObjetPostal*) &C;
  if (C.recommande) o << "En urgence \n";
  else o << "Pas en urgence \n";
  o << "Prix : " <<C.prix() << " euros\n";
```

```
}
```
Collection
```
#include <iostream>
#include <string>
using namespace std;

//// Collection, classe generique

template<class T>
class Collection
{
private :
  T **elements;
  int taille, taille_courante;
public :
Collection(int t);
  // prevoir aussi const de copie et =
~Collection();
void insere(T *elt);
float somme();
  T *operator[](int i);
 // friend ostream &operator<<(ostream &, const T &);
};
template<class T>
Collection<T>::Collection(int t)
{
  elements = new T*[t];
  taille = t;
  taille_courante = 0;
}

template<class T>
Collection<T>::~Collection()
{
```

```
  delete [] elements;
}

template<class T>
void Collection<T>::insere(T *elt)
{
if (taille_courante < taille)
elements[taille_courante++] = elt;
}

// pour faire la somme de tous les elements
template<class T>
float Collection<T>::somme()
{
  float res=0;
  for (int i=0; i<taille_courante; i++)
    res = *elements[i] + res;
  return res;
}

// surcharge de l'operateur [] pour l'acces aux elements
template<class T>
T *Collection<T>::operator[](int i)
{
return elements[i];
}
```

Lettre.hpp

```
#include "ObjetPostal.hpp"

class Lettre : public ObjetPostal
{
bool urgence;
public :
Lettre(string n, string a, int c, string v, bool r, bool u);
```

```
float prix() const;
friend ostream & operator<<(ostream &, Lettre &) ;
};
```

Lettre.cpp

```
#include "Lettre.hpp"
Lettre::Lettre(string n, string a, int c, string v, bool r, bool u) :
ObjetPostal(n, a, c, v, r)
{
  urgence = u;
}
// calcul du prix des lettres
float Lettre::prix() const
{
  float p = 0.5;
  if (recommande) p+=2;
if (urgence) p+=3;
return p;
}
// fct d'affichage specifique
ostream& operator<<(ostream &o, Lettre &L)
{
  o << "Cet objet est une lettre..." <<endl;
 o<< *(ObjetPostal*) &L;
  if (L.urgence) o << "En urgence \n";
  else o << "Pas en urgence \n";
  o << "Prix : " <<L.prix() << " euros\n";
}
```

Main.cpp

```
#include <cstdlib>
#include <exception>
#include "Lettre.hpp"
#include "colis.hpp"
#include "collection.hpp"
//// Collection, classe générique
```

```
main()
{
try
   {
Collection<ObjetPostal> BoiteaLettres(4);

BoiteaLettres.insere(new Colis("Vincent", "23 Avenue Berthelot", 75005, "Paris",
                              false, 741));
BoiteaLettres.insere(new Lettre("Dupont", "8 Rue Colbert", 69003, "Lyon",
                               false, true));
BoiteaLettres.insere(new Lettre("Durand", "12 Rue Vauban", 13001, "Marseille",
true, false));
     BoiteaLettres.insere(new Colis("Dubois", "3 Bd Victor Hugo", 83000, "Toulon",
                             true, 1387));

cout << "Objets recommandes :\n"<<endl;
cout << "******************  ";
     for (int i=0; i<4; i++)
       {
        if (BoiteaLettres[i]->get_recom())
          {
           cout << "** Objet no " << i << " :\n";
cout<<*BoiteaLettres[i];
          }
       }
     cout << "Objets non recommandes : \n";
cout << "**********************  \n";
     for (int i=0; i<4; i++)
       {
        if (!(BoiteaLettres[i]->get_recom()))
          {
           cout << "** Objet no " << i << " :\n";
cout << *BoiteaLettres[i];
          }
```

```
        }

cout<< "===== Somme totale pour affranchissement : "
<<  BoiteaLettres.somme() << " euros" << endl;
    }
catch(exception &e)
    {
      cerr << "Pb a la construction d'un objet : " <<e.what() << endl;
}

    system("PAUSE");
    return EXIT_SUCCESS;
}
```