Atelier 4: Classes, Objets et Constructeurs

Exercice 1 :

Effectuer les opérations arithmétiques sur des **nombres complexes** à l'aide d'une classe et d'un objet.
Le programme doit demander la partie réelle et imaginaire de deux nombres complexes et afficher les parties réelle et imaginaire de l'opération demandée. (égalité, addition, soustraction, multiplication, division). *Le choix de l'opération peut être fait par un Menu.*

Réponse :

```
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;

class complexe{
 private:
    int reel1,imaginaire1,reel2,imaginaire2;
 public:
    complexe(){}
    complexe(int a1, int b1, int a2, int b2):reel1(a1),imaginaire1(b1),reel2(a2),imaginaire2(b2){}

    void complexe_egalite(int a1, int b1, int a2, int b2){
           if(a1==a2 && b1==b2)
            cout<<"Les deux nombres sont egaux"<<endl;
           else
            cout<<"Les deux nombres ne sont pas egaux"<<endl;
           }

    void complexe_addition(int a1, int b1, int a2, int b2){
        int ad1=a1+a2;
        int ad2=b1+b2;
        cout<<"L'addition de ces deux nombre complexes donne : "<<ad1<<"+("<<ad2<<")i"<<endl;
    }

    void complexe_soustraction(int a1, int b1, int a2, int b2){
        int sou1=a1-a2;
        int sou2=b1-b2;
        cout<<"La soustraction de ces deux nombre complexes donne : "<<sou1<<"+("<<sou2<<")i"<<endl;
    }

    void complexe_multiplication(int a1, int b1, int a2, int b2){
        int mu1=(a1*a2)-(b1*b2);
        int mu2=(a1*b2)+(a2*b1);
        cout<<"La multiplication de ces deux nombre complexes donne : "<<mu1<<"+("<<mu2<<")i"<<endl;
    }

     void complexe_division(int a1, int b1, int a2, int b2){
```

```
        if(a2 ==0 && b2 ==0)
            cout<<"Impossible de divise sur un nombre complexe null"<<endl;
        else if(a2 !=0 && b2==0){
            float div1=a1/a2;
            float div2=b1/a2;
            cout<<"La division de ces deux nombre complexes donne : "<<div1<<"+("<<div2<<")i"<<endl;

        }
        else if(a2 ==0 && b2!=0){
            float div1=a1/b2;
            float div2=b1/b2;
            cout<<"La division de ces deux nombre complexes donne : "<<div2<<"-("<<div1<<")i"<<endl;
        }else{
            int d1=(a1*a2)+(b1*b2);
            int d2=(b1*a2)-(a1*b2);
            int d3=pow(a2,2)+pow(b2,2);
            double div1=(d1/d3);
            double div2=(d2/d3);
            cout<<"La division de ces deux nombre complexes donne : ("<<d1<<"+("<<d2<<"i))/"<<d3<<" =
"<<setprecision(2) <<div1<<"+"<<setprecision(2) <<div2<<"i"<<endl;
        }

    }
};

int main()
{
    int r1,im1,r2,im2,y;
    cout<<"--------- Bienvenue dans les operateurs des nombres complexes ---------\n"<<endl;
    cout<<"Entrer la partie reel du premier nombre complexe : ";
    cin>>r1;
    cout<<"Entrer la partie imaginaire du premier nombre complexe : ";
    cin>>im1;
    cout<<"\nEntrer la partie reel du deuxieme nombre complexe : ";
    cin>>r2;
    cout<<"Entrer la partie imaginaire du deuxieme nombre complexe : ";
    cin>>im2;
    complexe c(r1,im1,r2,im2);
    do{
       int a;
       cout<<"\nVos nombres complexes : "<<r1<<"+("<<im1<<")i   ,  "<<r2<<"+("<<im2<<")i"<<endl;
       cout<<"\n--------- Menu des operateurs ---------\n"<<endl;
       cout<<"Pour l'egalite, entrer le numero : 1 "<<endl;
       cout<<"Pour l'addition, entrer le numero : 2 "<<endl;
       cout<<"Pour la soustraction, entrer le numero : 3 "<<endl;
       cout<<"Pour la multiplication, entrer le numero : 4 "<<endl;
       cout<<"Pour la division, entrer le numero : 5 "<<endl;
       cin>>a;

       switch(a){
          case 1: c.complexe_egalite(r1,im1,r2,im2);
                  break;
          case 2: c.complexe_addition(r1,im1,r2,im2);
                  break;
```

```
        case 3: c.complexe_soustraction(r1,im1,r2,im2);
                break;
        case 4: c.complexe_multiplication(r1,im1,r2,im2);
                break;
        case 5: c.complexe_division(r1,im1,r2,im2);
                break;
        default : cout<<"Erreur nombre entre invalide"<<endl;
      }
      cout<<"Voulez vous continuer ? Oui(1)/Non(0) : ";
      cin>>y;
      }while(y != 0);
   return 0;
}
```

Exercice 2 :

Ecrire un programme en C++ avec une classe mère **Animal**. À l'intérieur, définir des variables nom et d'âge, et la fonction set_value(). Créer ensuite deux sous classes de base **Zebra** et **Dolphin** qui écrivent un message indiquant l'âge, le nom et donnant des informations supplémentaires (par exemple, le lieu d'origine), Créer 2 variables un de type Zebra et l'autre Dolphin puis appeler la méthode set_value() pour chaque instance.

Réponse :

```
#include <iostream>

using namespace std;

class Animal{
 protected:
 string nom;
 int age;
 public :
 void set_value(string a, int b){
     nom=a;
     age=b;
 }
};

class Zebra : public Animal{
  int poid;
  public :
      void set_value(string a, int b, int c){
          nom=a;
          age=b;
          poid=c;
      }
      void Affichage(){
       cout<<"Le nom : "<<nom<<" ,l'age : "<<age<<" , le poid(en kg) : "<<poid<<endl;
      }
};
```

```
class Dolphin : public Animal{
 string couleur;
 public :
      void set_value(string a, int b, string c){
          nom=a;
          age=b;
          couleur=c;
          }
      void Affichage(){
       cout<<"Le nom : "<<nom<<" ,l'age : "<<age<<" , la couleur du peau : "<<couleur<<endl;
      }
};

int main()
{
    Zebra z;
    Dolphin d;
    z.set_value("Roky",15,100);
    d.set_value("Doli",40,"Bleu");
    z.Affichage();
    d.Affichage();


    return 0;
}
```

Exercice 3 :

Créer une classe **Personne** qui comporte trois champs privés, nom, prénom et date de naissance. Cette classe comporte un constructeur pour permettre d'initialiser des données. Elle comporte également une méthode polymorphe *Afficher* pour afficher les données de chaque personne.

- Créer une classe **Employe** qui dérive de la classe Personne, avec en plus un champ Salaire accompagné de sa propriété, un constructeur et la redéfinition de la méthode Afficher.
- Créer une classe **Chef** qui dérive de la classe Employé, avec en plus un champ Service accompagné de sa propriété, un constructeur et la redéfinition de la méthode Afficher.
- Créer une classe **Directeur** qui dérive de la classe Chef, avec en plus un champ Société accompagné de sa propriété, un constructeur et la redéfinition de la méthode Afficher.

Réponse :

```
#include <iostream>

using namespace std;

class Person{
protected:
    string nom;
    string prenom;
    string date;
public:
    Person():nom("Vide"),prenom("Vide"),date("Vide"){}
```

```
    Person(string a, string b, string c):nom(a),prenom(b),date(c){}
    void Afficher(){}
};

class Employe : public Person{
    protected:
    int salaire;
    public:
    Employe():Person("Vide","Vide","Vide"),salaire(0){}
    Employe(string a, string b, string c, int d):Person(a,b,c),salaire(d){}
    void Afficher(){
    cout<<"Le nom de l'employe : "<<nom<<", le prenom de l'employe : "<<prenom<<", la date de
naissance de ce employe : "<<date<<", \nson salaire(DH) : "<<salaire<<endl;
    }
};
class Chef : public Employe{
    protected:
    string service_accompagne;
    public:
    Chef():Employe("Vide","Vide","Vide",0),service_accompagne("Vide"){}
    Chef(string a, string b, string c, int d , string e):Employe(a,b,c,d),service_accompagne(e){}
    void Afficher(){
    cout<<"\nLe nom de chef : "<<nom<<", le prenom de chef : "<<prenom<<", la date de naissance de
chef : "<<date<<", son salaire(DH) : \n"<<salaire<<", le service qui accompagne :
"<<service_accompagne<<endl;
    }
};
class Directeur : public Chef{
    protected:
   string societe_accompagne;
    public:
    Directeur():Chef("Vide","Vide","Vide",0,"Vide"),societe_accompagne("Vide"){}
    Directeur(string a, string b, string c, int d , string e, string
f):Chef(a,b,c,d,e),societe_accompagne(f){}
    void Afficher(){
    cout<<"\nLe nom du directeur : "<<nom<<", le prenom du directeur : "<<prenom<<", la date de
naissance du directeur : "<<date<<",\n son salaire(DH) : "<<salaire<<", le service qui accompagne :
"<<service_accompagne<<", societe qui accompagne : "<<societe_accompagne<<endl;
    }
};

int main()
{
    Employe e("El bakkali","Said","2 janvier 1985",5000);
    Chef c("Ben taib","Omar","7 juillet 1972",9000,"maintenance");
    Directeur d("El aarabi","Ismail","26 avril 1965",14000,"Gestion de l'entreprise","IBM");
    e.Afficher();
    c.Afficher();
    d.Afficher();

    return 0;
}
```

Exercice 4 :

Réaliser une classe C++ "**vecteur3d**" permettant de manipuler des vecteurs à 3 composantes (de type float). On y prévoira :

- un constructeur, avec des valeurs par défaut (0),
- une fonction d'affichage des 3 composantes du vecteur, sous la forme : (x, y, z)
- une fonction permettant d'obtenir la **somme** de 2 vecteurs ;
- une fonction permettant d'obtenir le **produit** scalaire de 2 vecteurs.
- une fonction **coincide** permettant de savoir si 2 vecteurs ont mêmes composantes.
- une fonction qui renvoie la **norme** du vecteur
- une fonction nommée normax permettant d'obtenir, parmi deux vecteurs, celui qui a la plus grande norme.

On prévoira trois situations :

- le résultat est renvoyé par valeur ;
- le résultat est renvoyé par adresse, l'argument étant également transmis par adresse.
- le résultat est renvoyé par référence, l'argument étant également transmis par référence.

Réponse :

```
#include <iostream>
#include <vector>
#include <iterator>
#include <math.h>
#include <iomanip>

using namespace std;

class vecteur3d{
    vector <float> vect;
    vector <float>::iterator itr;
public:
    vecteur3d(){}
    vecteur3d(float x, float y, float z){
        vect={x,y,z};
    }
    void affichervect(){
        for(itr=vect.begin();itr!=vect.end();++itr){
            cout<<*itr<<", ";
        }
       cout<<endl;
    }
    vecteur3d somme(const vecteur3d & autre){
        return vecteur3d(vect[0]+autre.vect[0],vect[1]+autre.vect[1],vect[2]+autre.vect[2]);
    }
    vecteur3d produit(const vecteur3d & autre){
        return vecteur3d(vect[0]*autre.vect[0],vect[1]*autre.vect[1],vect[2]*autre.vect[2]);
    }
    void coincide(vecteur3d v1 , vecteur3d v2){
       if(v1.vect[0]== v2.vect[0] && v1.vect[1]== v2.vect[1] && v1.vect[2]== v2.vect[2])
        cout<<"Les deux vecteurs ont memes composantes."<<endl;
       else
```

```
        cout<<"Les deux vecteurs ont des composantes differentes."<<endl;
    }
    float norme(vecteur3d v){
        float calc;
        calc=sqrt(pow(v.vect[0],2)+pow(v.vect[1],2)+pow(v.vect[2],2));
        return calc;
    }
    void normaux(vecteur3d v1 , vecteur3d v2){
       float v1_norm= norme(v1);
       float v2_norm= norme(v2);
       if(v1_norm > v2_norm)
        cout<<"Le premier vecteur ve1 a le plus grand norme qui est egale a : "<<v1_norm<<endl;
       else if (v1_norm < v2_norm)
        cout<<"Le deuxieme vecteur ve2 a le plus grand norme qui est egale a : "<<v2_norm<<endl;
       else
        cout<<"Les deux vecteur ont le meme norme qui est egale a : "<<v1_norm<<endl;
    }
};

int main()
{
    vecteur3d ve1(1.5,2.5,3.5);
    vecteur3d ve2(4.5,5.5,6.5);
    ve1.affichervect();
    ve2.affichervect();
    vecteur3d somm =ve1.somme(ve2);
    cout<<"La somme est :";
    somm.affichervect();
    vecteur3d prod =ve1.produit(ve2);
    cout<<"Le produit est :";
    prod.affichervect();
    vecteur3d coi;
    coi.coincide(ve1,ve2);
    vecteur3d n;
    cout<<"La norme du vecteur ve1 est : "<<setprecision(3)<<n.norme(ve1)<<endl;
    vecteur3d no;
    no.normaux(ve1,ve2);

    return 0;
}
```

Exercice 5 :

Ecrire un programme en C++ qui vérifie combien de fois une fonction « call » d'une classe Test a été appelée à partir du programme principal, main.
Note : penser à utiliser une variable static.

Réponse :

```
#include <iostream>

using namespace std;
```

```cpp
class T{
public:
    static int compt;
    void call();
    void afficher(){
        cout<<"La fonction call est appelee "<<compt<<" fois."<<endl;
    }
};
int T::compt=0;
void T::call(){compt++;}

int main()
{
    T t;
    t.call();
    t.call();
    t.call();
    t.afficher();

    return 0;
}
```

Exercice 6 :

Réaliser une classe point permettant de manipuler un point d'un plan. On prévoira :

- un constructeur recevant en arguments les coordonnées (float) d'un point ;
- une fonction membre ***deplace*** effectuant une translation définie par ses deux arguments (float);
- une fonction membre ***affiche*** se contentant d'afficher les coordonnées cartésiennes du point.

Les coordonnées du point seront des membres donnés privés.
On écrira séparément :

- un fichier source constituant la déclaration de la classe ;
- un fichier source correspondant à sa définition.

Dans le programme principale (main), testez la classe en : déclarant un point, l'affichant, le déplaçant et l'affichant à nouveau.

Réponse :

Dans point.h :

```cpp
#ifndef POINT_H_INCLUDED
#define POINT_H_INCLUDED
class point{
    float x;
    float y;
public:
    point(){}
    point(float abs , float cor):x(abs),y(cor){}
    void afficher();
    void deplace(float depx, float depy);
};
#endif // POINT_H_INCLUDED
```

Dans point.cpp :

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

using namespace std;

void point::afficher(){
cout<<"Les coordonnees du point sont : ("<<x<<","<<y<<") ."<<endl;
}

void point::deplace(float depx, float depy){
 x+=depx;
 y+=depy;
 cout<<"le point a est deplace , les nouveaux coordonees sont : ("<<x<<","<<y<<") ."<<endl;
}
```

Dans main.cpp :

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

using namespace std;
int main()
{
    point p(1.5,2.5);
    p.afficher();
    p.deplace(1.1,1.2);

    return 0;
}
```

Exercice 7 :

Une pile est un ensemble dynamique d'éléments où le retrait se fait d'une façon particulière. En effet, lorsque l'on désire enlever un élément de l'ensemble, ce sera toujours le dernier inséré qui sera retiré. Un objet ***pile*** doit répondre aux fonctions suivantes :

- Initialiser une pile *(constructeur(s))*
- Empiler un élément sur la pile *(push)*
- Dépiler un élément de la pile *(pop)*

Pour simplifier, nous allons supposer que les éléments à empiler sont de type **int**.
Le programme principale main comprend la définition d'une classe ***pile*** et un programme de test qui crée deux piles **p1** et **p2**, empile dessus des valeurs entières et les dépiler pour vérifier les opérations **push** et **pop**.

Réponse :

```
#include <iostream>
#include <list>

using namespace std;

class Pile{
    list <int> l1;
    list <int> ::iterator itr;
public:
    Pile(){}
```

```
    Pile(int el1 , int el2 , int el3){
        l1.push_front(el1);
        l1.push_front(el2);
        l1.push_front(el3);
    }
    Pile(int el1 , int el2 , int el3 , int el4){
        l1.push_front(el1);
        l1.push_front(el2);
        l1.push_front(el3);
        l1.push_front(el4);
    }
    void empliler( int el){
        l1.push_front(el);
    }
    void depliler(){
        l1.pop_front();
    }
    void afficher(){
        for(itr = l1.begin() ; itr!= l1.end() ;++itr){
            cout<<*itr<<"  ";
        }
        cout<<endl;
    }
};

int main()
{
    Pile p1(2,5,4,8),p2(7,3,6);
    cout<<"La pile p1 est :"<<endl;
    p1.afficher();
    cout<<"La pile p2 est :"<<endl;
    p2.afficher();
    p1.empliler(3);
    cout<<"Apres empiler 3 dans p1, la pile devient : "<<endl;
    p1.afficher();
    p2.depliler();
    cout<<"Apres depiler un element de p2, la pile devient : "<<endl;
    p2.afficher();
    return 0;
}
```

Exercice 8 :

Imaginons une application qui traite des fichiers. Ces fichiers vont être lus en mémoire, traités puis sauvegardés. Une fois lu en mémoire, un fichier a deux caractéristiques, une adresse à partir de laquelle se situe le fichier et une longueur, ce qui se concrétisera par un pointeur et une longueur en nombre d'octets.
Imaginons la classe "***Fichier***" avec un constructeur et un destructeur et les trois méthodes suivantes:

- la méthode "***Creation***" qui va allouer un certain espace à partir du pointeur P,
- la méthode "***Remplit***" qui va remplir arbitrairement cet espace (ces remplissages arbitraires sont la preuve de la bonne gestion mémoire car l'accès à une zone non déclarée provoque une violation d'accès),
- la méthode "***Affiche***" qui va afficher la zone mémoire pointée par P.

Puis écrivons un programme *main* qui instancie notre classe par **new**, appelle nos trois méthodes et détruit l'objet par **delete** et par un **destructeur**.

Réponse :

```
#include <iostream>

using namespace std;

class fichier{
private:
    char *P;
    int longueur;
public:
    fichier(){}
    fichier(int b):longueur(b){}
    void creation (){
    P= new char [longueur];
    }
    void remplit (){
        cout<<"Remplit le fichier en ne depacant pas la taille "<<longueur<<endl;
    for(int i=0 ; i<longueur ; i++){
        cin>>P[i];
    }
    cout<<endl;
    }
    void afficher(){
      cout<<"Le fichier est :"<<endl;
      for(int i=0 ; i<longueur ; i++){
        cout<<P[i];
      }
      cout<<endl;
    }
    ~fichier(){
       delete []P;
    }
};

int main()
{
    fichier f(10);
    f.creation();
    f.remplit();
    f.afficher();

    return 0;
}
```

Exercice 9 :

Créez une classe **liste simplement chaînée**, avec une classe **liste**. Cette classe a un pointeur sur le premier élément de la liste. Elle a une méthode pour **ajouter** ou **supprimer** un élément *au début* de la liste et une pour **afficher** la liste en entier.
Evitez toute fuite mémoire. Les éléments de la liste seront contenu dans la structure *element*.

Réponse :

```cpp
#include <iostream>
#include <list>

using namespace std;

struct Element {
    int valeur;
    Element* suivant;

    Element(int val) : valeur(val), suivant(nullptr) {}
};

class liste{
   Element* premier;
   public:
       liste() : premier(nullptr) {}

       ~liste() {
        Element* courant = premier;
        while (courant != nullptr) {
            Element* suivant = courant->suivant;
            delete courant;
            courant = suivant;
        }
    }

       void ajouter(int a){
        Element* nouvel_element = new Element(a);
        nouvel_element->suivant = premier;
        premier = nouvel_element;
       }
       void supprimer(){
           if (premier == nullptr) {
            return;
        }

        Element* temp = premier;
        premier = premier->suivant;
        delete temp;
       }
       void afficher(){
           Element* courant = premier;
        if (courant == nullptr) {
            cout << "Liste est vide" << endl;
            return;
        }

        cout << "Liste est :  ";
        while (courant != nullptr) {
            cout << courant->valeur;
            if (courant->suivant != nullptr) {
                cout << " -> ";
            }
            courant = courant->suivant;
```

```
        }
        cout <<endl;
    }

};

int main()
{
    liste l;
    l.ajouter(4);
    l.ajouter(5);
    l.ajouter(6);
    l.ajouter(7);
    l.afficher();
    l.supprimer();
    l.afficher();

    return 0;
}
```

Exercice 10 :

Soit une chaîne de caractères contenant une **date** (JJ/MM/AAAA) et une **heure** (HH:NN) sous la forme JJMMAAAAHHNN.

Par exemple 010920091123 représente la date du 1er septembre 2009 à 11h23.

Créer un programme permettant d'extraire les différents champs et de les afficher.

Réponse :

```
#include <iostream>

using namespace std;

class date{
    string jour;
    string mois;
    string annee;
    string heure;
    string minute;

   public:
       date():jour("00"),mois("00"),annee("0000"),heure("00"),minute("00"){}
       date(string dateheure){
           jour=dateheure.substr(0,2);
           mois=dateheure.substr(2,2);
           annee=dateheure.substr(4,4);
           heure=dateheure.substr(8,2);
           minute=dateheure.substr(10,2);
       }
       string nommois(){
            string
m[]={"Janvier","Fevrier","Mars","Avril","Mai","Juin","Juillet","Aout","Septembre","Octobre","Nouvembr
e","Decembre"};
```

```
            return m[stoi(mois)-1];
        }
        void afficher(){
            cout<<"C'est le "<<jour<<" "<<nommois()<<" "<<annee<<" a "<<heure<<"h"<<minute<<endl;
        }
};

int main()
{
    string c;
    cout<<"Donner une date/heure de cette forme JJMMAAAAHHNN (Jour/Mois/Annee/Heure/Minute) :
"<<endl;
    cin>>c;
    date d(c);
    d.afficher();

    return 0;
}
```

Exercice 11 :

Écrire une classe **Traitement** en C++ qui implémente les méthodes suivantes :

1. La méthode « **Initialise** » qui demande à l'utilisateur de saisir 15 entiers, un par un, puis stoke ces entiers dans un vecteur (**attribut de la classe**), le programme vérifie les valeurs saisies par l'utilisateur (**cas de chaine de caractère par exemple**), la méthode doit uniquement accepter des entiers **pairs** et n'accepte pas **les valeurs nulles « 0 »**.
2. La méthode **show** qui affiche les éléments de vecteur **en utilisant la récursivité**.
3. Les deux méthodes **amies** qui renvoient un **double**, la première « **moyenne** » pour calculer la moyenne du vecteur et la deuxième « **médian** » pour calculer la médian de vecteur.

Réponse :

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

class Traitement{
   public:
       vector <int> v;
       Traitement(){}
       void initialise(){
            int a;
             cout<<"Donner 15 entiers pairs :"<<endl;
             for(int i=0 ; i<15 ; i++){
                cout<<"Le nombre "<<i+1<<" : ";
                cin>>a;
                if(a == 0 || a%2!=0){
                do{
                    cout<<"Invalid ! entrer le nombre "<<i+1<<" de nouveau :";
```

```
                    cin>>a;
                }while(a!=0 || a%2==0);
                }
                v.push_back(a);
             }
             cout<<endl;
       }
       void show(vector <int> v ,int i){
           if(i >= 15)
            return;
           cout<<v[i]<<"  ";
           show(v,i+1);
       }
       friend double moyenne(vector <int> v);
       friend double mediane(vector <int> v);
};
double mediane(vector <int> v){
            sort(v.begin(),v.end());
            return (v[15/2 -1]+v[15/2] /2);

       }

double moyenne(vector <int> v){
           int som=0;
           for(int i=0 ; i<15 ; i++){
            som+=v[i];
           }
           double moy = som/15;
           return moy;
       }

int main()
{
    Traitement t1,t2;
    t1.initialise();
    t2.show(t1.v,0);
    cout<<"\nLa moyenne de ce vecteur est : "<<moyenne(t1.v)<<endl;
    cout<<"\nLa mediane de ce vecteur est : "<<mediane(t1.v)<<endl;

    return 0;
}
```