## A t e l i e r 4 : C l a s s e s , O b j e t s , C o n s t r u c t e u r s

**Essafia Hajaj Elaaroud**

Pr :ikram BENABDELOUAHAB

## Exercice1:

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

## solution:

#include <iostream>

using namespace std;

class Complex {

private:

double rel;

double img;

public:

Complex() : rel(0), img(0) {}

Complex(double r, double i) : rel(r), img(i) {}

void saisir() {

cout << "donnez la partie réelle : ";

cin >> rel;

cout << "donnez la partie imaginaire : ";

cin >> img;

}

void afficher() const {

cout << rel;

if (img >= 0) {

cout << " + " << img << "i" << endl;

} else {

cout << " - " << -img<< "i" << endl;

}

}

Complex operator+(const Complex& other) const {

return Complex(rel + other.rel, img + other.img);

}

Complex operator-(const Complex& other) const {

return Complex(rel - other.rel, img - other.img);

}

Complex operator\*(const Complex& other) const {

return Complex(rel \* other.rel - img \* other.img, rel \* other.img + img \* other.rel);

}

Complex operator/(const Complex& other) const {

double denominateur = other.rel \* other.rel + other.img \* other.img;

if (denominateur == 0) {

cout << "Erreur : Division par zéro !" << endl;

return Complex();

}

return Complex((rel \* other.rel + img \* other.img) / denominateur,

(img \* other.rel - rel \* other.img) / denominateur);

}

bool operator==(const Complex& other) const {

return (rel == other.rel && img == other.img);

}

};

int main() {

Complex c1, c2, resultat;

int choix;

cout << "entrez le premier nombre complexe :" << endl;

c1.saisir();

cout << "entrez le deuxième nombre complexe :" << endl;

c2.saisir();

cout << "\n Choisissez une opération :\n";

cout << "1. Addition\n";

cout << "2. Soustraction\n";

cout << "3. Multiplication\n";

cout << "4. Division\n";

cout << "5. Tester l'égalité\n";

cout << "Votre choix : ";

cin >> choix;

switch (choix) {

case 1:

resultat = c1 + c2;

cout << " l'addition : ";

resultat.afficher();

break;

case 2:

resultat = c1 - c2;

cout << " la soustraction : ";

resultat.afficher();

break;

case 3:

resultat = c1 \* c2;

cout << " la multiplication : ";

resultat.afficher();

break;

case 4:

resultat = c1 / c2;

cout << " la division : ";

resultat.afficher();

break;

case 5:

if (c1 == c2) {

cout << "Les deux nombres complexes sont égaux." << endl;

} else {

cout << "Les deux nombres complexes ne sont pas égaux." << endl;

}

break;

default:

cout << "Choix invalide." << endl;

break;

}

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.

## solution:

#include <iostream>

#include <string>

using namespace std;

class Animal {

protected:

string nom;

int age;

public:

void set\_value(string n, int a) {

nom = n;

age = a;

}

virtual void display\_info() {

cout << "Nom: " << nom << ", Âge: " << age << endl;

}

};

class Zebra : public Animal {

public:

void display\_info() override {

cout << "Nom: " << nom << ", Âge: " << age << " ans." << endl;

cout << "Information supplémentaire : Le zèbre est originaire en Afrique central et australe." << endl;

}

};

class Dolphin : public Animal {

public:

void display\_info() override {

cout << "Nom: " << nom << ", Âge: " << age << " ans." << endl;

cout << "Information supplémentaire : Le dauphin vivent dans les océans du monde entier." << endl;

}

};

int main() {

Zebra zebra;

Dolphin dolphin;

zebra.set\_value("Marty", 8);

dolphin.set\_value("Flipper", 6);

zebra.display\_info();

dolphin.display\_info();

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.

## solution:

#include <iostream>

#include <string>

using namespace std;

class Personne {

protected:

string nom;

string prenom;

string date\_naissance;

public:

Personne(string nom, string prenom, string date\_naissance): nom(nom), prenom(prenom), date\_naissance(date\_naissance) {}

virtual void Afficher() const {

cout << "Nom: " << nom << ", Prenom: " << prenom << ", Date de naissance: " << date\_naissance << endl;

}

};

class Employe : public Personne {

protected:

double salaire;

public:

Employe(string nom, string prenom, string date\_naissance, double salaire)

: Personne(nom, prenom, date\_naissance) , salaire(salaire) {}

void Afficher() const override {

Personne::Afficher();

cout << "Salaire: " << salaire << " euros" << endl;

}

};

class Chef : public Employe {

protected:

string service;

public:

Chef(string nom, string prenom, string date\_naissance, double salaire, string service)

: Employe(nom, prenom, date\_naissance, salaire), service(service) {}

void Afficher() const override {

Employe::Afficher();

cout << "Service: " << service << endl;

}

};

class Directeur : public Chef {

protected:

string societe;

public:

Directeur(string nom, string prenom, string date\_naissance, double salaire, string service, string societe)

: Chef(nom, prenom, date\_naissance, salaire, service), societe(societe) {}

void Afficher() const override {

Chef::Afficher();

cout << "Société: " << societe << endl;

}

};

int main() {

Personne p("hajaj", "safia", "04/01/2004");

p.Afficher();

Employe e("chairi", "hamza", "10/03/2000", 3500.0);

e.Afficher();

Chef c("hichou", "Laila", "07/11/1995", 4000.0, "Mathématique");

c.Afficher();

Directeur d("hassoun", "Mariam", "09/12/1989", 6000.0, "Ressources financières", "Capitaliste");

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.

## solution:

#include <iostream>

#include <cmath>

using namespace std;

class Vecteur3D {

private:

float x, y, z;

public:

Vecteur3D(float x = 0, float y = 0, float z = 0) : x(x),y(y),z(z){}

void afficher() const {

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

}

Vecteur3D somme(const Vecteur3D & other) const {

return Vecteur3D(x + other.x, y + other.y, z + other.z);

}

float produitScalaire (const Vecteur3D & other) const {

return x \* other.x + y \* other.y + z \* other.z;

}

bool coincide(const Vecteur3D& other) const {

return (x == other.x && y == other.y && z == other.z);

}

float norme() const {

return std::sqrt(x \* x + y \* y + z \* z);

}

Vecteur3D normaxParValeur (const Vecteur3D & other) const {

return (this->norme() > other.norme()) ? \*this : other;

}

Vecteur3D\* normaxParAdresse (const Vecteur3D\* other) const {

return (this->norme() > other->norme()) ? const\_cast<Vecteur3D\*>(this) : const\_cast<Vecteur3D\*>(other);

}

Vecteur3D & normaxParReference(Vecteur3D & other) const {

return (this->norme() > other.norme()) ? \*const\_cast<Vecteur3D\*>(this) : other;

}

};

int main() {

Vecteur3D v1(3.0f, 4.0f, 5.0f);

Vecteur3D v2(6.0f, 7.0f, 8.0f);

cout << "Vecteur v1: ";

v1.afficher();

cout << "Vecteur v2: ";

v2.afficher();

Vecteur3D somme = v1.somme(v2);

cout << "Somme de v1 et v2: ";

somme.afficher();

float produit = v1.produitScalaire(v2);

cout << "Produit scalaire de v1 et v2: " << produit <<endl;

bool coincide = v1.coincide(v2);

cout << "v1 et v2 coïncident: " << (coincide ? "Oui" : "Non") << endl;

cout << "Norme de v1: " << v1.norme() <<endl;

Vecteur3D maxValeur = v1.normaxParValeur(v2);

cout << "le Vecteur avec la plus grande norme (par valeur): ";

maxValeur.afficher();

Vecteur3D\* maxAdresse = v1.normaxParAdresse(&v2);

cout << "Le Vecteur avec la plus grande norme (par adresse): ";

maxAdresse->afficher();

Vecteur3D& maxReference = v1.normaxParReference(v2);

cout << "le Vecteur avec la plus grande norme (par référence): ";

maxReference.afficher();

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

## solution:

#include <iostream>

using namespace std;

class Test {

public:

void call() {

count++;

}

static int getCount() {

return count;

}

private:

static int count;

};

int Test::count = 0;

int main() {

Test obj1, obj2;

obj1.call();

obj2.call();

obj1.call();

obj2.call();

cout << "La fonction call a été appelée " << Test::getCount() << " fois." <<endl;

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.

## Solution:

**Déclaration de la classe**

#ifndef POINT\_M

#define POINT\_M

class Point {

private:

float x;

float y;

public:

Point(float x = 0.0, float y = 0.0);

void deplace(float dx, float dy);

void affiche() const;

};

#endif

**Définition de la classe**

#include "Point.M"

#include <iostream>

using namespace std;

Point::Point(float x, float y) : x(x), y(y) {}

void Point::deplace(float dx, float dy) {

x += dx;

y += dy;

}

void Point::affiche() const {

cout << "Point(" << x << ", " << y << ")" << endl;

}

int main() {

Point p(1,5, 3.5);

cout << "Donnez les Coordonnées initiales : ";

p.affiche();

p.deplace(2.0, 4.5);

cout << "les Coordonnées après déplacement : ";

p.affiche

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.

## solution:

#include <iostream>

#include <vector>

#include <stdexcept>

class Pile {

private:

std::vector<int> elements;

public:

Pile() {}

bool estVide() const {

return elements.empty();

}

void empiler(int element) {

elements.push\_back(element);

}

int depiler() {

if (estVide()) {

throw std::out\_of\_range("La pile est vide");

}

int element = elements.back();

elements.pop\_back();

return element;

}

int sommet() const {

if (estVide()) {

throw std::out\_of\_range("La pile est vide");

}

return elements.back();

}

};

int main() {

Pile p1;

Pile p2;

p1.empiler(10);

p1.empiler(15);

p1.empiler(30);

p2.empiler(50);

p2.empiler(60);

std::cout << "Dépiler de p1 :" << std::endl;

while (!p1.estVide()) {

std::cout << p1.depiler() << std::endl;

}

std::cout << "\n Dépiler de p2 :" << std::endl;

while (!p2.estVide()) {

std::cout << p2.depiler() << std::endl;

}

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.

## solution :

#include <iostream>

#include <cstring>

using namespace std;

class Fichier {

private:

char\* P;

size\_t longueur;

public:

Fichier() : P(nullptr), longueur(0) {

cout << "Constructeur: Fichier initialisé.\n"; }

void Creation(size\_t taille) {

P = new char[taille];

longueur = taille;

cout << "Mémoire allouée: " << taille << " octets.\n";

}

void Remplit() {

if (P != nullptr) {

memset(P, 'A', longueur);

cout << "Mémoire remplie avec des valeurs arbitraires.\n";

} else {

cout << "Aucune mémoire n'a été allouée.\n";

}

}

void Affiche() const {

if (P != nullptr) {

cout << "Contenu de la mémoire : ";

for (size\_t i = 0; i < longueur; ++i) {

cout << P[i];

}

cout << '\n';

} else {

cout << "Aucune mémoire à afficher.\n";

}

}

~Fichier() {

if (P != nullptr) {

delete[] P;

cout << "Destructeur: Mémoire libérée.\n";

}

}

};

int main() {

Fichier\* fichier = new Fichier();

fichier->Creation(8);

fichier->Remplit();

fichier->Affiche();

delete fichier;

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.

## solution:

#include <iostream>

using namespace std;

struct Element {

int valeur;

Element\* suivant;

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

};

class Liste {

private:

Element\* premier;

public:

Liste() : premier(nullptr) {}

~Liste() {

vider();

}

void ajouterDebut (int valeur) {

Element\* nouveau = new Element(valeur);

nouveau->suivant = premier;

premier = nouveau;

}

void supprimerDebut() {

if (premier != nullptr) {

Element\* ancienPremier = premier;

premier = premier->suivant;

delete ancienPremier;

} else {

cout << "La liste est déjà vide !" <<endl;

}

}

void afficher() const {

Element\* actuel = premier;

while (actuel != nullptr) {

cout << actuel->valeur << " -> ";

actuel = actuel->suivant;

}

cout << "NULL" << endl;

}

void vider() {

while (premier != nullptr) {

supprimerDebut();

}

}

};

int main() {

Liste maListe;

maListe.ajouterDebut(5);

maListe.ajouterDebut(10);

maListe.ajouterDebut(20);

cout << "Contenu de la liste : ";

maListe.afficher();

maListe.supprimerDebut();

cout << "Après suppression : ";

maListe.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.

## solution:

#include <iostream>

#include <string>

using namespace std;

int main() {

string dateHeure = "010920091123";

if (dateHeure.length() != 12) {

cerr << "Erreur : La chaîne doit contenir exactement 12 caractères." << endl;

return 1;

}

string jour = dateHeure.substr(0, 2); // JJ

string mois = dateHeure.substr(2, 2); // MM

string annee = dateHeure.substr(4, 4); // AAAA

string heure = dateHeure.substr(8, 2); // HH

string minute = dateHeure.substr(10, 2); // NN

cout << "Date extraite : " << jour << "/" << mois << "/" << annee << endl;

cout << "Heure extraite : " << heure << ":" << minute <<endl;

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

## solution:

#include <iostream>

#include <vector>

#include <algorithm>

#include <limits>

using namespace std;

class Traitement {

private:

vector<int> vecteur;

public:

void Initialise() {

int num;

while (vecteur.size() < 15) {

cout << "entrez un entier pair non nul : ";

cin >> num;

if (cin.fail()) {

cin.clear(); cin.ignore(numeric\_limits<streamsize>::max(), '\n'); cout << "Entrée invalide. Veuillez en saisir un entier." <<endl;

continue;

}

if (num != 0 && num % 2 == 0) {

vecteur.push\_back(num);

} else {

cout << "Le nombre doit être pair et non nul." << endl;

}

}

}

void show(int index = 0) const {

if (index < vecteur.size()) {

cout << vecteur[index] << " ";

show(index + 1);

} else if (index == 0) {

cout << "Le vecteur est vide." << endl;

}

}

friend double moyenne(const Traitement &t);

friend double median(const Traitement &t);

};

double moyenne(const Traitement &t) {

if (t.vecteur.empty()) return 0.0;

double sum = 0;

for (int n : t.vecteur) {

sum += n;

}

return sum / t.vecteur.size();

}

double median(const Traitement &t) {

if (t.vecteur.empty()) return 0.0;

vector<int> sorted\_vect = t.vecteur;

sort(sorted\_vect.begin(), sorted\_vect.end());

size\_t size = sorted\_vect.size();

if (size % 2 == 0) {

return (sorted\_vect[size / 2 - 1] + sorted\_vect[size / 2]) / 2.0;

} else {

return sorted\_vect[size / 2];

}

}

int main() {

Traitement t;

t.Initialise();

cout << "Contenu du vecteur : ";

t.show();

cout << endl;

cout << "Moyenne : " << moyenne(t) << endl;

cout << "Médiane : " << median(t) << endl;

return 0;

}