Module : IDAI-54 Programmation Orientée Objet en C++/Java

ATELIER4

Botaina Ahadri

IDAI S5

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.

Solution :

```
#include <iostream>
using namespace std;
class Complexe {
private:
   double real;
   double imag;
public:
   Complexe(double r = 0, double i = 0) : real(r), imag(i) {}
   void setValues(double r, double i) {
      real = r;
      imag = i;
   }
   Complexe operator+(const Complexe& other) const {
      return Complexe(real + other.real, imag + other.imag);
   }
   Complexe operator-(const Complexe& other) const {
      return Complexe(real - other.real, imag - other.imag);
   }
   Complexe operator*(const Complexe& other) const {
      return Complexe(real * other.real - imag * other.imag, real * other.imag + imag * other.real);
   }


   Complexe operator/(const Complexe& other) const {
      double denominator = other.real * other.real + other.imag * other.imag;
```

```
        return Complexe((real * other.real + imag * other.imag) / denominator,
                        (imag * other.real - real * other.imag) / denominator);
    }
    bool operator==(const Complexe& other) const {
        return (real == other.real && imag == other.imag);
    }



    void afficher() const {
        cout << real << " + " << imag << "i" << endl;
    }
};

int main() {
    Complexe num1, num2, result;
    double real1, imag1, real2, imag2;
    int choice;

    cout << "Entrez la partie reelle et imaginaire du premier nombre complexe: ";
    cin >> real1 >> imag1;
    num1.setValues(real1, imag1);

    cout << "Entrez la partie reelle et imaginaire du deuxieme nombre complexe: ";
    cin >> real2 >> imag2;
    num2.setValues(real2, imag2);

    cout << "Choisissez l'operation:\n 1. Addition\n 2. Soustraction\n 3. Multiplication\n 4. Division\n
5. Égalité\n";
    cin >> choice;

    switch (choice) {
```

```
    case 1:
        result = num1 + num2;
        cout << "Resultat de l'addition: ";
        result.afficher();
        break;
    case 2:
        result = num1 - num2;
        cout << "Resultat de la soustraction: ";
        result.afficher();
        break;
    case 3:
        result = num1 * num2;
        cout << "Resultat de la multiplication: ";
        result.afficher();
        break;
    case 4:
        result = num1 / num2;
        cout << "Resultat de la division: ";
        result.afficher();
        break;
    case 5:
        if (num1 == num2)
            cout << "Les nombres complexes sont egaux." << endl;
        else
            cout << "Les nombres complexes ne sont pas egaux." << endl;
        break;
    default:
        cout << "Choix invalide." << endl;
}

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 en suite 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), Créer2 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 name;

   int age;


public:

   void set_value(string n, int a) {

      name = n;

      age = a;

   }


   virtual void afficher_info() = 0;

};


class Zebra : public Animal {

private:

   string origin;


public:
```

```
    Zebra(string o) : origin(o) {}

    void afficher_info() override {
        cout << "Zebra Name: " << name << ", Age: " << age << ", Origin: " << origin << endl;
    }
};

class Dolphin : public Animal {
private:
    string origin;

public:
    Dolphin(string o) : origin(o) {}

    void afficher_info() override {
        cout << "Dolphin Name: " << name << ", Age: " << age << ", Origin: " << origin << endl;
    }
};



int main() {
    string name;
    int age;
    string origin;


    cout << "Entrez le nom du zèbre: ";
    cin >> name;
    cout << "Entrez l'âge du zèbre: ";
    cin >> age;
```

```
    cout << "Entrez l'origine du zèbre: ";
    cin >> origin;
    Zebra zebra(origin);
    zebra.set_value(name, age);
    zebra.afficher_info();



    cout << "\n Entrez le nom du dauphin: ";
    cin >> name;
    cout << "Entrez l'âge du dauphin: ";
    cin >> age;
    cout << "Entrez l'origine du dauphin: ";
    cin >> origin;
    Dolphin dolphin(origin);
    dolphin.set_value(name, age);
    dolphin.afficher_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 des apropriété, 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 {
private:
   string nom;
   string prenom;
   string dateNaissance;

public:
   Personne(string n, string p, string d) : nom(n), prenom(p), dateNaissance(d) {}

   virtual void Afficher() {
      cout << "Nom: " << nom << ", Prénom: " << prenom << ", Date de Naissance: " << dateNaissance
<< endl;
   }
};

class Employe : public Personne {
private:
   double salaire;

public:
   Employe(string n, string p, string d, double s) : Personne(n, p, d), salaire(s) {}

   void Afficher() override {
      Personne::Afficher();
      cout << "Salaire: " << salaire << endl;
   }
};

class Chef : public Employe {
```

```
private:
    string service;

public:
    Chef(string n, string p, string d, double s, string ser) : Employe(n, p, d, s), service(ser) {}

    void Afficher() override {
        Employe::Afficher();
        cout << "Service: " << service << endl;
    }
};

class Directeur : public Chef {
private:
    string societe;

public:
    Directeur(string n, string p, string d, double s, string ser, string soc) : Chef(n, p, d, s, ser),
societe(soc) {}

    void Afficher() override {
        Chef::Afficher();
        cout << "Société: " << societe << endl;
    }
};
```

Exercice 4 :

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

−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 sont 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& v) const {
      return Vecteur3D(x + v.x, y + v.y, z + v.z);
   }
   float produitScalaire(const Vecteur3D& v) const {
      return x * v.x + y * v.y + z * v.z;
```

```
    }
    bool coincide(const Vecteur3D& v) const {
        return (x == v.x && y == v.y && z == v.z);
    }

    float norme() const {
        return sqrt(x * x + y * y + z * z);
    }
};
```

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 :

```
#ifndef PILE_H
#define PILE_H

class Pile {
private:
    int* elements; // Tableau dynamique pour stocker les éléments
    int taille;    // Capacité de la pile
    int sommet;    // Indice du sommet de la pile
```

```
public:
    Pile(int taille);   // Constructeur
    ~Pile();           // Destructeur
    void push(int element);  // Empiler un élément
    int pop();              // Dépiler un élément
    bool estVide() const;    // Vérifier si la pile est vide
    bool estPleine() const;  // Vérifier si la pile est pleine
};

#endif // PILE_
#include <iostream>
#include "Pile.h"
using namespace std;

Pile::Pile(int taille) : taille(taille), sommet(-1) {
    elements = new int[taille];
}

Pile::~Pile() {
    delete[] elements;
}

void Pile::push(int element) {
    if (estPleine()) {
        cout << "Erreur : La pile est pleine." << endl;
    } else {
        elements[++sommet] = element;
    }
}
int Pile::pop() {
    if (estVide()) {
```

```
        cout << "Erreur : La pile est vide." << endl;
        return -1;
    } else {
        return elements[sommet--];
    }
}

bool Pile::estVide() const {
    return sommet == -1;
}

bool Pile::estPleine() const {
    return sommet == taille - 1;
}
```

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;

void extraireDateHeure(const string& chaine) {
    if (chaine.length() != 12) {
        cout << "La chaîne doit contenir 12 caractères." << endl;
        return;
    }
```

```
    cout << "Date : " << chaine.substr(0, 2) << "/" << chaine.substr(2, 2) << "/" << chaine.substr(4, 4) << endl;
    cout << "Heure : " << chaine.substr(8, 2) << ":" << chaine.substr(10, 2) << endl;
}

int main() {
    string chaine = "010920091123";
    extraireDateHeure(chaine);
    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 :

```
#ifndef TRAITEMENT_H
#define TRAITEMENT_H

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

```
using namespace std;

class Traitement {
private:
   std::vector<int> vecteur;  // Stocke les entiers pairs

public:
   void Initialise();

   void show(int index = 0) const;

   friend double moyenne(const Traitement& t);
   friend double median(const Traitement& t);
};

#endif // TRAITEMENT_H

void Traitement::Initialis() {
   int valeur;
   vecteur.clear();  // Réinitialiser le vecteur
   cout << "Veuillez saisir 15 entiers pairs (différents de 0) : " << endl;


   while (vecteur.size() < 15) {
      cin >> valeur;


      if (cin.fail() || valeur % 2 != 0 || valeur == 0) {
         cin.clear(); // Efface le drapeau d'erreur
         cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignore l'entrée incorrecte
         cout << "Entrée invalide. Veuillez saisir un entier pair différent de 0 : " << endl;
```

```
        } else {
            vecteur.push_back(valeur);
        }
    }
}

void Traitement::show(int index) const {

    if (index < vecteur.size()) {
        cout << vecteur[index] << " ";
        show(index + 1);
    } else {
        cout << endl;
    }
}

double moyenne(const Traitement& t) {

    double sum = 0;
    for (int val : t.vecteur) {
        sum += val;
    }
    return sum / t.vecteur.size();
}

double median(const Traitement& t) {
    vector<int> sortedVecteur = t.vecteur;
    sort(sortedVecteur.begin(), sortedVecteur.end());

    int n = sortedVecteur.size();
    if (n % 2 == 0) {
```

```
        return (sortedVecteur[n / 2 - 1] + sortedVecteur[n / 2]) / 2.0;
    } else {
        return sortedVecteur[n / 2];
    }
}

int main() {
    Traitement t;

    t.Initialise();

    cout << "Valeurs dans le vecteur : ";
    t.show();

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

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

    return 0;
}
```