Atelier 4

El Kajdouhi Mohamed Ayman

elkajdouhi.mohamedayman@etu.uae.ac.ma

Exercice 1 :

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

class Complex {
private:
    double real;
    double imag;

public:
    Complex(double r = 0, double i = 0) : real(r), imag(i) {}

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

    Complex add(const Complex &other) const {
        return Complex(real + other.real, imag + other.imag);
    }

    Complex subtract(const Complex &other) const {
        return Complex(real - other.real, imag - other.imag);
    }

    Complex multiply(const Complex &other) const {
        return Complex(real * other.real - imag * other.imag,
                       real * other.imag + imag * other.real);
    }

    Complex divide(const Complex &other) const {
        double denominator = other.real * other.real + other.imag * other.imag;
        return Complex((real * other.real + imag * other.imag) / denominator,
                       (imag * other.real - real * other.imag) / denominator);
    }

    bool isEqual(const Complex &other) const {
        return (real == other.real) && (imag == other.imag);
    }
};

int main() {
    double real1, imag1, real2, imag2;
    int choice;

    cout << "Enter the real and imaginary parts of the first complex number: ";
    cin >> real1 >> imag1;
    Complex num1(real1, imag1);
    cout << "Enter the real and imaginary parts of the second complex number: ";
```

```
    cin >> real2 >> imag2;
    Complex num2(real2, imag2);
    cout << "\nChoose an operation:\n";
    cout << "1. Check Equality\n";
    cout << "2. Addition\n";
    cout << "3. Subtraction\n";
    cout << "4. Multiplication\n";
    cout << "5. Division\n";
    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice) {
        case 1:
            if (num1.isEqual(num2))
                cout << "The complex numbers are equal.\n";
            else
                cout << "The complex numbers are not equal.\n";
            break;

        case 2: {
            Complex result = num1.add(num2);
            cout << "Result of addition: ";
            result.display();
            break;
        }

        case 3: {
            Complex result = num1.subtract(num2);
            cout << "Result of subtraction: ";
            result.display();
            break;
        }

        case 4: {
            Complex result = num1.multiply(num2);
            cout << "Result of multiplication: ";
            result.display();
            break;
        }

        case 5: {
            Complex result = num1.divide(num2);
            cout << "Result of division: ";
            result.display();
            break;
        }

        default:
            cout << "Invalid choice!\n";
            break;
```

```
    }

    return 0;
}
```

Exercice 2 :

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

class Animal{

public:
    const char* name;
    int age;
    Animal(const char* myname) : name (myname){};
    void set_value(int age_value){age = age_value;};
};

class Zebra : public Animal{
public:
    const char* origin;
    Zebra (const char* myname, const char* myorigin) : Animal (myname), origin (myorigin){};
    void display_info(){cout << "Im: "<< name << " , my age is: " << age << " and im from:
"<< origin<< endl;};

};

class Dolphin : public Animal{
public:
    const char* origin;
    Dolphin (const char* myname, const char* myorigin) : Animal (myname), origin
(myorigin){};
    void display_info(){cout << "Im: "<< name << " , my age is: " << age << " and im from:
"<< origin<< endl;};

};

int main()
{
    const char* name = "shelly";
    const char* origin = "usa";
    Zebra zeb(name, origin);
    zeb.set_value(20);
    zeb.display_info();
    return 0;
}
```

Exercice 3 :

```cpp
#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() const {
        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() const 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 serv) : Employe(n, p, d, s),
service(serv) {}
```

```
    void Afficher() const 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 serv, string soc) : Chef(n, p,
d, s, serv), societe(soc) {}

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

int main() {

    Directeur directeur("El Kajdouhi", "Mohamed Ayman", "04/11/2004", 80000, "Développement",
"TechX");

    directeur.Afficher();

    return 0;
}
```

Exercice 4 :

```
#include<iostream>
#include<cmath>
using namespace std;
class vecteur3d {
    float x;
    float y;
    float z;

    public:

    vecteur3d(float a = 0, float b = 0, float c = 0) : x(a), y(b), z(c) {
    }
```

```
vecteur3d(const vecteur3d & v) {
    x = v.x;
    y = v.y;
    z = v.z;
}

void afficher() {
    cout << "("<<x<<","<<y<<","<<z<<")" << endl;
}

vecteur3d somme(const vecteur3d & v) {
    vecteur3d s;
    s.x = x + v.x;
    s.y = y + v.y;
    s.z = z + v.z;
    return s;
}

float produit(const vecteur3d & v) {
    return x*v.x + y*v.y + z*v.z;
}

bool coincide(const vecteur3d & v) {
    return (x == v.x && y == v.y && z == v.z);
}

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

vecteur3d normax(vecteur3d v) {
    if( this->norme() > v.norme())
        return *this;

    return v;
}

vecteur3d * normax(vecteur3d * v) {
    if( this->norme() > v->norme())
        return this;

    return v;
}

vecteur3d & normaxR(vecteur3d &v) {
    if( this->norme() > v.norme())
        return *this;

    return v;
}
```

```
};

int main() {
    vecteur3d v1(6,2,9);
    cout << "Vecteur V1";
    v1.afficher();
    vecteur3d v2(4,1,7);
    cout << "Vecteur V2";
    v2.afficher();
    cout<<endl;
    cout << "La somme des vecteurs v1 et v2 est : ";
    (v1.somme(v2)).afficher();
    cout << "Le produit scalaire des vecteurs v1 et v2 est : " << v1.produit(v2) << endl;
    cout<<endl;
    cout << "Copier le vecteur V1 dans V3:" << endl;
    vecteur3d v3(v1);
    cout << "Vecteur V3";
    v3.afficher();
    if(v1.coincide(v3))
        cout << "Les vecteurs v1 et v3 coincident " << endl;
    else
        cout << "Les vecteurs v1 et v3 ne coincident pas " << endl;

    cout<<endl;
    cout << "Le vecteur qui a la plus grande norme est (par valeur): ";
    (v1.normax(v2)).afficher();
    cout << "Le vecteur qui a la plus grande norme est (par adresse): ";
    (v1.normax(&v2))->afficher();
    cout << "Le vecteur qui a la plus grande norme est (par reference) :";
    (v1.normaxR(v2)).afficher();
    cout<<endl;
}
```

Exercice 5 :

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

class Test {
public:

    void call() {
        static int count = 0;
        count++;
        cout << "La fonction call a été appelée " << count << " fois." << endl;
    }
};

int main() {
```

```
    Test test;


    test.call();
    test.call();
    test.call();
    test.call();
    test.call();
    test.call();

    return 0;
}
```

Exercice 6 :

1. Fichier d'en-tête : `point.h`

```
#ifndef POINT_H
#define POINT_H

class point {
private:
   float x;
   float y;

public:
   void saisir();
   void deplacer(float a, float b);
   void afficher();
};

#endif // POINT_H
```

2. Fichier source : `point.cpp`

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

void point::saisir() {
    cout << "Entrer l'abscisse x : ";
    cin >> x;
    cout << "Entrer l'ordonnée y : ";
    cin >> y;
```

```
}

void point::deplacer(float a, float b) {
    x += a;
    y += b;
}

void point::afficher() {
    cout << "Les coordonnes: P(" << x << "," << y << ")" << endl;
}
```

3. Fichier principal : `main.cpp`

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

int main() {
    point p;
    p.saisir();
    p.afficher();

    p.deplacer(1.5, 2);
    cout << "Les coordonnes du point apres
deplacement :" << endl;
    p.afficher();

    return 0;
}
```

Exercice 7 :

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

using namespace std;

class pile {
private:
    int limit;
    int taille;
    int *elements;

public:
    pile(const size_t &taille);
    ~pile();
    void push(const int &);
    int pop();
};

pile::pile(const size_t &taille) {
    elements = (int *)malloc(sizeof(int) * taille);
    this->taille = taille;
    limit = 0;
}

void pile::push(const int &src) {
    if (limit < taille) {
        elements[limit] = src;
        limit += 1;
    } else {
        cerr << "La pile est pleine, impossible d'ajouter l'élément." << endl;
    }
}

int pile::pop() {
    if (limit > 0) {
        limit -= 1;
        return elements[limit];
    } else {
        cerr << "La pile est vide, impossible de dépiler." << endl;
        return -1;
    }
}

pile::~pile() {
```

```
    free(elements);
}

int main() {
    pile p1(5);
    pile p2(3);

    p1.push(10);
    p1.push(20);
    p1.push(30);

    p2.push(100);
    p2.push(200);

    cout << "Dépilement de p1: " << p1.pop() << endl;
    cout << "Dépilement de p1: " << p1.pop() << endl;
    cout << "Dépilement de p1: " << p1.pop() << endl;
    cout << "Dépilement de p1: " << p1.pop() << endl;

    cout << "Dépilement de p2: " << p2.pop() << endl;
    cout << "Dépilement de p2: " << p2.pop() << endl;
    cout << "Dépilement de p2: " << p2.pop() << endl;

    return 0;
}
```

Exercice 8 :

```
#include <iostream>

using namespace std;

class Fichier
{
    char* p;
    unsigned int taille_buffer;

public:
    Fichier();
    ~Fichier();
    bool creation(unsigned int);
    void remplit();
    void affiche();
};
```

```cpp
Fichier::Fichier()
{
    p = NULL;
    taille_buffer = 0;
}

Fichier::~Fichier()
{
    delete[] p;
}

bool Fichier::creation(unsigned int taille)
{
    if ((p = new char[taille])==NULL) return false;
    taille_buffer = taille;
    return true;
}

void Fichier::remplit()
{
    for(unsigned int i=0 ; i<taille_buffer ; i++) p[i]='a';
}

void Fichier::affiche()
{
    for(unsigned int i=0 ; i<taille_buffer ; i++) cout<<p[i];
}

int main()
{
    Fichier* f = new Fichier();
    if (f->creation(10))
    {
        f->remplit();
        f->affiche();
    }
    delete f;
    return 0;
}
```

Exercice 9 :

```cpp
#include <iostream>

using namespace std;
```

```cpp
struct Element {
    int valeur;
    Element *suivant;

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

class Liste {
private:
    Element *tete;

public:
    Liste() : tete(nullptr) {}
    ~Liste();

    void ajouterDebut(int val);
    void supprimerDebut();
    void afficher() const;
};

Liste::~Liste() {
    while (tete != nullptr) {
        supprimerDebut();
    }
}

void Liste::ajouterDebut(int val) {
    Element *nouveau = new Element(val);
    nouveau->suivant = tete;
    tete = nouveau;
}

void Liste::supprimerDebut() {
    if (tete != nullptr) {
        Element *temp = tete;
        tete = tete->suivant;
        delete temp;
    } else {
        cout << "La liste est déjà vide." << endl;
    }
}

void Liste::afficher() const {
    Element *current = tete;
    while (current != nullptr) {
        cout << current->valeur << " -> ";
        current = current->suivant;
    }
    cout << "nullptr" << endl;
}
```

```
int main() {
    Liste maListe;

    maListe.ajouterDebut(10);
    maListe.ajouterDebut(20);
    maListe.ajouterDebut(30);

    cout << "Liste après ajouts : ";
    maListe.afficher();

    maListe.supprimerDebut();
    cout << "Liste après suppression : ";
    maListe.afficher();

    return 0;
}
```

Exercice 10 :

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;
void afficherDateEtHeure(const string& s)
{
if ( s.length() != 12 )
cerr << "Chaine invalide." << endl;
else
{
cout << "Date : " << s.substr(0,2) << "/" << s.substr(2,2) << "/" << s.substr(4,4) <<endl;
cout << "Heure : " << s.substr(8,2) << "h" << s.substr(10,2) << endl;
}
}
int main(int argc, char** argv)
{
string s("010920091123");
afficherDateEtHeure(s); // exemple
}
```

Exercice 11 :

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

using namespace std;

class Traitement {
private:
    vector<int> valeurs;

public:
    void Initialise();
    void show(int index = 0) const;

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

void Traitement::Initialise() {
    valeurs.clear();
    cout << "Entrez 15 entiers pairs et non-nuls:" << endl;
    while (valeurs.size() < 15) {
        int val;
        cout << "Entier #" << (valeurs.size() + 1) << ": ";
        cin >> val;


        if (cin.fail() || val == 0 || val % 2 != 0) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Veuillez entrer un nombre entier pair et non-nul." << endl;
        } else {
            valeurs.push_back(val);
        }
    }
}

void Traitement::show(int index) const {
    if (index >= valeurs.size()) {
        cout << endl;
        return;
    }
    cout << valeurs[index] << " ";
    show(index + 1);
```

```cpp
}

double moyenne(const Traitement& t) {
    if (t.valeurs.empty()) return 0.0;
    int somme = 0;
    for (int val : t.valeurs) {
        somme += val;
    }
    return static_cast<double>(somme) / t.valeurs.size();
}

double median(const Traitement& t) {
    if (t.valeurs.empty()) return 0.0;
    vector<int> temp = t.valeurs;
    sort(temp.begin(), temp.end());

    int n = temp.size();
    if (n % 2 == 0) {
        return (temp[n / 2 - 1] + temp[n / 2]) / 2.0;
    } else {
        return temp[n / 2];
    }
}

int main() {
    Traitement t;
    t.Initialise();

    cout << "Les éléments du vecteur sont : ";
    t.show();

    cout << "La moyenne des éléments est : " << moyenne(t) << endl;
    cout << "La médiane des éléments est : " << median(t) << endl;

    return 0;
}
```