TD/TP03 Solution

Solution 01

```
#include <iostream>
using namespace std;
class Rectangle {
private:
    double longueur;
    double largeur;
public:
    // Constructeur avec arguments
    Rectangle(double l, double L) : longueur(l), largeur(L) {}

    // Méthode pour calculer la surface
    double surface() const { return longueur * largeur; }

    // Méthode pour afficher les dimensions et la surface
    void afficher() const {
        cout << "Rectangle [Longueur: " << longueur << ", Largeur: " << largeur <<
"]" << endl;
        cout << "Surface: " << surface() << endl; }
};
int main() {
    double l, L;
    cout << "Entrez la longueur du rectangle : ";
    cin >> l;
    cout << "Entrez la largeur du rectangle : ";
    cin >> L;
    Rectangle rect(l, L);
    rect.afficher();
    return 0;
}
```

Solution 02

```
#include <iostream>
using namespace std;
class Somme {
private:
    int n1, n2;
public:
    // Constructeur avec arguments
    Somme(int a, int b) : n1(a), n2(b) {}
    // Méthode pour calculer la somme
    int calculer() const { return n1 + n2; }
    // Méthode pour afficher la somme
    void afficher() const {
        cout << "Somme de " << n1 << " et " << n2 << " est : " << calculer() << endl; }
};
int main() {
    int a, b;
    cout << "Entrez le premier nombre : ";   cin >> a;
    cout << "Entrez le deuxième nombre : ";  cin >> b;
    Somme s(a, b);
    s.afficher();
    return 0;
}
```

TD/TP03 Solution

Solution 03

```
#include <iostream>
#include <cstring>
using namespace std;
class Student {
private:
    char nom[50];
    double note1, note2;
public:
    // Constructeur avec arguments
    Student(const char* n, double n1, double n2) {
        strncpy(nom, n, sizeof(nom));
        nom[sizeof(nom) - 1] = '\0'; // Assurez que la chaîne est bien terminée
        note1 = n1;
        note2 = n2;
    }
    // Méthode pour calculer la moyenne
    double calc_moyenne() const { return (note1 + note2) / 2.0; }
    // Méthode pour afficher les informations
    void afficher() const {
        cout << "Nom : " << nom << endl;
        cout << "Moyenne : " << calc_moyenne() << endl; }
};
int main() {
    char nom[50];
    double note1, note2;
    cout << "Entrez le nom de l'étudiant : ";  cin >> nom;
    cout << "Entrez la première note : ";      cin >> note1;
    cout << "Entrez la deuxième note : ";      cin >> note2;
    Student etudiant(nom, note1, note2);
    etudiant.afficher();
    return 0;
}
```

L'utilisation de **strncpy** (string copy) en C++ est une méthode pour copier des chaînes de caractères de manière sécurisée, particulièrement utile lorsqu'on manipule des tableaux de caractères (type char) dans des classes ou des structures.

Solution 04

```
#include <iostream>
#include <cmath> // Pour sqrt()
using namespace std;
class Complex {
private:
    double Re;  // Partie réelle
    double Img; // Partie imaginaire
public:
    // Constructeur avec arguments (valeurs par défaut)
    Complex(double r = 0, double i = 0) : Re(r), Img(i) {}
    // Méthode pour afficher un nombre complexe
    void afficher() const {
        cout << Re;
        if (Img >= 0)  cout << " + " << Img << "i";
        else  cout << " - " << -Img << "i";
        cout << endl; }
```

TD/TP03 Solution

```
    // Méthode pour calculer le module
    double module() const { return sqrt(Re * Re + Img * Img); }
    // Méthode pour retourner le conjugué
    Complex conjugue() const { return Complex(Re, -Img); }
    // Surcharge de l'opérateur +
    Complex operator+(const Complex& o) const {
        return Complex(Re + o.Re, Img + o.Img); }
    // Surcharge de l'opérateur -
    Complex operator-(const Complex& o) const {
        return Complex(Re - o.Re, Img - o.Img); }
    // Surcharge de l'opérateur *
    Complex operator*(const Complex& o) const {
        return Complex(Re * o.Re - Img * o.Img, Re * o.Img + Img * o.Re); }
    // Surcharge de l'opérateur /
    Complex operator/(const Complex& o) const {
        double denom = o.Re * o.Re + o.Img * o.Img;
        if (denom == 0) {
            throw runtime_error("Division par zéro !");
        }
        return Complex((Re * o.Re + Img * o.Img) / denom, (Img * o.Re - Re * o.Img)
/ denom); }
};

int main() {
    // Variables pour les parties réelles et imaginaires des deux nombres complexes
    double r1, i1, r2, i2;

    // Saisie des deux nombres complexes
    cout << "Entrez la partie réelle du premier nombre complexe : ";     cin >> r1;
    cout << "Entrez la partie imaginaire du premier nombre complexe : "; cin >> i1;

    cout << "Entrez la partie réelle du deuxième nombre complexe : ";    cin >> r2;
    cout << "Entrez la partie imaginaire du deuxième nombre complexe : ";cin >> i2;

    // Création des deux nombres complexes
    Complex c1(r1, i1);
    Complex c2(r2, i2);

    // Affichage des nombres complexes
    cout << "Premier nombre complexe : "; c1.afficher();
    cout << "Module : " << c1.module() << endl;
    cout << "Conjugué : ";  c1.conjugue().afficher();

    cout << "Deuxième nombre complexe : "; c2.afficher();
    cout << "Module : " << c2.module() << endl;
    cout << "Conjugué : "; c2.conjugue().afficher();

    // Opérations entre nombres complexes
    cout << "\nAddition : ";      (c1 + c2).afficher();
    cout << "Soustraction : ";    (c1 - c2).afficher();
    cout << "Multiplication : ";  (c1 * c2).afficher();
    try {
        cout << "Division : ";    (c1 / c2).afficher();
    } catch (const runtime_error& e) { cout << e.what() << endl; }
    return 0;
}
```

TD/TP03 Solution

Solution 05

```
#include <iostream>
#include <cmath> // Pour sqrt()
using namespace std;
class Point {
private:
    int x; // Coordonnée x
    int y; // Coordonnée y
public:
    // Constructeur
    Point(int xCoord, int yCoord) : x(xCoord), y(yCoord) {
        cout << "Construction du point (" << x << ", " << y << ") à l'adresse : " <<
this << endl; }
    // Constructeur par copie
    Point(const Point& other) : x(other.x), y(other.y) {
        cout << "Construction par copie du point (" << x << ", " << y << ") à
l'adresse : " << this << endl; }
    // Destructeur
    ~Point() {
        cout << "Destruction du point (" << x << ", " << y << ") à l'adresse : " <<
this << endl; }
    // Méthode pour calculer la distance entre deux points
    double distance(const Point& other) const {
        return sqrt(pow(other.x - x, 2) + pow(other.y - y, 2)); }
    // Méthode pour afficher les coordonnées du point
    void afficher() const { cout << "(" << x << ", " << y << ")"; }
};
int main() {
    // Création d'un point a
    Point a(3, 4);
    // Création d'un point b par copie
    Point b = a;
    // Affichage des deux points
    cout << "Point a : "; a.afficher(); cout << endl;
    cout << "Point b : "; b.afficher(); cout << endl;
    // Calcul et affichage de la distance entre a et b
    cout << "Distance entre a et b : " << a.distance(b) << endl;
    return 0;
}
```

Solution 06

```
#include <iostream>
using namespace std;
class Tableau {
private:
    int taille;       // Taille du tableau
    float* elements;  // Tableau dynamique
public:
    // Constructeur avec paramètre
    Tableau(int t) : taille(t) {
        elements = new float[taille];
        cout << "Construction d'un tableau de taille " << taille << " à l'adresse :
" << this << endl;
    }


    // Constructeur par copie
```

TD/TP03 Solution

```
    Tableau(const Tableau& autre) : taille(autre.taille) {
        elements = new float[taille];
        for (int i = 0; i < taille; i++) { elements[i] = autre.elements[i]; }
        cout << "Construction par copie d'un tableau à l'adresse : " << this <<
endl;
    }
    // Destructeur
    ~Tableau() {
        cout << "Destruction du tableau à l'adresse : " << this << endl;
        delete[] elements;
    }
    // Méthode pour remplir le tableau
    void saisir() {
        cout << "Entrez les éléments du tableau : ";
        for (int i = 0; i < taille; i++) { cin >> elements[i]; }
    }
    // Méthode pour afficher le contenu du tableau
    void afficher() const {
        cout << "Contenu du tableau : ";
        for (int i = 0; i < taille; i++) { cout << elements[i] << " "; }
        cout << endl;
    }

    // Méthode pour générer un tableau avec les valeurs opposées
    Tableau opposé() const {
        Tableau tOppose(taille);
        for (int i = 0; i < taille; i++) { tOppose.elements[i] = -elements[i]; }
        return tOppose;
    }
};

int main() {
    int taille;
    // Demander à l'utilisateur de spécifier la taille
    cout << "Entrez la taille du tableau : ";  cin >> taille;
    // Création d'un tableau
    Tableau t1(taille);   t1.saisir();  t1.afficher();
    // Copie du tableau
    Tableau t2 = t1;
    cout << "Contenu du tableau copié : "; t2.afficher();
    // Tableau opposé
    Tableau tOppose = t1.opposé();
    cout << "Contenu du tableau opposé : "; tOppose.afficher();
    return 0;
}
```