TD/TP04 Solution

Exercice 01 :

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

// Patron de fonction pour calculer le maximum entre deux valeurs
template <typename T>
T trouverMax(T a, T b) {
    return (a > b) ? a : b; // Retourne la plus grande valeur
}

int main() {
    // Test avec des entiers
    int intA = 10, intB = 20;
    cout << "Le maximum entre " << intA << " et " << intB << " est : "
         << trouverMax(intA, intB) << endl;

    // Test avec des doubles
    double doubleA = 5.6, doubleB = 3.14;
    cout << "Le maximum entre " << doubleA << " et " << doubleB << " est : "
         << trouverMax(doubleA, doubleB) << endl;

    // Test avec des caractères
    char charA = 'A', charB = 'Z';
    cout << "Le maximum entre '" << charA << "' et '" << charB << "' est : '"
         << trouverMax(charA, charB) << "'" << endl;

    return 0;
}
```

Exercice 02 :

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

// Patron de fonction pour calculer la moyenne d'un tableau
template <typename T>
T calculerMoyenne(T tableau[], int taille) {
    T somme = 0;
    for (int i = 0; i < taille; ++i) {
        somme += tableau[i];
    }
    return somme / taille; // Retourne la moyenne
}

int main() {
    // Test avec un tableau d'entiers
    int tableauInt[] = {1, 2, 3, 4, 5};
    int tailleInt = sizeof(tableauInt) / sizeof(tableauInt[0]);
    cout << "Moyenne des entiers : " << calculerMoyenne(tableauInt, tailleInt) <<
endl;

    // Test avec un tableau de flottants
    float tableauFloat[] = {1.2f, 3.4f, 5.6f, 7.8f};
    int tailleFloat = sizeof(tableauFloat) / sizeof(tableauFloat[0]);
```

TD/TP04 Solution

```
    cout << "Moyenne des flottants : " << calculerMoyenne(tableauFloat, tailleFloat) << endl;

    // Test avec un tableau de doubles
    double tableauDouble[] = {1.11, 2.22, 3.33, 4.44, 5.55};
    int tailleDouble = sizeof(tableauDouble) / sizeof(tableauDouble[0]);
    cout << "Moyenne des doubles : " << calculerMoyenne(tableauDouble, tailleDouble) << endl;

    return 0;
}
```

Exercice 03 :

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

// Patron de classe Pair pour stocker deux éléments de même type
template <typename T>
class Pair {
private:
    T first; // Premier élément
    T second; // Deuxième élément

public:
    // Constructeur pour initialiser les deux valeurs
    Pair(T val1, T val2) : first(val1), second(val2) {}

    // Méthode pour afficher les deux valeurs
    void display() const {
        cout << "First: " << first << ", Second: " << second << endl;
    }

    // Méthode pour échanger les deux valeurs
    void swapValues() {
        T temp = first;
        first = second;
        second = temp;
    }
};

int main() {
    // Test avec des entiers
    cout << "Test avec des entiers:" << endl;
    Pair<int> intPair(10, 20);
    intPair.display();
    intPair.swapValues();
    cout << "Après échange : ";
    intPair.display();

    // Test avec des doubles
    cout << "\nTest avec des doubles:" << endl;
    Pair<double> doublePair(3.14, 2.718);
    doublePair.display();
    doublePair.swapValues();
```

TD/TP04 Solution

```
    cout << "Après échange : ";
    doublePair.display();

    // Test avec des chaînes de caractères
    cout << "\nTest avec des chaînes de caractères:" << endl;
    Pair<string> stringPair("Hello", "World");
    stringPair.display();
    stringPair.swapValues();
    cout << "Après échange : ";
    stringPair.display();

    return 0;
}
```

Exercice 04 :

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

// Patron de classe pour stocker une valeur de n'importe quel type
template <typename T>
class Box {
private:
    T value; // La valeur stockée

public:
    // Constructeur pour initialiser la valeur
    Box(T val) : value(val) {}

    // Méthode pour renvoyer la valeur stockée
    T getValue() const {
        return value;
    }

    // Méthode pour changer la valeur
    void setValue(T val) {
        value = val;
    }
};

int main() {
    // Test avec un entier
    Box<int> intBox(42);
    cout << "Valeur initiale de intBox : " << intBox.getValue() << endl;
    intBox.setValue(100);
    cout << "Nouvelle valeur de intBox : " << intBox.getValue() << endl;

    // Test avec un double
    Box<double> doubleBox(3.14);
    cout << "\nValeur initiale de doubleBox : " << doubleBox.getValue() << endl;
    doubleBox.setValue(2.718);
    cout << "Nouvelle valeur de doubleBox : " << doubleBox.getValue() << endl;

    // Test avec un caractère
    Box<char> charBox('A');
    cout << "\nValeur initiale de charBox : " << charBox.getValue() << endl;
```

TD/TP04 Solution

```
    charBox.setValue('Z');
    cout << "Nouvelle valeur de charBox : " << charBox.getValue() << endl;

    return 0;
}
```

Exercice 05 :

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

// Patron de classe pour gérer un tableau générique
template <typename T>
class Array {
private:
    T* elements;  // Pointeur vers le tableau
    int size;     // Taille du tableau

public:
    // Constructeur pour initialiser le tableau avec une taille spécifiée
    Array(int taille) : size(taille) {
        elements = new T[size];
    }

    // Destructeur pour libérer la mémoire allouée dynamiquement
    ~Array() {
        delete[] elements;
    }

    // Méthode pour remplir le tableau
    void remplir() {
        cout << "Entrez " << size << " éléments du tableau : " << endl;
        for (int i = 0; i < size; i++) {
            cout << "Élément [" << i << "] : ";
            cin >> elements[i];
        }
    }

    // Méthode pour afficher le contenu du tableau
    void afficher() const {
        cout << "Contenu du tableau : ";
        for (int i = 0; i < size; i++) {
            cout << elements[i] << " ";
        }
        cout << endl;
    }

    // Méthode pour trouver la plus grande valeur dans le tableau
    T trouverMax() const {
        T max = elements[0];
        for (int i = 1; i < size; i++) {
            if (elements[i] > max) {
                max = elements[i];
            }
        }
        return max;
```

TD/TP04 Solution

```
    }
};

int main() {
    // Test avec des entiers
    cout << "Test avec un tableau d'entiers :\n";
    Array<int> tableauInt(5);
    tableauInt.remplir();
    tableauInt.afficher();
    cout << "La plus grande valeur est : " << tableauInt.trouverMax() << endl;

    // Test avec des flottants
    cout << "\nTest avec un tableau de flottants :\n";
    Array<float> tableauFloat(5);
    tableauFloat.remplir();
    tableauFloat.afficher();
    cout << "La plus grande valeur est : " << tableauFloat.trouverMax() << endl;

    return 0;
}
```

Exercices 06 :

```
#include <iostream>
#include <iomanip> // Pour formater l'affichage
using namespace std;

class Time {
private:
    int hours;   // Heures
    int minutes; // Minutes
    int seconds; // Secondes

public:
    // Constructeur par défaut (initialise à 0)
    Time() : hours(0), minutes(0), seconds(0) {}

    // Constructeur avec valeurs fixes
    Time(int h, int m, int s) : hours(h), minutes(m), seconds(s) {}

    // Fonction membre pour afficher l'heure au format HH:MM:SS
    void getTime() const {
        cout << setfill('0') << setw(2) << hours << ":"
             << setw(2) << minutes << ":"
             << setw(2) << seconds << endl;
    }

    // Fonctions membres pour obtenir chaque membre (getters)
    int getHours() const { return hours; }
    int getMin() const { return minutes; }
    int getSec() const { return seconds; }

    // Fonction membre pour ajouter deux objets Time
    void addTime(const Time& t1, const Time& t2) {
        int totalSeconds = t1.seconds + t2.seconds;
        int totalMinutes = t1.minutes + t2.minutes + totalSeconds / 60;
```

TD/TP04 Solution

```
        hours = t1.hours + t2.hours + totalMinutes / 60;
        minutes = totalMinutes % 60;
        seconds = totalSeconds % 60;
    }
};

int main() {
    // Initialisation des objets Time
    Time t1(4, 45, 59);
    Time t2(1, 20, 32);
    Time t3;

    // Affichage des temps initiaux
    cout << "Temps T1 : ";
    t1.getTime(); // 04:45:59

    cout << "Temps T2 : ";
    t2.getTime(); // 01:20:32

    // Ajout des deux temps
    t3.addTime(t1, t2);

    // Affichage du résultat de l'addition
    cout << "Temps T3 (T1 + T2) : ";
    t3.getTime(); // 06:06:31

    return 0;
}
```

Exercice 07 :

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

// Classe Account pour la gestion d'un compte bancaire
class Account {
private:
    double balance; // Solde du compte

public:
    // Constructeur par défaut : solde initial à zéro
    Account() : balance(0.0) {}

    // Constructeur avec un solde initial comme paramètre
    Account(double initialBalance) : balance(initialBalance) {}

    // Méthode pour obtenir le solde actuel
    double getBalance() const {
        return balance;
    }

    // Méthode pour déposer un montant spécifié
    void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        } else {
```

TD/TP04 Solution

```
            cout << "Le montant du dépôt doit être positif.\n";
        }
    }

    // Méthode pour retirer un montant spécifié
    void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
        } else {
            cout << "Retrait impossible : montant invalide ou solde
insuffisant.\n";
        }
    }

    // Méthode pour ajouter de l'intérêt au compte
    void addInterest(double rate) {
        if (rate > 0) {
            balance *= (1 + rate);
        } else {
            cout << "Le taux d'intérêt doit être positif.\n";
        }
    }
};

int main() {
    // Création des comptes bancaires
    Account account1;              // Compte avec solde initial à 0
    Account account2(3000.0);      // Compte avec solde initial de 3000.0

    // Opérations sur account1
    account1.deposit(100);         // Dépôt de 100 dans account1
    account1.addInterest(0.3);     // Ajout d'un intérêt de 30% à account1

    // Opérations sur account2
    account2.withdraw(1000);       // Retrait de 1000 dans account2

    // Affichage des soldes finaux
    cout << "Solde de account1 : " << account1.getBalance() << "\n";
    cout << "Solde de account2 : " << account2.getBalance() << "\n";

    return 0;
}
```

Exercice 08 :

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

// Classe mère A
class A {
public:
    // Méthode display de la classe mère
    void display() {
        cout << "Ceci est la méthode display() de la classe mère A." << endl;
    }
};
```

TD/TP04 Solution

```
// Classe fille B héritant de la classe A
class B : public A {
public:
    // Redéfinition de la méthode display dans la classe fille
    void display() {
        cout << "Ceci est la méthode display() de la classe fille B." << endl;
    }
};

int main() {
    // Création d'un objet de la classe fille B
    B obj;

    // Appel de la méthode display()
    obj.display();

    return 0;
}
```

Exercice 09 :

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

// Classe de base Shape
class Shape {
protected:
    double x, y; // Largeur et hauteur
public:
    // Constructeur pour initialiser largeur et hauteur
    Shape(double largeur, double hauteur) : x(largeur), y(hauteur) {}

    // Méthode virtuelle pure pour calculer l'aire
    virtual double area() const = 0;
};

// Classe Rectangle dérivée de Shape
class Rectangle : public Shape {
public:
    // Constructeur pour Rectangle
    Rectangle(double largeur, double hauteur) : Shape(largeur, hauteur) {}

    // Implémentation de la méthode area
    double area() const override {
        return x * y; // Aire du rectangle = largeur * hauteur
    }
};

// Classe Triangle dérivée de Shape
class Triangle : public Shape {
public:
    // Constructeur pour Triangle
    Triangle(double largeur, double hauteur) : Shape(largeur, hauteur) {}

    // Implémentation de la méthode area
```

TD/TP04 Solution

```
    double area() const override {
        return (x * y) / 2; // Aire du triangle = largeur * hauteur / 2
    }
};

int main() {
    Rectangle rectangle(2, 3); // Largeur = 2, Hauteur = 3
    Triangle triangle(2, 3);   // Largeur = 2, Hauteur = 3

    cout << rectangle.area() << endl; // Affiche 6
    cout << triangle.area() << endl;  // Affiche 3

    return 0;
}
```