Nom et Prénom : NAWRAS GRAOUI

Atelier 2

Exercice 1

Écrire une fonction, sans argument ni valeur de retour, qui se contente d'afficher, à chaque appel, le nombre total de fois où elle a été appelée sous la forme : ***appel numéro 3***

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

void nombres_appels() {
    static int afficher = 0;
    // Le mot-clé 'static' permet de garder la valeur entre les appels
    /* Après la première initialisation,
    la variable garde sa valeur même lorsque la fonction termine.
    Lors du prochain appel de la fonction,
    la variable aura la valeur
    qu'elle avait lors du dernier appel.*/
    afficher++;  // On incrémente le compteur à chaque appel
    cout << "Appel numero " << afficher << endl;
}

int main() {
    nombres_appels();  // Appel 1
    nombres_appels();  // Appel 2
    nombres_appels();  // Appel 3
    nombres_appels();  // Appel 4

    return 0;
}
```

Exercice 2

Écrire 2 fonctions à un argument **entier** et une valeur de retour **entière** permettant de préciser si l'argument reçu est multiple de **2** (pour la première fonction) ou multiple de **3** (pour la seconde fonction).
Utiliser ces deux fonctions dans un petit programme qui lit un nombre entier et qui précise s'il est pair, multiple de 3 et/ou multiple de 6, comme dans cet exemple (il y a deux exécutions) :

donnez un entier : 9
il est multiple de 3
---
donnez un entier : 12
il est pair
il est multiple de 3
il est divisible par 6

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

// Fonction qui vérifie si un nombre est pair (multiple de 2)
bool est_pair(int nombre) {
    return nombre % 2 == 0;
}

// Fonction qui vérifie si un nombre est multiple de 3
bool multiple_3(int nombre) {
    return nombre % 3 == 0;
}

int main() {
    int nombre;

    cout << "Donnez un entier : ";
    cin >> nombre;

    // Vérification
    if (est_pair(nombre)) {
        cout << "Il est pair" << endl;
    }

    if (multiple_3(nombre)) {
        cout << "Il est multiple de 3" << endl;
    }

    // Vérifier s'il est multiple de 6 (multiple de 2 ET de 3)
    if (est_pair(nombre) && multiple_3(nombre)) {
        cout << "Il est divisible par 6" << endl;
    }

    return 0;
}

```

Exercice 3

Écrire, de deux façons différentes, un programme qui lit **10 nombres entiers** dans un tableau avant d'en rechercher le plus grand et le plus petit :

a. en utilisant uniquement le « *formalisme tableau* » ;

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

int main() {
    int nombres[10]; // Déclaration
    int max, min;

    cout << "Entrez 10 nombres entiers : " << endl;
    for (int i = 0; i < 10; i++) {
        cin >> nombres[i];
    }

    // Initialisation de max et min avec le premier élément
    max = nombres[0];
    min = nombres[0];

    // Recherche du max et du min
    for (int i = 1; i < 10; i++) {
        if (nombres[i] > max) {
            max = nombres[i];
        }
        if (nombres[i] < min) {
            min = nombres[i];
        }
    }

    //résultats
    cout << "Le plus grand nombre est : " << max << endl;
    cout << "Le plus petit nombre est : " << min << endl;

    return 0;
}
```

b. en utilisant le « *formalisme pointeur* », à chaque fois que cela est possible.

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

int main() {
    int nombres[10]; // Déclaration
    int *p = nombres; // Pointeur pointant vers le début du tableau
    int max, min;
    cout << "Entrez 10 nombres entiers : " << endl;
    for (int i = 0; i < 10; i++) {
        cin >> *(p + i);
 // Utilisation du pointeur pour accéder aux éléments
    }

    // Initialisation
    max = *p;
    min = *p; // Premier élément du tableau

    for (int i = 1; i < 10; i++) {
        if (*(p + i) > max) {
            max = *(p + i);
        }
        if (*(p + i) < min) {
            min = *(p + i);
        }
    }

    // résultats
    cout << "Le plus grand nombre est : " << max << endl;
    cout << "Le plus petit nombre est : " << min << endl;

    return 0;
}
```

Exercice 4

Écrire un programme **allouant dynamiquement** un emplacement pour un **tableau d'entiers**, dont la taille est fournie en donnée.
1. Utiliser ce tableau pour y placer des nombres entiers lus également en donnée.
2. Créer ensuite dynamiquement un nouveau tableau destiné à recevoir les carrés des nombres contenus dans le premier.
3. Supprimer le premier tableau, afficher les valeurs du second et supprimer le tout.

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

int main() {
    // Étape 1 : Demander la taille et allouer le tableau dynamiquement
    int taille;
    cout << "Entrez la taille du tableau : ";
    cin >> taille;
    int* tableau1 = new int[taille];
 // Allocation dynamique du premier tablea
u
    // Étape 2 : Lire les nombres dans le tableau
    for (int i = 0; i < taille; i++) {
        cout << "Entrez un nombre : ";
        cin >> tableau1[i];
    }

    // Étape 3 : Créer le second tableau
    int* tableau2 = new int[taille];
 // Allocation dynamique du second tableau

    // Étape 4 : Calculer les carrés et les stocker dans le second tablea
u
    for (int i = 0; i < taille; i++) {
        tableau2[i] = tableau1[i] * tableau1[i]; // Calcul du carré
    }

    // Étape 5 : Afficher les carrés
    for (int i = 0; i < taille; i++) {
        cout << "Le carré de " << tableau1[i] << " est " << tableau2[i]
<< endl;
    }

    // Étape 6 : Libération de la mémoire
    delete[] tableau1; // Libérer le premier tableau
    delete[] tableau2; // Libérer le second tableau

    return 0;
}

```

Exercice 5

Ecrire un programme C++ qui :
1. déclare un entier a;
2. déclare une référence vers cet entier ref_a;
3. déclare un pointeur vers cet entier p_a;
4. affiche les variables, leurs adresses, la valeur pointée.

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

int main() {
    // Étape 1 : Déclarer un entier a
    int a = 1;

    // Étape 2 : Déclarer une référence vers cet entier ref_a
    int& ref_a = a;

    // Étape 3 : Déclarer un pointeur vers cet entier p_a
    int* p_a = &a; // p_a pointe vers l'adresse de a

    // Étape 4 : Afficher les variables, leurs adresses et la valeur poin
tée
    cout << "Valeur de a: " << a << endl;
    cout << "Adresse de a: " << &a << endl;

    cout << "Valeur de ref_a: " << ref_a << endl;
 // Affiche la valeur pointée par ref_a
    cout << "Adresse de ref_a: " << &ref_a << endl;
 // Adresse de ref_a est la même que celle de a

    cout << "Valeur de p_a: " << *p_a << endl;
 // Affiche la valeur pointée par p_a
    cout << "Adresse de p_a: " << p_a << endl;
 // Affiche l'adresse stockée dans p_a (adresse de a)

    return 0;
}

```

Exercice 6

Écrire une fonction nommée **incrementer**() permettant d'incrémenter la valeur d'une variable passée en paramètre et une fonction nommée **permuter**() permettant d'échanger les contenus de 2 variables de type int fournies en argument :
1. en transmettant l'adresse des variables concernées (seule méthode utilisable en C) ;
2. en utilisant la transmission par référence.
Dans les deux cas, écrire un programme (**main**) qui teste les deux fonctions.

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

// Fonction pour incrémenter une variable (par adresse)
void incrementer(int* var) {
    (*var)++; // Incrémente la valeur pointée par var
}

// Fonction pour permuter deux variables (par adresse)
void permuter(int* a, int* b) {
    int temp = *a; // Stocke la valeur de a dans une variable temporaire
    *a = *b;       // Affecte la valeur de b à a
    *b = temp;     // Affecte la valeur temporaire à b
}

// Fonction pour permuter deux variables (par référence)
void permuterRef(int& a, int& b) {
    int temp = a; // Stocke la valeur de a dans une variable temporaire
    a = b;        // Affecte la valeur de b à a
    b = temp;     // Affecte la valeur temporaire à b
}

int main() {
    // Test de la fonction incrementer
    int x = 1;
    cout << "Avant incrementation : x = " << x << endl;
    incrementer(&x); // On passe l'adresse de x
    cout << "Apres incrementation : x = " << x << endl;

    // Test de la fonction permuter (par adresse)
    int a = 2, b = 3;
    cout << "Avant permutation : a = " << a << ", b = " << b << endl;
    permuter(&a, &b); // On passe les adresses de a et
    cout << "Apres pebmutation : a = " << a << ", b = " << b << endl;

    // Test de la fonction permuter (par référence)
    int c = 4, d = 5;
    cout << "Avant permutation par reference : c = " << c << ", d = " << d << endl;
    permuterRef(c, d); // On passe c et d par référence
    cout << "Apres permutation par reference : c = " << c << ", d = " << d << endl;

    return 0;
}
```

Exercice 7

*Écrire une fonction **récursive** en **C++** qui affiche **toutes les permutations possibles** d'une chaîne de caractères donnée. Aucune bibliothèque spéciale ne doit être utilisée !!*

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

// Fonction récursive pour générer toutes les permutations
void permuter(string &str, int debut, int fin) {
    // Cas de base : une permutation complète
    if (debut == fin) {
        cout << str << endl;
        return;
    }

    // Pour chaque position possible
    for (int i = debut; i <= fin; i++) {
        // Échanger le caractère à la position 'debut' avec celui à la position 'i'
        char temp = str[debut];
        str[debut] = str[i];
        str[i] = temp;

        // Appel récursif pour le reste de la chaîne
        permuter(str, debut + 1, fin);

        // Annuler l'échange (backtracking)
        temp = str[debut];
        str[debut] = str[i];
        str[i] = temp;
    }
}

int main() {
    string chaine;
    cout << "Entrez une chaine de caracteres : ";
    cin >> chaine;

    cout << "Les permutations possibles sont :" << endl;
    permuter(chaine, 0, chaine.length() - 1);

    return 0;
}
```

Exercice 8

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

class Voiture {
private:
    string marque;
    string modele;
    int annee;
    float kilometrage;
    float vitesse;

public:
    // Constructeur par défaut
    Voiture() : marque("Inconnue"), modele("Inconnu"), annee(0), kilometrage(0.0), vitesse(0.0) {}

    // Constructeur avec paramètres
    Voiture(string m, string mod, int a, float km, float v)
        : marque(m), modele(mod), annee(a), kilometrage(km), vitesse(v) {}

    // Méthode pour accélérer
    void accelerer(float valeur) {
        vitesse += valeur;
        cout << "La voiture accelere de " << valeur << " km/h. Vitesse actuelle: " << vitesse << " km/h." << endl;
    }

    // Méthode pour freiner
    void freiner(float valeur) {
        vitesse -= valeur;
        if (vitesse < 0) vitesse = 0;
        cout << "La voiture freine de " << valeur << " km/h. Vitesse actuelle: " << vitesse << " km/h." << endl;
    }

    // Méthode pour avancer
    void avancer(float distance) {
        kilometrage += distance;
        cout << "La voiture avance de " << distance << " km. Kilometrage: " << kilometrage << " km." << endl;
    }

    // Méthode pour afficher les informations
    void afficherInfo() const {
        cout << "\n--- Informations Voiture ---" << endl;
        cout << "Marque : " << marque << endl;
        cout << "Modele : " << modele << endl;
        cout << "Annee : " << annee << endl;
        cout << "Kilometrage : " << kilometrage << " km" << endl;
        cout << "Vitesse : " << vitesse << " km/h" << endl;
    }

    // Destructeur
    ~Voiture() {
        cout << "Voiture " << marque << " " << modele << " detruite." << endl;
    }
};

// --- Programme principal ---
int main() {
    Voiture v1("Toyota", "Corolla", 2020, 35000, 0);
    v1.afficherInfo();
    v1.accelerer(50);
    v1.avancer(120);
    v1.freiner(30);
    v1.afficherInfo();

    return 0;
}
```

Exercice 9

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

class Vecteur3D {
private:
    float x, y, z;

public:
    // Constructeur avec valeurs par défaut
    Vecteur3D(float a = 0, float b = 0, float c = 0) : x(a), y(b), z(c) {}

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

    // Somme de deux vecteurs (par valeur)
    Vecteur3D somme(const Vecteur3D& v) const {
        return Vecteur3D(x + v.x, y + v.y, z + v.z);
    }

    // Produit scalaire
    float produitScalaire(const Vecteur3D& v) const {
        return x * v.x + y * v.y + z * v.z;
    }

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

    // Norme du vecteur
    float norme() const {
        return sqrt(x*x + y*y + z*z);
    }

    // normax : renvoie par valeur
    Vecteur3D normax_val(const Vecteur3D& v) const {
        return (this->norme() >= v.norme()) ? *this : v;
    }

    // normax : renvoie par adresse
    const Vecteur3D* normax_adr(const Vecteur3D* v) const {
        return (this->norme() >= v->norme()) ? this : v;
    }

    // normax : renvoie par référence
    const Vecteur3D& normax_ref(const Vecteur3D& v) const {
        return (this->norme() >= v.norme()) ? *this : v;
    }
};

// --- Programme principal ---
int main() {
    Vecteur3D v1(3, 4, 5);
    Vecteur3D v2(1, 2, 2);

    cout << "Vecteur 1 : "; v1.afficher();
    cout << "Vecteur 2 : "; v2.afficher();

    cout << "Somme : "; v1.somme(v2).afficher();
    cout << "Produit scalaire : " << v1.produitScalaire(v2) << endl;
    cout << "Norme de v1 : " << v1.norme() << endl;

    cout << "V1 et V2 coincident ? " << (v1.coincide(v2) ? "Oui" : "Non") << endl;

    cout << "Vecteur de plus grande norme (par valeur) : ";
    v1.normax_val(v2).afficher();

    cout << "Vecteur de plus grande norme (par adresse) : ";
    v1.normax_adr(&v2)->afficher();

    cout << "Vecteur de plus grande norme (par référence) : ";
    v1.normax_ref(v2).afficher();

    return 0;
}
```

Exercice 10

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

class Complexe {
private:
    float re, im;

public:
    // Constructeur
    Complexe(float r = 0, float i = 0) : re(r), im(i) {}

    // Affichage
    void afficher() const {
        cout << "(" << re << ", " << im << "i)" << endl;
    }

    // Addition
    Complexe addition(const Complexe& c) const {
        return Complexe(re + c.re, im + c.im);
    }

    // Soustraction
    Complexe soustraction(const Complexe& c) const {
        return Complexe(re - c.re, im - c.im);
    }

    // Multiplication
    Complexe multiplication(const Complexe& c) const {
        return Complexe(re * c.re - im * c.im, re * c.im + im * c.re);
    }

    // Division
    Complexe division(const Complexe& c) const {
        float denom = c.re * c.re + c.im * c.im;
        return Complexe((re * c.re + im * c.im) / denom,
                        (im * c.re - re * c.im) / denom);
    }

    // Égalité
    bool egal(const Complexe& c) const {
        return (re == c.re && im == c.im);
    }
};

// --- Programme principal ---
int main() {
    Complexe c1, c2;
    cout << "Entrez la partie reelle et imaginaire du premier complexe : ";
    float r1, i1; cin >> r1 >> i1;
    c1 = Complexe(r1, i1);

    cout << "Entrez la partie reelle et imaginaire du second complexe : ";
    float r2, i2; cin >> r2 >> i2;
    c2 = Complexe(r2, i2);

    int choix;
    do {
        cout << "\n=== MENU ===\n";
        cout << "1. Addition\n2. Soustraction\n3. Multiplication\n4. Division\n5. Egalite\n
0. Quitter\n";
        cout << "Choix : ";
        cin >> choix;

        switch (choix) {
            case 1: cout << "Résultat : "; c1.addition(c2).afficher(); break;
            case 2: cout << "Résultat : "; c1.soustraction(c2).afficher(); break;
            case 3: cout << "Résultat : "; c1.multiplication(c2).afficher(); break;
            case 4: cout << "Résultat : "; c1.division(c2).afficher(); break;
            case 5: cout << (c1.egal(c2) ? "Les complexes sont égaux" :
"Les complexes sont différents") << endl; break;
        }
    } while (choix != 0);

    return 0;
}

```