ATELIER 2

EXERCICE 1 :

```
#include <iostream>
using namespace std ;
int nombre = 0; // Variable globale pour le compte soit garder dans
chaque appellation

void Appels() {
    nombre++; // Incrémente le compteur
    cout << "appel numéro " << nombre << endl; // Affiche le numéro
d'appel
}
int main() {
    Appels(); // appel numéro 1
    Appels(); // appel numéro 2
    Appels(); // appel numéro 3
    return 0;
}
```

EXERCICE 2 :

```

#include <iostream>

using namespace std;
// fonction à un argument entier et une valeur de retour entière
permettant de préciser si l'argument reçu est multiple de 2
int MULTIPLE2 (int x) {
      return ( x%2==0 );
      }

      // fonction à un argument entier et une valeur de retour entière
permettant de préciser si l'argument reçu est multiple de 3
int MULTIPLE3 (int x) {
      return (x%3==0);
      }
 int main ()  {
  int nombre ;
  cout << "donner un nombre entier " ;
  cin >> nombre ;
 // Vérification si le nombre est pair
```

```
    if (MULTIPLE2(nombre)) {
        cout << "Il est pair" << endl;
       } else
            cout << "Il n'est pas pair" << endl;

            // Vérification si le nombre est multiple de 3

 if (MULTIPLE3(nombre)) {
        cout << "Il est multiple de 3" << endl ;
 } else
  cout << "Il n'est pas multiple de 3" << endl ;

    // Vérification si le nombre est divisible par 6 (multiple de 2 et
de 3)

    if (MULTIPLE2(nombre) && MULTIPLE3(nombre)) {
        cout << "Il est divisible par 6" << endl;
    }
      else  cout << "Il n'est pas divisible par 6" << endl;


    return 0;
}
```

EXERCICE 3 :

premier méthodes : (tableau)

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

int main() {
    int T[10];
    int maxx , minn;

    // Lecture des 10 nombres dans le tableau
    cout << "Entrez 10 nombres entiers :" << endl;
    for (int i = 0; i <= 10; i++) {
        cin >> T[i];
    }

    // Initialisation du plus grand et plus petit avec le premier
élément du tableau
```

```
    maxx = T[0];
    minn = T[0];

    // Recherche du plus grand et du plus petit dans le tableau
    for (int i = 1; i < 10; i++) {
        if (T[i] > maxx) {
            maxx = T[i];
        }
        if (T[i] < minn) {
            minn= T[i];
        }
    }

    // Affichage des résultats
    cout << "Le plus grand nombre est : " << maxx << endl;
    cout << "Le plus petit nombre est : " << minn << endl;

    return 0;
}
```

deuxième methodes : (pointeur)

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

int main() {
    int T[10];
    int *p = T;  // Pointeur vers le début du tableau
    int maxx , minn;

    // Lecture des 10 nombres dans le tableau en utilisant un pointeur
    cout << "Entrez 10 nombres entiers :" << endl;
    for (int i = 0; i < 10; i++) {
        cin >> *(p + i);  // Utilisation du pointeur pour remplir le
tableau
    }

    // Initialisation du plus grand et plus petit avec le premier
élément
    maxx = *p;
    minn = *p;
```

```
    // Incrémentation du pointeur pour parcourir le tableau
    for (int *p1 = p + 1; p1 < p + 10; p1++) {
        if (*p1 > maxx) {
           maxx= *p1;
        }
        if (*p1 <minn) {
           maxx = *p1;
        }
    }

    // Affichage des résultats
    cout << "Le plus grand nombre est : " << maxx << endl;
    cout << "Le plus petit nombre est : " << minn << endl;

    return 0;
}
```

EXERCICE 7:

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

int main() {
    int T[10];

    // Lecture des 10 entiers à partir de l'utilisateur
    cout << "Entrez 10 nombres entiers :" << endl;
    for (int i = 0; i <= 10; i++) {
        cin >> T[i];
    }

    // Tri à bulles
    int ctr= 0 ;  // Réinitialisation du nombre d'échanges
    bool echange ;
    do {
            echange=false ;
        for (int i = 0; i < 9 ; i++) {
            // Comparaison des éléments adjacents
            if (T[i] > T[i + 1]) {
```

```
                // Si les éléments ne sont pas dans l'ordre, on les
échange
                int temp = T[i];
                T[i] = T[i + 1];
                T[i + 1] = temp;
                 echange=true ;
            }
        }
        ctr++;  // Un échange a été effectue
    } while (echange);  // On continue tant qu'il y a eu un échange

    // Affichage du tableau trié
    cout << "Le tableau trié par ordre croissant est :" << endl;
    for (int i = 0; i < 10; i++) {
        cout << T[i] << " ";
    }
    cout << "et le nombre des echanges est:" << ctr << endl;


    return 0;
}
```

EXERCICE 4 :

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

int main() {
    int taille;

    // Demander la taille du tableau
    cout << "Entrez la taille du tableau : ";
    cin >> taille;

    // Allocation dynamique d'un tableau d'entiers
    int*T = new int[taille];

    // Lecture des nombres entiers pour remplir le tableau
    cout << "Entrez " << taille << " nombres entiers :" << endl;
    for (int i = 0; i < taille; i++) {
```

```
        cin >> T[i];
    }

    // Allocation dynamique d'un nouveau tableau pour stocker les
carrés
    int* tableauCarres = new int[taille];

    // Remplir le second tableau avec les carrés des éléments du
premier tableau
    for (int i = 0; i < taille; i++) {
        tableauCarres[i] = T[i] * T[i];
    }

    // Suppression du premier tableau
    delete[] T;

    // Affichage des valeurs du second tableau (les carrés)
    cout << "Les carrés des nombres sont :" << endl;
    for (int i = 0; i < taille; i++) {
        cout << tableauCarres[i] << " ";
    }
    cout << endl;

    // Suppression du second tableau
    delete[] tableauCarres;

    return 0;
}
```

EXERCICE 5

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

int main() {
    // 1. Déclaration d'un entier a
    int a;
    cout << "entrez la Valeur de a: " << endl;
    cin >> a ;

    // 2. Déclaration d'une référence vers cet entier ref_a
```

```
    int& ref_a = a;

    // 3. Déclaration d'un pointeur vers cet entier p_a
    int* p_a = &a;

    // 4. Affichage des variables, leurs adresses, et la valeur pointée
    cout << "Valeur de a est : " << a << endl;
    cout << "Adresse de a est : " << &a << endl;

    cout << "Valeur de ref_a : " << ref_a << endl;
    cout << "Adresse de ref_a: " << &ref_a << endl;

    cout << "Valeur du pointeur p_a (adresse de a): " << p_a << endl;
    cout << "Valeur pointée par p_a: " << *p_a << endl;

    return 0;
}
```

EXERCICE 6 :

\\ 1. en transmettant l'adresse des variables concernées (seule méthode utilisable en C) ;

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

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

// Fonction pour permuter deux valeurs passées par adresse
void permuter(int* x, int* y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}

int main() {
    int a = 1;
    int b = 2;
```

```
    cout << "Avant incrémentation, a = " << a << endl;
    incrementer(&a); // Passer l'adresse de a
    cout << "Après incrémentation, a = " << a << endl;

    cout << "Avant permutation, a = " << a << ", b = " << b << endl;
    permuter(&a, &b); // Passer les adresses de a et b
    cout << "Après permutation, a = " << a << ", b = " << b << endl;

    return 0;
}
```

\\ 2. en utilisant la transmission par référence.

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

// Fonction pour incrémenter la valeur passée par référence
void incrementer(int & reference) {
    reference++; // Incrémente la valeur directement en utilisant
reference
}

// Fonction pour permuter deux valeurs passées par référence
void permuter(int& ref1, int& ref2) {
    int temp =ref1;
    ref1 = ref2;
   ref2= temp;
}

int main() {
    int a = 1;
    int b = 2;

    cout << "Avant incrémentation, a = " << a << endl;
    incrementer(a); // Passer la variable directement
    cout << "Après incrémentation, a = " << a << endl;

    cout << "Avant permutation, a = " << a << ", b = " << b << endl;
    permuter(a, b); // Passer les variables directement
    cout << "Après permutation, a = " << a << ", b = " << b << endl;
```

```
    return 0;
}
```