الكلية متعددة التخصصات – ورزازات
ⵜⴰⵖⵉⵎⵉⵜ ⵜⴰⵙⴳⴳⵉⵔⵜ – ⵡⴰⵔⵣⴰⵣⴰⵜ
FACULTÉ POLYDISCIPLINAIRE DE OUARZAZATE
UNIVERSITÉ IBN ZOHR - AGADIR

Année universitaire 2020/2021

Travaux dirigés et travaux pratiques

<u>Série N° : 4</u>

Filière : ***SMI***

Module : *Programmation II*

Semestre : ***4***

<u>Exercice N° :1 les Tableaux</u>

Soit n le nombre d'étudiants. Ecrire un programme en langage C et:

- Saisir les N=40 notes et les conserver ;
- Calculer la moyenne de ces n notes ;
- Déterminer combien, parmi ces n notes, sont supérieures à la moyenne obtenue.

<u>Exercice N° :2 les Tableaux multidimensionnels (Matrice)</u>

- Ecrire un programme en C qui :
- **définit** deux tableaux des réels m [M][N] et n[M][N] de type **matrice3x3**.
- Calcule une matrice *C* ou C est La somme de 2 matrices *A* et *B,* chaque élément *cij* est la somme des éléments correspondants *aij* et *bij*
- Calcule La transposée A^t de la matrice *A.*
- Calcule La matrice C résultant du produit de 2 matrices A et B (le nombre de colonnes de A doit être égal au nombre de lignes de B, on le note n) est obtenu par la formule :

$$c_{ij} = \sum_{k=0}^{n} a_{ik} b_{kj}$$

- Calcule L'inverse de la matrice m.

<u>Exercice N° :3 les structures</u>

Ecrire un programme qui permet de créer une liste chainée dont les éléments sont des **structures** comprenant un **nom**, un **prénom** et un **numéro de téléphone** et un pointeur **ptr** permettant d'accéder à l'élément suivant. Le répertoire lui-même est un pointeur, de nom **début**, permettant d'accéder au premier élément.

N.B : Pour simplifier, la liste est crée en partant du dernier élément, et non pas du premier. On affiche, dans une seconde étape, les valeurs de la liste. Celle-ci sera affichée dans l'ordre inverse de l'ordre d'entrée.

Exercice N° :4 Calcul de factorielle

- Ecrire un programme en C qui demande à l'utilisateur un nombre entre 0 et 7 jusqu'à ce qu'il soit effectivement entre 0 et 7, puis affiche la factorielle du nombre saisi.
- Ecrire un programme en C qui calcul de la factorielle d'un nombre avec la méthode itérative et la méthode récursive.

***Exercice N° :5 : le Tri, Échange de deux variables,* Tri à bulles, Tri par sélection, Tri par insertion**

1. Ecrire un programme en C qui trie et affiche un tableau de valeurs entières. voici le tableau à trier :
   a. tab [MAX] = {3,10,-5,8,2 }.
2. Ecrire un programme en C qui fait l'échange deux valeurs réelles saisies au clavier.

Utiliser la fonction **echanger (float * ad_f1, float * ad_f2)** avec:

- ad_f1 est un pointeur (ad comme adresse) sur un réel (appelé f1)
- ad_f1 est la valeur réelle pointée :float *ad_f1<=>float f1.

3. Tri à bulles

L'algorithme de tri à bulles parcourt la liste, et compare les couples d'éléments successifs. Lorsque deux éléments successifs ne sont pas dans l'ordre croissant, ils sont échangés. Après chaque parcours complet de la liste, l'algorithme recommence l'opération. Lorsqu'aucun échange n'a lieu pendant un parcours, cela signifie que la liste est triée : l'algorithme peut s'arrêter.

1) Triez le tableau suivant et Donnez Donnez l'algorithme et programme en C, l'implémentation de la fonction tri_bulles(tableau) qui prend en paramètre un tableau et le tri à l'aide de l'algorithme de tri à bulles.

3	5	2	0	1	4

4. Tri par sélection

Il consiste en la rechercher du plus grand élément (ou le plus peit) dans le tableau que l'on va replacer à sa position finale c'est-à-dire en dernière position (ou en première), puis on recherche le second plus grand élément (ou le second plus petit) que l'on va replacer également à sa position finale c'est-à-dire en avant-dernière position (ou en seconde), etc., jusqu'à ce que le tableau soit entièrement trié.

2) Triez le tableau ci-dessus et Donnez l'algorithme et programme en C, l'implémentation de la fonction tri_selection(tableau).

5. Tri par insertion

Le tri par insertion consiste à trier la liste au fur et à mesure que l'on ajoute un élément. Pour ce faire, on parcourt la liste en selectionnant les éléments dans l'ordre. Pour chaque élément selectionner, on le compare avec les éléments précédents qui ont déjà été ordonné jusqu'à trouver sa place. Il ne

reste plus qu'à décaler les éléments du tableau pour insérer l'élément considéré à sa place dans la partie déjà triée.
3) Triez le tableau ci-dessus et Donnez l'algorithme et programme en C, l'implémentation de la fonction tri_insertion (tableau).

Corrections :1

Programme.- Ceci nous conduit au programme suivant en supposant qu'il y ait 59 ´el`eves :

```
/* Programme note_1.c */
#include <stdio.h>
void main(void)
{
int i, s;
float m;
float note[59];
/* Saisie des notes */
for (i=0 ; i < 59 ; i++)
{
/* TABLEAUX `A UNE DIMENSION */
printf("\nQuelle est la note numero %d : ",i+1);
scanf("%f", &note[i]);
}
/* Calcul de la moyenne */
m = 0;
for (i = 0 ; i < 59 ; i++) m = m + note[i];
m = m/59;
printf("\nLa moyenne est de : %4.2f", m);
/* Calcul du nombre de notes superieures a la
* moyenne */
s = 0;
for (i = 0 ; i < 59 ; i++)
if (note[i] >= m) s = s+1;
printf("\nLe nombre de notes superieures a la");
printf("\nmoyenne de la classe est de : %d", s);
}
```

Corrections :2

```
typedef float Matrice3x3[3][3];
Matrice3x3 m1,m2;
Matrice3x3 identite= { {1.0, 0.0, 0.0},
                       {0.0, 1.0, 0.0},
                      {0.0, 0.0, 1.0}};

void initMatrice(Matrice3x3 m)
{
int i,j;
for(i=0;i<3;i++)
for(j=0;j<3;j++) {
printf("entrez l'élément m[%d][%d] : ",i,j);
scanf("%d", &m[i][j]);
}
}
```

/*De la même manière, la fonction qui affiche à l'écran une matrice 3x3 est :*/

```
void afficheMatrice(Matrice3x3 m)
{
int i,j;
for(i=0;i<3;i++) {
        for(j=0;j<3;j++)

printf("%6.2f ",m[i][j]);
printf("\n");
}
        }
```

/*La somme de 2 matrices *A* et *B* est une matrice *C/**

```
void sommeMatrices(Matrice3x3 A, Matrice3x3 B, Matrice3x3 C)
{
int i,j;
for(i=0;i<3;i++)
for(j=0;j<3;j++)
C[i][j]=A[i][j]+B[i][j];
        }
```

/*La transposée A_t d'une matrice *A* est*/

```
void transposeMatrice(Matrice3x3 A, Matrice3x3 At)
{
int i,j;
for(i=0;i<3;i++)
for(j=0;j<3;j++)
At[i][j]=A[j][i];
}
void transposeMatrice(Matrice3x3 A, Matrice3x3 At)
{
int i,j;
        for(i=0;i<3;i++)

for(j=0;j<3;j++)
At[i][j]=A[i][j];
for(i=0;i<3;i++)
for(j=0;j<i;j++)
permuterEntiers(&At[i][j], &At[j][i]);
        }
```

/*La matrice *C* résultant du produit de 2 matrices *A* et *B/**

```
void produitMatrices(Matrice3x3 A, Matrice3x3 B, Matrice3x3 C)
{
Matrice3x3 Ctmp;
int i,j,k;
for(i=0;i<3;i++)
for(j=0;j<3;j++)
{
Ctmp[i][j]=0.0;
for (k=0;k<3;k++)
Ctmp[i][j] += A[i][k]*B[k][j];
}
```

```
        copieMatrice(Ctmp,C);
```

/* ou l'inversion d'une matrice 3x3 avec calcul du déterminant :*/

```
/* Inversion de matrice 3x3. Résultat : succès ou échec */
int inverseMatrice(Matrice3x3 m, Matrice3x3 m2)
{
float determinant, unSurDeterminant;
float cf[3][3];
int i, j, res=0;
determinant = m[0][0]*(m[1][1]*m[2][2] - m[1][2]*m[2][1]) -
m[0][1]*(m[1][0]*m[2][2] - m[1][2]*m[2][0]) + m[0][2]*(m[1][0]*m[2][1] - m[1][1]*m[2][0]);
if (determinant != 0.0) {
res = 1;
unSurDeterminant = 1.0 / determinant;
cf[0][0] = m[1][1]*m[2][2] - m[1][2]*m[2][1];
cf[0][1] = -(m[1][0]*m[2][2] - m[1][2]*m[2][0]);
cf[0][2] = m[1][0]*m[2][1] - m[1][1]*m[2][0];
cf[1][0] = -(m[0][1]*m[2][2] - m[0][2]*m[2][1]);
cf[1][1] = m[0][0]*m[2][2] - m[0][2]*m[2][0];
cf[1][2] = -(m[0][0]*m[2][1] - m[0][1]*m[2][0]);
cf[2][0] = m[0][1]*m[1][2] - m[0][2]*m[1][1];
cf[2][1] = -(m[0][0]*m[1][2] - m[0][2]*m[1][0]);
cf[2][2] = m[0][0]*m[1][1] - m[0][1]*m[1][0];
for (i = 0; i < 3; i++)
for (j = 0; j < 3; j++)
m2[i][j] = unSurDeterminant*cf[j][i];
}
        return res;
```

Correction:3

```
/* liste_1.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void main(void)
{
struct donnee
{
char nom[30];
char tel[20];
struct donnee *suivant;
};
struct donnee item, *debut, *ptr;
char name[30], telephone[20];
/* Initialisation de la liste */
debut = NULL;
do
{
printf("\nNom : ");
scanf("%s",name);
if (name[0] != '#')
{
printf("Telephone : ");
```

```
scanf("%s",telephone);
ptr = (struct donnee *) malloc(sizeof(item));
strcpy(item.nom, name);
strcpy(item.tel,telephone);
item.suivant = debut;
*ptr = item;
debut = ptr;
}
}
while (name[0] != ’#’);
/*STRUCTURES AUTO-R´EF´ERENTES */
/* Affichage de la liste */
printf("\nLa liste dans l\’ordre inverse est :\n");
while (debut != NULL)
{
printf("%s : %s\n",debut->nom, debut->tel);
debut = debut->suivant;
}
/* Liberation des pointeurs */
while (debut != NULL)
{
ptr = debut;
debut = debut->suivant;
free(ptr);
}
}
```

Correction:4

```
#include <stdio.h>
int main (){
char n; /*le type « char » est un 'petit' entier de 0 à 255*/
int i , fact = 1; /*initialisation possible lors de la déclaration*/
/*Lecture au clavier d’un décimal mis dans n (à son adresse)*/
printf("entrez 0<=n<=7 : "); scanf("%d",&n);
/*While effectue le bloc {} tant que la condition entre () est vraie*/
while((n<=0) | | (n>7)) { /*| | est le « ou » logique*/
if (n<=0) printf("0>n! "); else printf("n>7! ");
printf("entrez 0<=n<=7 : "); scanf("%d",&n);
}
/*Pour i de 2 « i=2 » à n « i<=n » par pas de 1 « i++ »*/
for(i=2;i<=n;i++) fact*=i; /*fact*=i <=> fact=fact*i;*/
/*Le 1er %d affiche n en décimal, le 2ème fact. Ex : écrit 5!=120 si n=5.*/
printf("%d! = %d\n",n,fact);
return 0;
}
```

Correction:5

1)

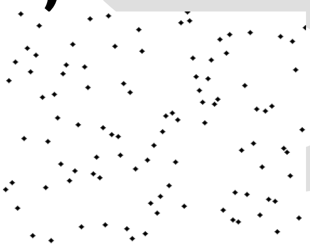
```
/*Ce programme comporte plusieurs fonctions*/
#include <stdio.h>
enum {MAX = 5}; /*MAX est une constante entière globale*/
int tab[MAX] = {3,10,-5,8,2 } ; /*le tableau tab[0..4] est global*/
/* prototype des fonctions : type rendu nom ( paramètres ) */
/* trie le tableau global tab de taille MAX (pas de paramètres) */
void trier(void);
```

```
/*affiche le tableau t de taille éléments */
void afficher(int taille,int t[]);
void main(void) { /* main a pour rôle d'appeler trier et afficher */
trier();
afficher(MAX,tab);
}
void trier(void) {
int i, j, cle;
for (i = 1; i < MAX; i++) {
cle = tab[i];
for (j = i-1; (j >= 0) && (cle<tab[j]); j--) tab[j+1] = tab[j];
tab[j+1] = cle;
}
}
void afficher(int taille,int t[]) {
int i;
printf("\nvoici le tableau trié\n");
for(i = 0; i < taille; i++) printf("%5d",t[i]);
printf("\n");
}
```

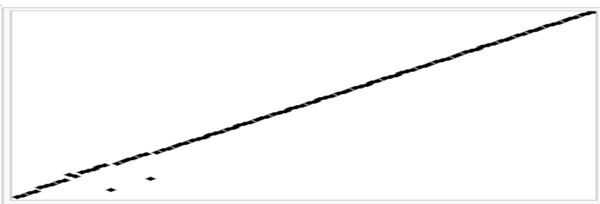
- **2)/* echange de deux variables*/**

```
#include <stdio.h>
void echanger(float * ad_f1, float * ad_f2) {
/* ad_f1 est un pointeur (ad comme adresse) sur un réel (appelé f1)
*ad_f1 est la valeur réelle pointée :float *ad_f1<=>float f1 */
float tampon = *ad_f1;
*ad_f1 = *ad_f2;
*ad_f2 = tampon;
}
void main() {
float x, y;
printf("Donnez deux réels : \n");
scanf("%f%f", &x, &y);
printf("Avant , x=%f ; y=%f\n", x, y);
echanger(&x, &y); /* A l'appel : ad_f1 reçoit &x, ad_f2 reçoit &y */
printf("Après , x=%f ; y=%f\n", x, y);
}
```

3) Tris a bulles



===>



Exemple du tri à bulles utilisant une liste de nombres aléatoires

L'algorithme :

```
procédure tri_bulle(tableau T, entier n)
    répéter
        aucun_échange = vrai
        pour j de 0 à n - 2
```

```
            si T[j] > T[j + 1], alors
                échanger T[j] et T[j + 1]
                aucun_échange = faux
    tant que aucun_échange = faux
```

Exemple étape par étape[modifier]

Prenons la liste de chiffres « 5 1 4 2 8 » et trions-la de manière croissante en utilisant l'algorithme de tri à bulles. Pour chaque étape, les éléments comparés sont écrits en gras.

Première étape:
(**5 1** 4 2 8) → (**1 5** 4 2 8) Les éléments 5 et 1 sont comparés, et comme 5 > 1, l'algorithme les intervertit.
(1 **5 4** 2 8) → (1 **4 5** 2 8) Interversion car 5 > 4.
(1 4 **5 2** 8) → (1 4 **2 5** 8) Interversion car 5 > 2.
(1 4 2 **5 8**) → (1 4 2 **5 8**) Comme 5 < 8, les éléments ne sont pas échangés.
Deuxième étape:
(**1 4** 2 5 8) → (**1 4** 2 5 8) Même principe qu'à l'étape 1.
(1 **4 2** 5 8) → (1 **2 4** 5 8)
(1 2 **4 5** 8) → (1 2 **4 5** 8)
(1 2 4 **5 8**) → (1 2 4 **5 8**)
À ce stade, la liste est triée, mais pour le détecter, l'algorithme doit effectuer un dernier parcours.
Troisième étape:
(**1 2** 4 5 8) → (**1 2** 4 5 8)
(1 **2 4** 5 8) → (1 **2 4** 5 8)
(1 2 **4 5** 8) → (1 2 **4 5** 8)
(1 2 4 **5 8**) → (1 2 4 **5 8**)
Comme la liste est triée, aucune interversion n'a lieu à cette étape, ce qui provoque l'arrêt de l'algorithme

Programme en C

```
#define TRUE 1;
#define FALSE 0;

void tri_a_bulle(int t[], int const n)
{
        int j   = 0; /* Variable de boucle */
        int tmp = 0; /* Variable de stockage temporaire */

        /* Booléen marquant l'arrêt du tri si le tableau est ordonné */
        int en_desordre = TRUE;
        /* Boucle de répétition du tri et le test qui arrête le tri dès
que le tableau est ordonné(en_desordre=FALSE) */
        while (en_desordre)
```

```
        {
    /* Supposons le tableau ordonné */
                en_desordre = FALSE;
                /* Vérification des éléments des places j et j+1 */
                for (j = 0; j < n-1; j++)
                {
                        /* Si les 2 éléments sont mal triés */
                        if(t[j] > t[j+1])
                        {
                                /* Inversion des 2 éléments */
                                tmp = t[j+1];
                                t[j+1] = t[j];
                                t[j] = tmp;
      /* Le tableau n'est toujours pas trié */
                                en_desordre = TRUE;
                        }
                }
        }
}
```

3) Tris par sélection

L'algorithme :

```
procédure tri_selection(tableau t, entier n)
      pour i de 1 à n - 1
          min ← i
          pour j de i + 1 à n
              si t[j] < t[min], alors min ← j
          si min ≠ i, alors échanger t[i] et t[min]
```

Programme en C

```
#define MAX 50

typedef int tab_entiers[MAX];

void selection(tab_entiers t)
{
     int i, mini, j , x;
     for(i = 0 ; i < MAX - 1 ; i++)
     {
         mini = i;
```

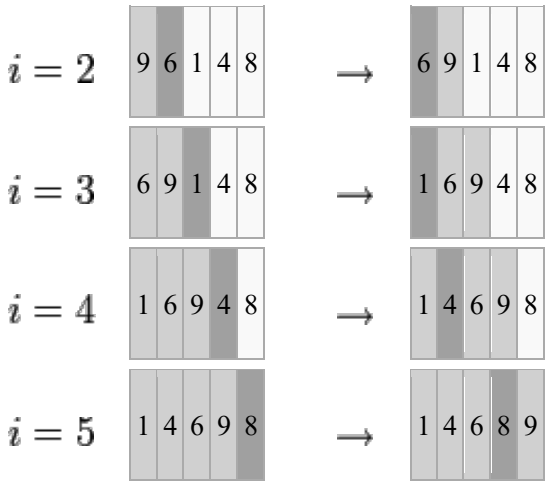

```
        for(j = i+1 ; j < MAX ; j++)
             if(t[j] < t[mini])
                 mini = j;
        if(mini != i)
        {
            x = t[i];
            t[i] = t[mini];
            t[mini] = x;
        }
    }
}
```

4) Tris par insertion

L'algorithme :

```
procédure tri_insertion(tableau T, entier n)
      pour i de 2 à n
          x ← T[i]
          j ← i
          tant que j > 1 et T[j - 1] > x
              T[j] ← T[j - 1]
              j ← j - 1
          fin tant que
          T[j] ← x
     fin pour
  fin procédure
```

Voici les étapes de l'exécution du tri par insertion sur le tableau $T = [9, 6, 1, 4, 8]$. Le tableau est représenté au début et à la fin de chaque itération.



le programme en C

```c
void tri_insertion(double* t, int n)
{
    int i, j;
    double elementInsere;

    for (i = 1; i < n; i++) {
        /* Stockage de la valeur en i */
        elementInsere = t[i];
        /* Décale les éléments situés avant t[i] vers la droite
           jusqu'à trouver la position d'insertion */
        for (j = i; j > 0 && t[j - 1] > elementInsere; j--) {
            t[j] = t[j - 1];
        }
        /* Insertion de la valeur stockée à la place vacante */
        t[j] = elementInsere;
    }
}
```