Année académique **2023-2024**

Parcours : **Licences Fondamentales (LF)**
UE : **INF1220 Structures de Données & Programmation en Langage C**
Enseignant : **Monsieur AKAKPO**
Évaluation : **Corrigé du DST**
Durée : **Deux (2) heures**

***Pensée de début** : « Ce n'est pas parce que les choses sont difficiles que nous n'osons pas, mais c'est parce que nous n'osons pas qu'elles sont difficiles. » SENEQUE*

NB : Traiter les questions dans leur ordre.

Partie 1 : TABLEAUX, POINTEURS ET FONCTIONS *[10 points]*

(Exercice 10.13 du support de FABER, support n°3, énoncé p.186, corrigé p.325)

1) Implémentez la fonction LIRE_TAB en choisissant bien le type des paramètres. *[4,0 pts]*

```
[0.5]void LIRE_TAB (int *TAB, int *N, int NMAX)
{
     /* Variables locales */
     int I;
     /* Saisie de la dimension du tableau */
     do {
[0.5]     printf("Dimension du tableau (max.%d) : ", NMAX);
[0.5]     scanf("%d", N); /* Attention: écrire N et non &N ! */
[1]  }while (*N<0 || *N>NMAX);
     /* Saisie des composantes du tableau */
[0.5]for (I=0; I<*N; I++){
[0.5]     printf("Elément[%d] : ", I+1);
[0.5]     scanf("%d", TAB+I);
     }
}// void LIRE_TAB (int *TAB, int *N, int NMAX)
```

2) Ecrivez la fonction SOMME_TAB qui calcule la somme des N éléments du tableau TAB du type int. N et TAB sont fournis comme paramètres ; la somme est retournée comme résultat du type long. Utilisez le formalisme POINTEUR. *[4,0 pts]*

```
[0.5]long SOMME_TAB(int TAB[], int N)
{
[0.5]long somme ;
     int cpt ;
[0.5]somme = *TAB ;
[0.5]for(cpt=0 ; cpt < N ; cpt++){
[1]        somme += *(TAB+I);
     }
[1]  return somme;
}// long SOMME_TAB(int TAB[], int N)
```

3) Écrivez les instructions du programme principal qui appellent ces fonctions. *[2,0 pts]*

```
[0.5]     const int nbMax=10;
[0.5]     int TAB[nbMax], N, cpt;
          long somme;

[0.5]     LIRE_TAB(TAB, &N, nbMax);
[0.5]     somme = SOMME_TAB(TAB, N);
```

Partie 2 : STRUCTURES, POINTEURS ET FONCTIONS *[10 points]*

Il y a nécessité d'écrire un programme qui permet d'opérer sur des points. Chaque point contiendra les deux coordonnées d'un point du plan.

4) Définissez la structure **TPoint**. *[2,0 pts]*

```
[0.5]   typedef struct TPoint TPoint ;
[0.5]   struct TPoint{
[0.5]       int abscisse ;
[0.5]       int ordonnee ;
        } ;
```

5) Donnez la définition de la fonction **lirePoint** qui sera appelée pour lire deux points de la façon suivante : *[3,0 pts]*

```
TPoint *p1, *p2 ;
… … …
p1 = lirePoint("\nEntrée des coordonnées du premier point :") ;
p2 = lirePoint("\nEntrée des coordonnées du deuxième point :") ;
```

```
[0.5]TPoint *lirePoint(char message[])
    {
[0.5]   TPoint *point=(TPoint*)malloc(sizeof(TPoint));
[0.5]   printf("%s", message);
        printf("\n\tAbscisse=");
[0.5]   scanf("%f",&point->abscisse);
        printf("\n\tOrdonnée=");
[0.5]   scanf("%d",&point->ordonnee);
[0.5]   return point;
    }
```

6) Donnez la définition de la fonction **calculerDistance** qui calcule et renvoie la distance entre deux points ; voici son prototype : *[4,0 pts]*

float calculerDistance(TPoint *point1, TPoint *point2) ;

La distance entre deux points M(a, b) et N(c, d) est donnée par la formule suivante :

$$d(M,N) = \sqrt{(a-c)^2 + (b-d)^2}$$

```
[1]     float calculerDistance(TPoint *point1, TPoint *point2)
        {
[0.5]       float dist ;
[0.5]       dist=sqrt(
[0.5]           pow(point1->abscisse - point2>abscisse, 2)+
[0.5]       pow(point1->ordonnee - point2>ordonnee, 2));
[1]         return dist ;
        }
```

7) Écrivez les instructions du programme principal qui appellent la fonction **calculerDistance.** *[1,0 pt]*

```
float dist ;
dist = calculerDistance(p1, p2) ;
```

Pensée de fin : ***« Si vous pratiquez régulièrement quelque chose, vous finirez par devenir très bon. L'entraînement mène à l'excellence. »*** Rick WARREN