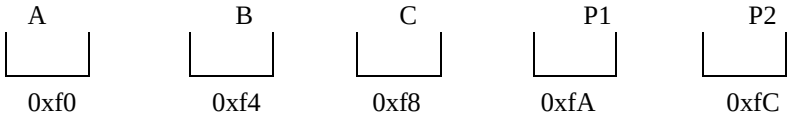
MODULE : Algorithmique et Structures de Données 2 (ASD 2)

SERIE D'EXERCICES SUR LES POINTEURS

Exercice 1 (TD/TP) :

Soient trois entiers A, B et C et deux pointeurs de type entier P1 et P2.



1. Déterminez quelle est la valeur des différents éléments donnés dans le tableau Pour chaque opération.

	A	B	C	&A	&B	&C	P1	P2	*P1	*P2
A←1, B←2, C←3 P1←&A, P2←&B										
P2= &C										
***P1 = *P2**										
(*P2)++										
P1= P2										
P2= &B										
***P2 = *P1 - 2 * *P2**										
(*P2)--										
C = (P2 == &C)										
***P2= *P1 + A**										

2. Confirmez les résultats de ce tableau à l'aide d'un programme C en utilisant une procédure d'affichage avec des variables globales***(les adresses réelles vont se changer bien sûr).***

Exercice 2 (TD):

On considère que les déclarations suivantes ont été faites :
int a;
char tab[10];
Une expression avec pointeurs (resp. sans pointeurs) vous est donnée, vous devez la ré-écrire sans (resp. avec) l'usage explicite des pointeurs.

1. *(&a)
2. *tab3
3. *(tab + 0)
4. (*tab) + 1
5. &(tab[0])
6. &(tab[i])
7. ++tab[i]

Exercice 3 (TD & TP)

Écrire un programme qui lit deux tableaux A et B et leurs dimensions N et M au clavier et qui ajoute les éléments de B à la fin de A. Utiliser le formalisme pointeur à chaque fois que cela est possible.

Exercice 4(TD & TP)

Écrire un programme dans lequel vous déclarerez un tableau statique d'entiers tab avec des valeurs (10 par exemple), dont certaines seront nulles. Le programme doit lire et parcourir le tableau et imprimer les index des éléments nuls du tableau ***(sans utiliser aucune variable de type entier mais des pointeurs)***.

Exercice 5 (TD & TP)

Ecrire un programme qui lit un entier X et un tableau A du type int au clavier et élimine toutes les occurrences de X dans A en tassant les éléments restants. Le programme utilisera un pointeur P1.

Exercice 6 (TD)

Écrire un programme avec les pointeurs qui construit une matrice unitaire avec une dimension N (fixé par l'utilisateur)

Rem : une matrice unitaire est une matrice carrée dont les éléments de la diagonale sont égaux à 1 les autres éléments sont nuls.

Exemple : ( )

Exercice 7 (TP)

Écrire un programme avec les pointeurs qui lit une matrice carrée avec une dimension N (fixé par l'utilisateur) et qui la transforme en matrice diagonale puis il calcul sa trace.

Rem :

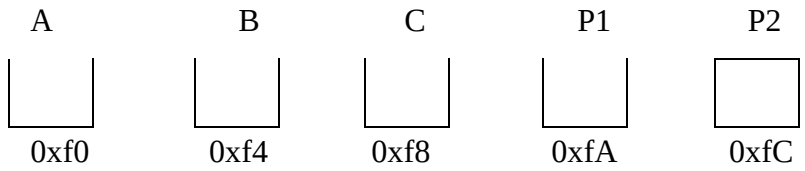
1- *une matrice diagonale* est une matrice carrée dont les coefficients en dehors de la diagonale principale sont nuls

Exemple d'une matrice diagonale : M ( )

2- Pour une matrice carrée M, sa trace est la somme de ses coefficients diagonaux, notée Tr(M). (Pour la matrice précédente : Tr(M)= 12-1+6=17

Corrigé type :

Exo1



	A	B	C	&A	&B	&C	P1	P2	*P1	*P2
A←1, B←2, C←3 P1←&A, P2←&B	1	2	3	0xf0	0xf4	0xf8	0xf0	0xf4	1	2
P2= &C	1	2	3	0xf0	0xf4	0xf8	0xf0	**0xf8**	1	**3**
***P1 = *P2**	**3**	2	3	0xf0	0xf4	0xf8	0xf0	0xf8	**3**	3
(*P2)++	3	2	**4**	0xf0	0xf4	0xf8	0xf0	0xf8	3	**4**
P1= P2	3	2	4	0xf0	0xf4	0xf8	**0xf8**	0xf8	**4**	4
P2= &B	3	2	4	0xf0	0xf4	0xf8	0xf8	**0xf4**	4	**2**
***P2 = *P1 - 2 * *P2**	3	**0**	4	0xf0	0xf4	0xf8	0xf8	0xf4	4	**0**
(*P2)--	3	**-1**	4	0xf0	0xf4	0xf8	0xf8	0xf4	4	**-1**
C = (P2 == &C)	3	**-1**	0	0xf0	0xf4	0xf8	0xf8	0xf4	**0**	**-1**
***P2= *P1 + A**	3	**3**	0	0xf0	0xf4	0xf8	0xf8	0xf4	**0**	**3**

Exo2

1. *(&a) = a
2. *tab = tab[0]
3. *(tab + 0) = tab[0]
4. (*tab) + 1 = tab[0] + 1
5. &(tab[0]) = tab1
6. &(tab[i]) = (tab + i)
7. ++tab[i] = ++(*(tab + i))

Exo 3

```
Algorithme Fusion
Variables
    A,B : ^entier
    n, m, i :entier
Début
   Ecrire ('Donnez les dimensions n et m')
   Lire (n, m)
   Allouer (A, n)
   Pour i=0 A n-1 Faire
      Lire (A[i])
   Fin pour
   Allouer (B, m)
   Pour i=0 A m-1 Faire
      Lire (B[i])
   Fin pour
   Réallouer (A, n+m)
   Pour i=n A (n+m-1) Faire
       A[i]=B[i-n]
   Fin pour

Ecrire ('Le tableau après la fusion devient : ')
   Pour i=0 A (n+m-1) Faire
      Ecrire A[i]
Fin pour
Désallouer (A)
Désallouer (B)
Fin
```

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int *A,*B; int n,m,i;
printf("donnez la dimension de A:");
scanf("%d",&n);
printf("donnez la dimension de B:");
scanf("%d",&m);
 A=(int*)malloc(n*sizeof(int));
 for (i=0;i<n;i++)
{
printf("A[%d]=",i);
scanf("%d",&A[i]);
}
B=(int*)malloc(m*sizeof(int));
 for (i=0;i<m;i++)
{
printf("B[%d]=",i);
scanf("%d",&B[i]);
}
A=(int*)realloc(A,(n+m)*sizeof(int));
for (i=n;i<n+m;i++)
A[i]=B[i-n];
printf("Le tableau apres fusion \n");
for (i=0;i<n+m;i++)
printf ("A[%d]=%d \n",i,A[i]);
free (A);
free (B);
return 0;
}
```

Exo 4

Algorithme Chercher_indices
 	
Variables
constante entier N =
10 tab : tableau [N]
d'entier
iPdebut, iPfin,iPCompteur :^ entier
 	
Début
iPdebut=tab
iPfin=&tab[
N-1]
 	
Pour iPCompteur=iPdebut **A** iPfin **Faire**
Lire (tab[iPComteur-iPdebut])
Fin pour
 	
Pour iPCompteur=iPdebut **A** iPfin
Faire Si (*iPCompteur = 0) **Alors**
Ecrire('Indice Num',iPCompteur-iPdebut)
Fin
 	
Si Fin
	#include <stdio.h>
	#define N 10
	int main ()
	{
	int tab[N];
	int * iPdebut;
	int * iPfin;
	int * iCompteur;
	//iPdebut pointe sur le premier élément iPdebut=tab; //
	iPdebut=&tab[0];
	//iPfin pointe sur le dernier élément//
	iPfin=&tab[N-1];
	for (iCompteur=iPdebut; iCompteur <= iPfin;
	iCompteur++)
	{
	printf("tab[%d] = ",iCompteur-iPdebut);
	scanf("%d",&tab[iCompteur-iPdebut]);
	}
	//Pour tous les élements du tableau on boucle sur les
	//adresses
	for (iCompteur=iPdebut; iCompteur <= iPfin;
	iCompteur++)
	//Si le contenu de iCompteur est nul on affiche l'indice
	if (*iCompteur==0) printf(" indice num :%d\n",
	iCompteur-iPdebut);
	return 0;
	}

Exo 5

```
Algorithme Eliminer_element

 Variables
   A,P1 :^ entier
   N,X,i,k :entier

Début
     Ecrire ('Donnez la dimension du tableau')
     Lire (N)
     Allouer (A,N)

    Pour i=0 A (N-1)  Faire
       Lire (A[i])
    Fin pour

    Ecrire ('Introduire l'élément X à éliminer')
    Lire (X)
    P1=A
    k=0 ;

//Ecrasement des éléments égaux à X

    Pour i=0 A (N-1) Faire
       Si (A[i] ≠ X) Alors
           P1[k]=A[i]
           k ← k+1
     Fin Si

//Affichage du nouveau tableau

      Pour i=0 A (k-1) Faire
         Ecrire (A[i])
     Fin Pour
Désallouer (A)
Fin
```

```
#include    <stdio.h>
#include   <stdlib.h>
int main()
{
  /* Déclarations */
  int *A;    /* tableau donné        */
  int N,i;       /* dimension du tableau */
  int X ;    /* valeur à éliminer    */
  int k=0; /* Compteur*/
  int *P1; /* pointeurs d'aide     */

  /* Saisie des données */
  printf("Donnez la dimension du tableau : ");
  scanf("%d", &N );
  A=(int*)malloc(N*sizeof(int));
  for (i=0; i<N; i++)
      {
       printf("A[%d] : ", i);
       scanf("%d", &*(A+i));
      }
 printf("Introduire l'élément X à éliminer du tableau : ");
  scanf("%d", &X );
  P1=A;
  for (i=0; i<N; i++)
       if (A[i] != X)
         {
            P1[k] = A[i];
           k++;
         }
    /* Edition du résultat */
  for (i=0; i<k; i++)
      printf("%d ", A[i]);
      printf("\n");
 free (A);
  return 0;
}
```

Exo 6

```
#include  <stdio.h>
#include <stdlib.h>
main()
{
 /* Déclarations */
 int **U; /* matrice unitaire */
```

```
 int N;         /* dimension de la matrice unitaire */
 int I, J;      /* indices courants */

 /* Saisie des données */
 printf("Dimension de la matrice carrée: ");
 scanf("%d", &N);

 /* Réservation mémoire de la matrice carrée */
U = (int**) malloc (N * sizeof(int*));
for (I = 0; I < N; I++)
    U[I] = (int*) malloc (N * sizeof(int));

/* Construction de la matrice carrée unitaire */
 for (I=0; I<N; I++)
    for (J=0; J<N; J++)
       if (I==J)
          U[I][J]=1;
       else
          U[I][J]=0;

  /* Edition du résultat */
 printf("Matrice unitaire de dimension %d :\n", N);
 for (I=0; I<N; I++)
   {
    for (J=0; J<N; J++)
        printf("%7d", U[I][J]);
    printf("\n");
   }

          for (I=0; I<N; I++)
          free(U[I]);
          free(U);
 return 0;
}
```

Exo 7

```
  #include <stdio.h>
#include  <stdlib.h>
main()
{
 /* Déclarations */
 int **U; /* matrice  */
 int N;         /* dimension de la matrice */
 int I, J;      /* indices courants */
 int Tr;      /* Trace de la matrice */

 /* Saisie des données */
```

```
 printf("Dimension de la matrice
 carrée: "); scanf("%d", &N);

 /* Réservation mémoire de la matrice
carrée */ U = (int**) malloc (N *
sizeof(int*));
for (I = 0; I < N; I++)
    U[I] = (int*) malloc (N * sizeof(int));

/* Lecture de la matrice
 carrée */ for (I=0; I<N;
 I++)
    for (J=0; J<N; J++)
    {
      printf("U[%d,%d]:",I,J);
      scanf("%d",&U[I][J]);
    }
/* Affichage de la matrice carrée */
  printf("Matrice source  :\n");
 for (I=0; I<N; I++)
   {
    for (J=0; J<N; J++)
    printf("%7d",
    U[I][J]);
    printf("\n");
   }
  /* Transformation de la matrice carrée en matrice
  diagonale */ for (I=0; I<N; I++)
    for (J=0; J<N; J++)
                    i
          f (I!=J)
          U[I][J]
          =0;
  /* Edition du résultat */
 printf("Matrice diagonale %d
 :\n", N); for (I=0; I<N; I++)
   {
    for (J=0; J<N; J++)
    printf("%7d",
    U[I][J]);
    printf("\n");
   }

  /* Calcule et affichage de la
trace */ Tr=0;
 for (I=0; I<N;
   I++) for
   (J=0; J<N;
   J++) if
   (I==J)
```

```
 Tr=Tr+U[I][J];
 printf("La trace de la Matrice Tr= %d:\n", Tr); for (I=0; I<N; I++)
 free(U[I]); free(U);
  return 0;
}
```

Programme C :

```
#include<stdio.h>
int A,B,C; //Variables globales
int *P1,*P2; //Variables globales
void imprimer()
{
printf ("A=%d B=%d C=%d &A=%p &B=%p &C=%p \nP1=%p P2=%p *P1=%d
*P2=%d \n \n", A,B,C,&A,&B,&C,P1,P2,*P1,*P2);
}
int main()
{
A=1;
B=2;
C=3;
printf("P1=&A P2=&B\n");
P1=&A; P2=&B;
imprimer();
printf("P2=&C \n");
P2= &C;
imprimer();
printf("*P1=*P2 \n");
*P1 = *P2;
imprimer();
printf("(*P2)++ \n");
(*P2)++ ;
imprimer();
printf("P1=P2 \n");
P1        = P2;
imprimer();
printf("P2=&B \n");
P2        = &B;
imprimer();
printf("*P2 = *P1 - 2 * *P2 \n");
*P2 = *P1 - 2 * *P2;
imprimer();
printf("(*P2)-- \n");
(*P2)--;
imprimer();
printf("C = P2 == &C \n");
C = (P2 == &C);
imprimer();
printf("*P2+= *P1+A \n");
*P2= *P1 + A;
imprimer();
return 0;
}
```