TD 05 : Les pointeurs & l'allocation dynamique
(3h)

Consignes :

⇒ Ce travail est INDIVIDUEL
⇒ Chaque étudiant doit répondre à TOUS les exercices
⇒ L'utilisation d'un IDE est INTERDITE
⇒ Support de cours à consulter : CM3

Acquis d'apprentissage :

A la fin de ce TD, chaque étudiant doit :
- Faire la différence entre les opérateurs & et *
- Maîtriser l'arithmétique des pointeurs
- Comprendre un programme utilisant des pointeurs
- Capable d'allouer / libérer dynamiquement de l'espace mémoire.

Compréhension & déroulement

Exercice 1

Quelle est la sortie du programme ci-dessous ?

```c
#include <stdio.h>
int main()
{
    int * p = NULL;
    int x=17;
    p=&x;
    printf("%d %d %d \n",&p, p, *p);
    return 0;
}
```

Exercice 2

Compléter le tableau ci-dessous en indiquant les contenus des variables après l'exécution de chacune des instructions suivantes :

```
I1  : int A = 1;
I2  : int B = 2;
I3  : int C = 3;
I4  : int *P1 = NULL, *P2 = NULL;
I5  : P1=&A;
I6  : P2=&C;
I7  : *P1=(*P2)++;
I8  : P1=P2;
I9  : P2=&B;
I10 : *P1-=*P2;
I11 : ++*P2;
I12 : *P1*=*P2;
I13 : A=++*P2**P1;
```

I_{14} : `P1=&A;`
I_{15} : `*P2=*P1/=*P2;`

Adresse en mémoire	Nom symbolique	Valeur														
		I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	I_{11}	I_{12}	I_{13}	I_{14}	I_{15}
5987	A	**1**	**1**	**1**												
254	B		**2**	**2**												
777	C			**3**												
2022	P1															
1994	P2															

Exercice 3

Pour chacune des expressions du tableau ci-dessous, donner :

- Leurs expressions équivalentes en ajoutant des parenthèses montrant l'ordre dans lequel elles sont évaluées.
- La valeur obtenue à l'affichage de chacune d'entre elles.

Déclarations et initialisations

```
int i = 3, j = 5, *p = &i, *q = &j, *r = NULL;
double x;
```

Expression	Expression équivalente	Valeur
`p == &i`	p == (&i)	
`* * & p`		
`r = & x`		
`7 * * p / * q + 7`		
`* (r = & j) *= * p`		

Exercice 4

Soient les trois instructions suivantes :

```
char T[9]={12,23,34,45,56,67,78,89,90};
char *p = NULL;
p=T;
```

En supposant que le tableau ***T*** est stocké à l'adresse 5000 en mémoire de l'ordinateur, que la variable ***p*** est stockée à l'adresse 2804 et qu'un pointeur occupe 8 octets, donner les valeurs et/ou adresses fournies par l'affichage de chacune de ces expressions.

```
(a) *p+2
(b) *(p+2)
(c) &p+1
(d) &T[4]-3
(e) T+3
(f) &T[7]-p
(g) p+(*p-10)
(h) *(p++(p+8)-T[7])
(i) (*(&p))[5]-(*(&p))[2]
(j) *(*(&p)+3)*(*(*(&p)+1)-13)
```

Indication : Pour bien réussir cet exercice, représentez vos variables dans le tableau suivant :

Adresse en mémoire	Nom symbolique de la variable	Valeur
2804	p	5000

Exercice 5

Quelle est la sortie du programme C suivant ?

```c
#include <stdio.h>
int main()
{
    int T[5] = {-6, 3, 15, 11, 76};
    int *p = NULL, i;

    p = T;

    for (i = 0; i < 5; i++)
    {
        printf("%d : %d\n", p - T, *p);
        p++;
    }
    return 0;
}
```

Exercice 6

Dérouler le programme C suivant en montrant l'état des pointeurs à chaque étape puis déduire son rôle.

```c
#include <stdio.h>
int main()
{
    int T[100];
    int N, X;
    int *P1 = NULL, *P2 = NULL;
```

```
    P1 = T;

    printf("Veuillez saisir la taille du tableau : ");
    scanf("%d", &N);

    for(P1 = T; P1 < N + T; P1++)
    {
        printf("T[%d] = ", P1 - T);
        scanf("%d", P1);
    }

    scanf("%d", &X);

    P2 = T;
    for(P1 = T; P1 < T + N; P1++)
    {
        *P2 = *P1;
        if (*P2 != X)
        {
            P2++;
        }
    }

    N = P2 - T;

    for(P2 = T; P2 < N + T; P2++)
    {
        printf("%d \t\t", *P2);
    }
    return 0;
}
```

Exercice 7

Considérons la déclaration de de la variable `JOUR` comme suit :

```
char *JOUR[] = {"dimanche", "lundi", "mardi",
                "mercredi", "jeudi", "vendredi",
                "samedi"};
```

1. Quel est le type de la variable `JOUR` ?
2. Représenter graphiquement la mémorisation de la variable `JOUR`
3. Que va afficher le programme suivant ?

```
int I;
for (I=0; I<7; I++)
    printf("%s\n", JOUR[I]);
```

4. Que va afficher celui-ci ?

```
int I;
for (I=0; I<7; I++)
    printf("%c\n", *JOUR[I]);
```

Exercice 8

Analyser le programme C ci-dessous puis compléter le tableau qui le suit en indiquant par un **[X]** ce qui est stocké sur le tas et ce qui est stocké sur la pile.

```c
#include <stdio.h>
#include <stdlib.h>

int* allocation(int size)
{
    int* array;
    array = (int*)malloc(size * sizeof(int));
    return array;
}

int main()
{

    int var = 10;
    int sizeArr = 5;
    int* Arr = allocation(sizeArr);

    for (int i = 0; i < sizeArr; i++)
    {
        Arr[i] = i * 2;
    }

    printf("Variable : %d\n", var);
    printf("Array content : ");
    for (int i = 0; i < sizeArr; i++) {
        printf("%d ", Arr[i]);
    }
    printf("\n");

    free(Arr);

    return 0;
}
```

	Pile	Tas
int var		
int sizeArr		
int* Arr		
int i		
int* array		
Contenu du tableau: **0 2 4 6 8**		

Le nom de la function : allocation		

Laquelle des deux mémoires est libérée par la fonction **free()** ?

Exercice supplémentaire : Optionnel

Soient les trois instructions suivantes :

```
int T[9]={12,23,34,45,56,67,78,89,90};
int *p;
p=T;
```

En supposant que le tableau ***T*** est stocké à l'adresse 4000 en mémoire de l'ordinateur, qu'un entier occupe 4 octets en mémoire, que la variable ***p*** est stockée à l'adresse 2606 et qu'un pointeur occupe 8 octets, donner les valeurs et/ou adresses fournies par l'affichage de chacune de ces expressions.

```
(a) *p+2
(b) *(p+2)
(c) &p+1
(d) &T[4]-3
(e) T+3
(f) &T[7]-p
(g) p+(*p-10)
(h) *(p+*(p+8)-T[7])
(i) (*(&p))[5]-(*(&p))[2]
(j) *(*(&p)+3)*(*(*(&p)+1)-13)
```

Indication : Pour bien réussir cet exercice, représentez vos variables dans le tableau suivant :

Adresse en mémoire	Nom symbolique de la variable	Valeur
2606	p	4000

Détection d'erreurs

Exercice 1

Des erreurs (syntaxiques / sémantiques) ont été glissées dans les programmes ci-dessous. Repérez-les et proposez des corrections.

a.

```
int * p;
int x=34;
*p=x;
```

b.

```c
int x=17;
int * p=x;
*p=17;
```

c.

```c
double * q;
int x=17;
int * p=&x;
q=p;
```

d.

```c
int x;
int * p;
&x=p;
```

e.

```c
char mot[10];
char car='A';
char * pc=&car;
mot=pc;
```

f.

```c
 #include <stdio.h>
int main()
{
    char * cours1 = "programmation";
    char cours2[] = "algorithmique";

    cours1[0] = 'P';
    cours2[0] = 'A';

    printf("%s et %s.", cours1, cours2);
    return 0;
}
```

Programmation

Exercice 1

En utilisant seulement des variables allouées dynamiquement, nous souhaitons écrire un programme qui calcule puis affiche la somme de tous les nombres impairs de 1 à ***n.***

En vous appuyant sur les commentaires, compléter le code ci-dessous.

```c
#include <stdio.h>
// Inclure la librairie qui permet d'utiliser les fonctions de l'allocation dynamique
……

int main()
{
    // Allouer dynamiquement l'entier n en utilisant un malloc
    ……

    // Allouer dynamiquement l'entier s en utilisant un calloc
    ……

    printf("Veuillez saisir un nombre :");
```

```
    // Saisie de la valeur de n
    ……

    // Initialisation de s
    ……

    // Compléter la condition d'arrêt
    for(int i=1; …… ; i+=2)
    {
        // Effectuer la somme
        ……
    }

    // Affichage du résultat
    printf("La somme des nombres impairs ……);

    // Libérer l'espace alloué pour n
    ……

    // Libérer l'espace alloué pour s
    ……

    return 0;
}
```

Exercice 2

Écrire le programme C qui permet de réaliser le fonctionnement suivant :

1. L'utilisateur choisit le nombre de valeurs réelles qu'il veut insérer
2. Réservation dynamique de la mémoire
3. L'utilisateur saisit des valeurs réelles
4. Calcul de la moyenne
5. Affichage de la moyenne
6. Demander à l'utilisateur s'il souhaite ajouter d'autres valeurs
7. Si la réponse est oui
    a. L'utilisateur choisit le nombre de valeurs à ajouter
    b. Modification de la mémoire déjà allouée
    c. L'utilisateur saisit les nouvelles valeurs réelles
    d. Calcul de la nouvelle moyenne
    e. Affichage de la nouvelle moyenne
    f. Aller à l'étape 6.
8. Sinon afficher le résultat de la moyenne à l'écran
9. Libérer l'espace alloué