École Polytechnique Sousse
Département Informatique
Section : Génie Informatique & Télécom et réseaux
Niveau : 3^{eme} Année , AU : 2020-2021

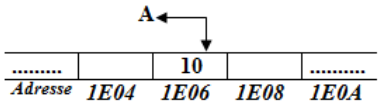
LANGAGE C
LES POINTEURS

1 Introduction

D'une manière générale, le langage C permet de manipuler des adresses par l'intermédiaire de variables nommées "Pointeurs".

1.1 Adressage direct

Accès au contenu d'une variable par le nom de la variable.
Exemple :
int A ; A=10 ;



1.2 Adressage indirect :

Si nous ne voulons pas utiliser le nom d'une variable A, nous pouvons copier l'adresse de cette variable dans une variable spéciale P appelée Pointeur. Ensuite, nous pouvons retrouver l'information de la variable A en passant par le pointeur P.
Accès au contenu d'une variable en passant par un pointeur qui contient l'adresse de la variable.
Exemple :
Soit A une variable contenant la valeur 10 et P un pointeur qui contient l'adresse de A.

2 Les pointeurs

2.1 Définition

Un pointeur est une variable spéciale qui peut contenir l'adresse d'une autre variable. Si un pointeur P contient l'adresse d'une variable A, on dit que 'P pointe sur A'.
Remarque : En C, chaque pointeur est limité à un type de données. Il peut contenir :
— L'adresse d'une variable simple de ce type,
— L'adresse d'une composante d'un tableau de ce type.

2.2 Les opérateurs de base

Lors du travail avec des pointeurs, nous avons besoin :
— D'un opérateur 'Adresse de' : & pour obtenir l'adresse d'une variable.
— D'un opérateur 'contenu de' : * pour accéder au contenu d'une adresse.
— D'une syntaxe de déclaration pour pouvoir déclarer un pointeur.

2.2.1 a) L'opérateur 'adresse de' : &

&< nom variable> fournit l'adresse de la variable < nom variable>.

2.2.2 b) L'opérateur 'contenu de' : *

***< nom pointeur >** désigne le contenu de l'adresse référencée par le pointeur <nom pointeur> .

2.2.3 c) Déclaration d'un pointeur :

<type > *<nom pointeur>
Déclare un pointeur <nom pointeur> qui peut recevoir des adresses de variables.
Exemple 1 :
L'opérateur unaire d'indirection * permet d'accéder directement à la valeur de l'objet pointé. Ainsi si p est un pointeur vers un entier i, *p désigne la valeur de i. Par exemple, le programme :

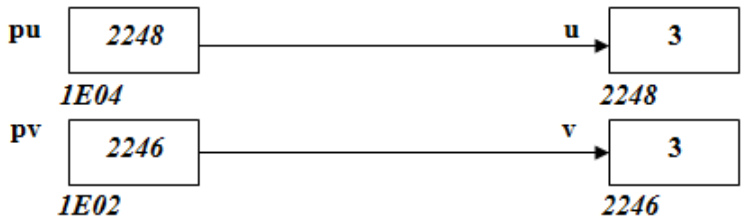
```
#include<stdio.h>
void main()
{ int  i =3;
  int* p ;//p est un pointeur sur une variable de type entier
  p=&i;//p contient adresse de i
  printf ("\n contenu de la case mémoire pointé par p est:  %d",*p);
  *p=5;//changement de i à travers p
  printf ("\n i = %d et *p = %d",i,*p);
}
```

Résultat :
contenu de la case mémoire pointé par p est : 3
i = 5 et *p = 5
Dans ce programme, les objets i et *p sont identiques. Ils ont même adresse et valeur. Cela signifie en particulier que toute modification de *p modifie i.
Exemple 2 :

```
#include <stdio.h>
void main ( )
{   int u=3,v ,*pu,*pv;
    printf ("1**\t u=%d\t&u=%x \n", u, &u);
    pu= &u ;
    printf ("2**\t *pu=%d\t pu=%x\n ", *pu, pu) ;
    pv = &v ; v =*pu ;
    printf ("3**\t  v=%d\t &v=%x\n ", v, &v) ;
    printf ("4**\t *pv=%d\t pv=%x\n ",*pv, pv) ;
}
```

Résultat :

1** u = 3 &u = 2248
2** *pu= 3 pu = 2248
3** v = 3 &v = 2246
4** *pv = 3 pv = 2246



Remarque :
Un pointeur qui ne pointe sur aucune variable est un pointeur nul. On lui affecte l'adresse 0.
Exemple :
int *p ;
p = 0 ; $\Longleftrightarrow$ p = NULL ;/*NULL indique que l'adresse mémoire est invalide.*/

3 Arithmétique des pointeurs

Exemple :
int *P, X, Y ;
P = &X
Y = *P+1 $\Longleftrightarrow$ Y = X+1
$*$P = *P+10 $\Longleftrightarrow$ X = X+10
$*$P + =2 $\Longleftrightarrow$ X + =2
++ *P $\Longleftrightarrow$ ++ X
(*P)++ $\Longleftrightarrow$ X ++
Dans le dernier cas, les parenthèses sont nécessaires : comme les opérateurs * et ++ sont évalués de droite à gauche, sans les parenthèses le pointeur P serait incrémenté, non pas l'objet sur lequel P pointe.

4 Paramètres d'une fonction

L'appel d'une fonction se fait par son nom suivi des paramètres effectifs. Ces paramètres doivent correspondre en nombre et en type aux paramètres formels spécifiés dans la définition de la fonction.En C, on distingue 2 méthodes pour passer des paramètres effectifs à une fonction :

4.1 Passage des paramètres par valeur

Exemple :

```
#include<stdio.h>
void PERMUTER (int A, int B)
{   int AIDE;
    AIDE = A;
    A = B;
    B = AIDE;
    printf("Dans PERMUTER : A=%d\t B=%d\n",A,B);
}
void main()
{
    int X=3,Y=4;
    printf("Avant appel de PERMUTER : X=%d\t Y=%d\n", X,Y);
    PERMUTER (X,Y);
    printf("Après appel de PERMUTER : X=%d\t Y=%d", X,Y);
}
```

Exécution :
Avant appel de PERMUTER : X = 3 Y = 4
Dans PERMUTER : A = 4 B = 3
Après appel de PERMUTER : X = 3 Y = 4
X et Y restent échangés.

- Lors de l'appel, les valeurs de X et Y sont copiées dans les paramètres A et B. PERMUTER échange bien le contenu des variables locales A et B, mais les valeurs de X et Y restent les mêmes.
- Pour changer la valeur d'une variable de la fonction appelante, nous allons procéder comme suit :
  * La fonction appelante doit fournir l'adresse de la variable et
  * La fonction appelée doit déclarer le paramètre comme pointeur.

4.2 Passage des paramètres par adresse

Exemple :

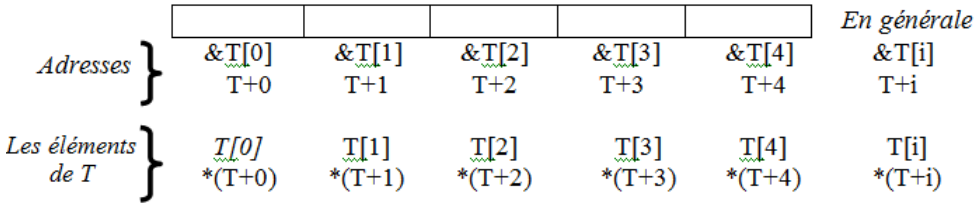
```
#include<stdio.h>
void PERMUTER (int *A, int *B)
{   int AIDE;
    AIDE = *A;
    *A = *B;
    *B = AIDE;
    printf(" Dans PERMUTER : A=%d\t B=%d\n",*A,*B);
}
void main()
{   int X=3, Y=4;
    printf("Avant appel de PERMUTER : X=%d\t  Y=%d\n ", X, Y);
    PERMUTER (&X, &Y);
    printf("Après appel de PERMUTER : X=%d\t  Y=%d\n ", X, Y);
}
```

***Exécution* :**
Avant appel de PERMUTER : X = 3 Y = 4
Dans PERMUTER : A = 4 B = 3
Après appel de PERMUTER : X = 4 Y = 3
Lors de l'appel, les adresse de X et Y sont copiées dans les pointeurs A et B. PERMUTER échange ensuite le contenu des adresses indiquées par les pointeurs A et B.

5 Pointeurs et Tableaux à une dimension

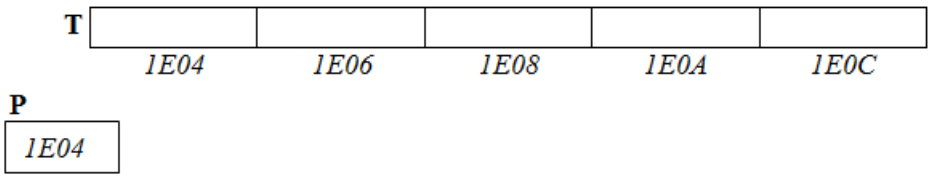
5.1 Adressage des composantes d'un tableau

Lorsqu'on déclare un tableau, son nom est considéré comme l'adresse de son premier élément. Donc, on définit automatiquement un pointeur sur le premier élément du tableau. Ce pointeur est le nom du tableau.
Exemple 1 :
int T[5] ;



Exemple 2 :

```
int T[5] ;
int *P ;
P = T ;  /*P=&T[0]*/
```



Si P pointe sur une composante quelconque d'un tableau, alors P+1 pointe sur la composante suivante. Plus généralement :

- P+i pointe sur la i-ème composante derrière P.
- *(P+0) désigne le contenu de T[0].
- *(P+1) désigne le contenu de T[1].
- .....
- *(P+i) désigne le contenu de T[i].

***Remarque* :**
Il existe toujours une différence essentielle entre un pointeur et le nom d'un tableau :

- Un pointeur est une variable, donc des opérations comme P =T ou P++ sont permises.
- Le nom d'un tableau est une constante, donc des opérations comme T = P ou T++ sont impossibles.

Application 1
Chargement et affichage des éléments d'un tableau
$1^{ère}$ ***méthode***

```
#include<stdio.h>
void main()
{
    int A[50], i, N;
    printf("Donner la taille de A :");
    scanf("%d", &N);
    for(i=0 ; i<N ; i++)
    {
        printf("A[%d]:", i);
        scanf("%d", A+i);
    }
    for (i=0; i<N; i++)
        printf("%d\t", *(A+i));
}
```

$2^{ème}$ ***méthode*** **:**

```
#include<stdio.h>
void main()
{
    int A[50],N;
    int *P;
    printf("Donner la taille de A :");
    scanf("%d", &N);
    for(P=A; P<A+N; P++)
    {
        printf("A[%d]:", P-A);
        scanf("%d", P);
    }
    for (P=A; P<A+N; P++)
        printf("%d\t",*P);
}
```

Application 2 :
Copier les éléments positifs d'un tableau T dans un deuxième tableau POS.

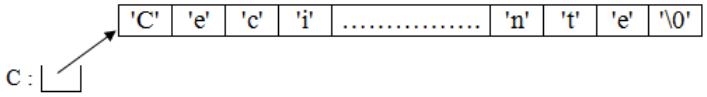
```
#include<stdio.h>
void main()
{   int T[10] = {-3, 4, 0, -7, 3, 8, 0, -1, 4, -9} , POS[10];
    int *P1,*P2;  /* pointeurs courants pour T et POS */
    for (P1=T, P2=POS ; P1<T+10 ; P1++)
        if (*P1>0)
        {  *P2 = *P1;
            P2++;
        }
    for(P1=POS; P1<P2; P1++)
        printf("%d\t",*P1);
}
```

6 Pointeurs et chaînes de caractères

De la même façon qu'un pointeur sur int peut contenir l'adresse d'un nombre isolé ou d'une composante d'un tableau, un pointeur sur char peut pointer sur un caractère isolé ou sur les éléments d'un tableau de caractères. Un pointeur sur char peut en plus contenir l'adresse d'une chaîne de caractères constante et il peut même être initialisé avec une telle adresse.

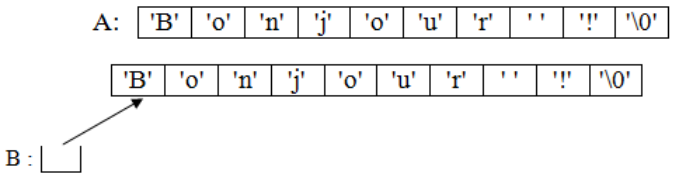
6.1 Affectation

On peut attribuer l'adresse d'une chaîne de caractères constante à un pointeur sur char :
Exemple :
char *C ;
C = "Ceci est une chaîne de caractères constantes" ;



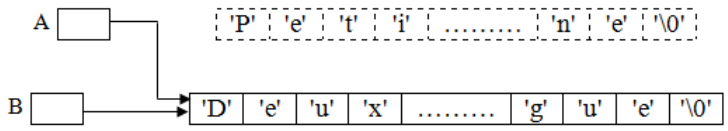
6.2 Initialisation

Un pointeur sur char peut être initialisé lors de la déclaration si on lui affecte l'adresse d'une chaîne de caractères constante :
char *B = "Bonjour !" ;
N.B.
Il existe une différence importante entre les deux déclarations :
char A[] = "Bonjour !" ; /* un tableau */
char *B = "Bonjour !" ; /* un pointeur */
A est un tableau qui a exactement la grandeur pour contenir la chaîne de caractères et la terminaison '\0'. Les caractères de la chaîne peuvent être changés, mais le nom A va toujours pointer sur la même adresse en mémoire.
B est un pointeur qui est initialisé de façon à ce qu'il pointe sur une chaîne de caractères constante stockée quelque part en mémoire. Le pointeur peut être modifié et pointer sur une chose. La chaîne constante peut être lue, copiée ou affichée, mais pas modifiée.



6.3 Modification

Si nous affectons une nouvelle valeur à un pointeur sur une chaîne de caractères constante, nous risquons de perdre la chaîne constante. D'autre part, un pointeur sur char a l'avantage de pouvoir pointer sur des chaînes de n'importe quelle longueur :
Exemple :
char *A = "Petite chaîne" ;
char *B = "Deuxième chaîne un peu plus longue" ;



A = B ;
Maintenant A et B pointent sur la même chaîne ; la "Petite chaîne" est perdue :

N.B.

Les affectations discutées ci-dessus ne peuvent pas être effectuées avec des tableaux de caractères :

Exemple :

```
char A[50] = "Petite chaîne" ;
char B[50] = "Deuxième chaîne un peu plus longue" ;
char C[30] ;
A = B ;          /* IMPOSSIBLE  ERREUR !!!*/
C = "Bonjour !" ; /* IMPOSSIBLE  ERREUR !!!*/
```

Application :

Reprenons l'exemple de la fonction « strcpy », qui copie la chaîne CH2 vers CH1.

```
void strcpy(char *CH1, char *CH2)
{
    char *P1=CH1,*P2=CH2;
    while (*P2 != '\0')
    {   *P1=*P2;
        P1++;
        P2++;
    }
    *P1='\0';
}
void main()
{
    char CH1[30],CH2[30];
    printf("Donner CH1 :"); gets(CH1);
    printf("Donner CH2 :"); gets(CH2);
    strcpy(CH1, CH2);
    printf("La chaine résultat :"); puts(CH1);
}
```

7 Allocation dynamique

Problème :

Souvent, nous devons travailler avec des données dont nous ne pouvons pas prévoir le nombre et la grandeur lors de la programmation. Ce serait alors un gaspillage de réserver toujours l'espace maximal prévisible. Il nous faut donc un moyen de gérer la mémoire lors de l'exécution du programme.

7.1 La fonction « malloc » et l'opérateur « sizeof »

7.1.1 La fonction « malloc » :

- La fonction « malloc » de la bibliothèque <stdlib> nous aide à localiser et à réserver de la mémoire au cours d'un programme.
- La fonction malloc() retourne un pointeur de type void * sur l'espace réservé. Il faut forcer le type de retour de malloc (faire un cast) pour récupérer un pointeur de type voulu. S'il n'y a pas d'espace libre, malloc() retourne le pointeur nul.

Syntaxe : malloc(<N>)
$\Longleftrightarrow$ Fournit l'adresse d'un bloc en mémoire de <N> octets libres ou la valeur zéro s'il n'y a pas assez de mémoire.
Exemple :
Supposons que nous avons besoin d'un bloc en mémoire pour un texte de 4000 caractères. Nous disposons d'un pointeur T sur char (char *T). Alors l'instruction :
T = (char *)malloc(4000) ;
$\Longleftrightarrow$ Fournit l'adresse d'un bloc de 4000 octets libres et l'affecte à T. S'il n'y a plus assez de mémoire, T obtient la valeur zéro.

7.1.2 L'opérateur unaire sizeof :

- sizeof <var> : fournit la grandeur de la variable <var>
- sizeof <const> :fournit la grandeur de la constante <const>

Exemple 1 :
Après la déclaration,
short A[10] ;
char B[5][10] ;
Nous obtenons les résultats suivants :

- sizeof A s'évalue à 20
- sizeof B s'évalue à 50
- sizeof "Bonjour !" s'évalue à 10
- sizeof(double) s'évalue à 8

Exemple 2 :

```
char *pc;
int *pi,*pj;
float *pr;
pc=(char*)malloc(10);/*Réserver 10 cases mémoire, soit la place pour 10 caractères*/
pi=(int*)malloc(16);/*Réserver 16 cases mémoire,soit la place pour 4 entiers*/
pr=(float*)malloc(24); /*Réserver 16 cases mémoire,soit la place pour 8 entiers*/
pj=(int*)malloc(3*sizeof(int)); /* Réserver la place en mémoire pour 3 entiers */
```

7.2 La fonction free

Si nous n'avons plus besoin d'un bloc de mémoire que nous avons réservé à l'aide de « malloc », alors nous pouvons le libérer à l'aide de la fonction « free » de la bibliothèque <stdlib>.
Syntaxe :
free(<Pointeur>)
$\Longleftrightarrow$ Libère le bloc de mémoire désigné par le <Pointeur> ; n'a pas d'effet si le pointeur a la valeur zéro.
Si nous ne libérons pas explicitement la mémoire à l'aide free, alors elle est libérée automatiquement à la fin du programme.

Exercice d'application :
Nous voulons réserver de la mémoire pour X cases d'un tableau du type int (la valeur de X est lue au clavier), charger le tableau, inverser et afficher son contenu. Utiliser la fonction «malloc», «sizeof» et «free»

```
#include <stdio.h>
void main()
{
    int *p, n, i, j, aux;
    printf("donner le nombre de cases du tableau :");
    scanf("%d",&n);
    p=(int*)malloc( n* sizeof(int));
    for(i=0; i<n; i++)
    {
        printf("Donner l'élément %d  :",i);
        scanf("%d",p+i);
    }
    printf("Contenu de T avant inversion \n");
    for(i=0;i<n;i++)
        printf("%d\t",*(p+i));
    for(i=0, j=n-1; i<j ; i+ +, j --)
    {
         aux=  *(p+i);
        *(p+i) = *(p+j);
        *(p+j) = aux;
    }
    printf("\nContenu de T aprés inversion \n");
    for(i=0;i<n;i++)
        printf("%d\t",*(p+i));
    free(p);
}
```

7.3 Les pointeurs sur structures et accès aux données.

Soit le type suivant :

```
#include<stdio.h>
typedef struct enreg
{
    float CIN;
    char nom[30];
    char prenom[30];
    char classe[10];
    float moyenne;
}etudiant;
etudiant *terminal ; /*Déclaration*/
```

terminal est un pointeur vers une structure de type étudiant.
Pour accéder à l'élément nom on utilise la notation suivante : ->
Nom_de_la_variable -> membre ; Où (*Nom_de_la_variable).membre ;
Exemple :
Terminal->nom ou (*terminal).nom