En C, les pointeurs jouent un rôle primordial dans la définition de fonctions: Comme le passage des paramètres en C se fait toujours par la valeur, les pointeurs sont le seul moyen de changer le contenu de variables déclarées dans d'autres fonctions.

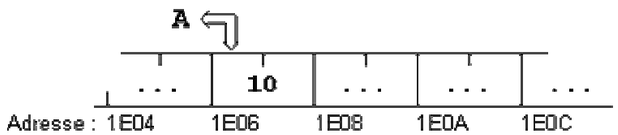
1. Adressage de variables

1.1. Adressage direct

Adressage direct: Accès au contenu d'une variable par le nom de la variable.

Exemple

```
short A;
A = 10;
```

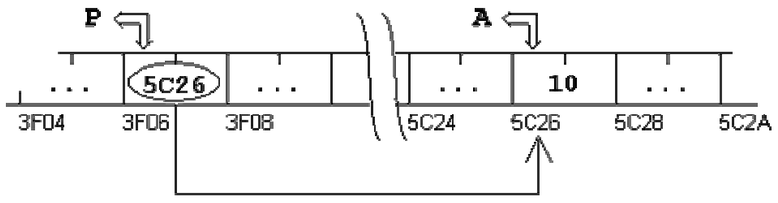


1.2. Adressage indirect

Adressage indirect: 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. En mémoire, A et P peuvent se présenter comme suit:



2. Les pointeurs

Un pointeur est une variable spéciale qui peut contenir l'adresse d'une autre variable.

En C, chaque pointeur est limité à un type de données. Il peut contenir l'adresse d'une variable simple de ce type ou l'adresse d'une composante d'un tableau de ce type.

---

http://www.ltam.lu/cours-c/prg-c02.htm

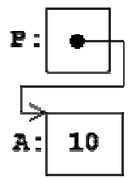
2.1. Les opérateurs de base

L'opérateur 'adresse de' : &

&<NomVariable> fournit l'adresse de la variable <NomVariable>

Soit P un pointeur non initialisé et A une variable (du même type) contenant la valeur 10 :

Alors l'instruction **P = &A** affecte l'adresse de la variable A à la variable P.

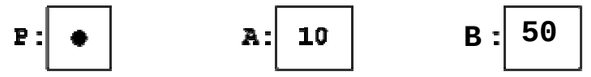


***L'opérateur 'contenu de' : **

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

Exemple

Soit A une variable contenant la valeur 10, B une variable contenant la valeur 50 et P un pointeur non initialisé:



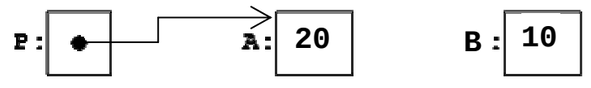
Après les instructions,

```
P = &A;
B = *P;
*P = 20;
```

- P pointe sur A,

- le contenu de A (référencé par *P) est affecté à B,

- le contenu de A (référencé par *P) est mis à 20.



Déclaration d'un pointeur

<Type> *<NomPointeur>
déclare un pointeur <NomPointeur> qui peut recevoir des adresses de variables du type <Type>

Exemple

```
|main()                    ou bien  |main()
|{                                  |{
| /* déclarations */                |/* déclarations */
| short A = 10;                     | short A, B, *P;
| short B = 50;                     | /* traitement */
| short *P;                         | A = 10;
| /* traitement */                  | B = 50;
| P = &A;                           | P = &A;
| B = *P;                           | B = *P;
| *P = 20;                          | *P = 20;
| return 0;                         | return 0;
|}                                  |}
```

3. Pointeurs et tableaux

3.1. Adressage des composantes d'un tableau

Le nom d'un tableau représente l'adresse de son premier élément. En d'autre termes:

&tableau[0] et **tableau** sont une seule et même adresse.

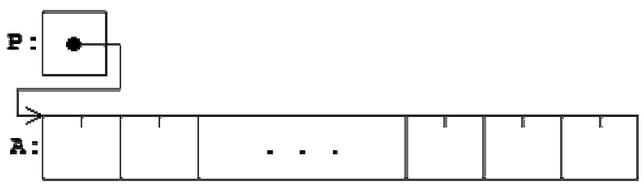
Exemple 1

En déclarant un tableau A de type **int** et un pointeur P sur **int**,

```
int A[10];
int *P;
```

l'instruction:

P = A; est équivalente à **P = &A[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-ième composante derrière P et
P-i pointe sur la i-ième composante devant P.

Ainsi, après l'instruction,

```
P = A;
```

le pointeur P pointe sur A[0], et

***(P+1)** désigne le contenu de A[1]
***(P+2)** désigne le contenu de A[2]
… **…**
***(P+i)** désigne le contenu de A[i]

Exemple 2

Soit A un tableau contenant des éléments du type **float** et P un pointeur sur **float**:

```
float A[20], X;
float *P;
```

Après les instructions,

```
P = A;
X = *(P+9);
```

X contient la valeur du 10-ième élément de A, (c.-à-d. celle de A[9]). Une donnée du type **float** ayant besoin de 4 octets, le compilateur obtient l'adresse P+9 en ajoutant 9 * 4 = 36 octets à l'adresse dans P.

Rassemblons les constatations ci dessus :

Comme A représente l'adresse de A[0],

***(A+1)** désigne le contenu de A[1]
***(A+2)** désigne le contenu de A[2]
…
***(A+i)** désigne le contenu de A[i]

Soit un tableau A d'un type quelconque et i un indice pour les composantes de A, alors

A	désigne l'adresse de	**A[0]**
A+i	désigne l'adresse de	**A[i]**
***(A+i)**	désigne le contenu de	**A[i]**

Si P = A, alors

P	pointe sur l'élément	**A[0]**
P+i	pointe sur l'élément	**A[i]**
***(P+i)**	désigne le contenu de	**A[i]**

Formalisme tableau et formalisme pointeur

Les deux programmes suivants copient les éléments positifs d'un tableau T dans un deuxième tableau POS.

Formalisme tableau

```
main()
{
 int T[10] = {-3, 4, 0, -7, 3, 8, 0, -1, 4, -9};
 int POS[10];
 int I,J;  /* indices courants dans T et POS */
 for (J=0,I=0 ; I<10 ; I++)
    if (T[I]>0)
       {
        POS[J] = T[I];
        J++;
       }
 return 0;
}
```

Nous pouvons remplacer la notation **tableau[I]** par ***(tableau + I)**, ce qui conduit à ce programme:

Formalisme pointeur

```
main()
{
 int T[10] = {-3, 4, 0, -7, 3, 8, 0, -1, 4, -9};
 int POS[10];
 int I,J;  /* indices courants dans T et POS */
 for (J=0,I=0 ; I<10 ; I++)
    if (*(T+I)>0)
       {
        *(POS+J) = *(T+I);
        J++;
       }
 return 0;
}
```

3.2. Arithmétique des pointeurs

Toutes les opérations avec les pointeurs tiennent compte automatiquement du type et de la grandeur des objets pointés.

- Affectation par un pointeur sur le même type

Soient P1 et P2 deux pointeurs sur le même type de données, alors l'instruction

P1 = P2; fait pointer P1 sur le même objet que P2

- Addition et soustraction d'un nombre entier

Si P pointe sur l'élément A[i] d'un tableau, alors

P+n	pointe sur A[i+n]
P-n	pointe sur A[i-n]

- Incrémentation et décrémentation d'un pointeur

Si P pointe sur l'élément A[i] d'un tableau, alors après l'instruction

P++;	P pointe sur A[i+1]
P+=n;	P pointe sur A[i+n]
P--;	P pointe sur A[i-1]
P-=n;	P pointe sur A[i-n]

Domaine des opérations

L'addition, la soustraction, l'incrémentation et la décrémentation sur les pointeurs sont seulement définies *à l'intérieur d'un tableau*. Si l'adresse formée par le pointeur et l'indice sort du domaine du tableau, alors le résultat n'est pas défini.

Exemples

```
int A[10];
int *P;
P = A+9;    /* dernier élément -> légal */
P = A+10;  /* dernier élément + 1 -> légal */
P = A+11;  /* dernier élément + 2 -> illégal */
P = A-1;    /* premier élément - 1 -> illégal */
```

- Soustraction de deux pointeurs

Soient P1 et P2 deux pointeurs qui pointent *dans le même tableau*:

P1-P2 fournit le nombre de composantes comprises entre P1 et P2.

Le résultat de la soustraction **P1-P2** est

- négatif, si P1 précède P2
- zéro, si P1 = P2
- positif, si P2 precède P1
- indéfini, si P1 et P2 ne pointent pas dans le même tableau

Plus généralement, la soustraction de deux pointeurs qui pointent dans le même tableau est équivalente à la soustraction des indices correspondants.

- Comparaison de deux pointeurs

On peut comparer deux pointeurs par **<**, **>**, **<=**, **>=**, **==**, **!=**.

La comparaison de deux pointeurs qui pointent *dans le même tableau* est équivalente à la comparaison des indices correspondants. (Si les pointeurs ne pointent pas dans le même tableau, alors le résultat est donné par leurs positions relatives dans la mémoire).

Exercices

1) Soit P un pointeur qui 'pointe' sur un tableau A:

```
int A[] = {12, 23, 34, 45, 56, 67, 78, 89, 90};
int *P;
P = A;
```

Quelles valeurs ou adresses fournissent ces expressions:

a) ***P+2**
b) ***(P+2)**
c) **P+1**
d) **&A[4]-3**
e) **A+3**
f) **&A[7]-P**
g) **P+(*P-10)**
h) ***(P+*(P+8)-A[7])**

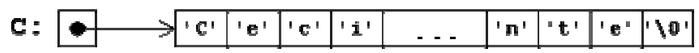
2) Ecrire un programme qui lit un entier X et un tableau A du type **int** au clavier et élimine toutes les occurrences de X dans A en tassant les éléments restants. Le programme utilisera un pointeur P1 pour parcourir le tableau.

3) Ecrire un programme qui range les éléments d'un tableau A du type **int** dans l'ordre inverse.

3.3. Pointeurs et chaînes de caractères

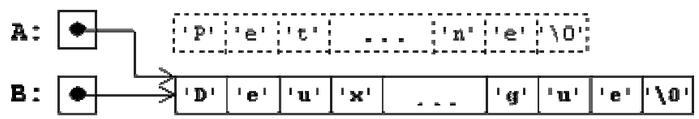
On peut attribuer *l'adresse d'une chaîne de caractères constante* à un pointeur sur **char**:

```
char *C;
C = "Ceci est une chaîne de caractères constante";
```




```
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:



Exercices :

1) Ecrire un programme qui lit deux tableaux d'entiers A et B et leurs dimensions N et M au clavier et qui ajoute les éléments de B à la fin de A. Utiliser deux pointeurs PA et PB pour le transfert et afficher le tableau résultant A.

2) Ecrire de deux façons différentes, un programme qui vérifie sans utiliser une fonction de <*string*>, si une chaîne CH introduite au clavier est un palindrome:

a) en utilisant uniquement le formalisme tableau

b) en utilisant des pointeurs au lieu des indices numériques

3) Ecrire un programme qui lit une chaîne de caractères CH et détermine la longueur de la chaîne à l'aide d'un pointeur P. Le programme n'utilisera pas de variables numériques.

4) Ecrire un programme qui lit une chaîne de caractères CH et détermine le nombre de mots contenus dans la chaîne. Utiliser un pointeur P, une variable logique, la fonction **isspace** et une variable numérique N qui contiendra le nombre des mots.

5) Ecrire un programme qui lit un caractère C et une chaîne de caractères CH au clavier. Ensuite toutes les occurrences de C dans CH seront

éliminées. Le reste des caractères dans CH sera tassé à l'aide d'un pointeur et de la fonction **strcpy**.

3.4. Pointeurs et tableaux à deux dimensions

Soit le tableau M à deux dimensions défini comme suit:

```
int M[4][10] = {{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9},
                {10,11,12,13,14,15,16,17,18,19},
                {20,21,22,23,24,25,26,27,28,29},
                {30,31,32,33,34,35,36,37,38,39}};
```

Le nom du tableau M représente l'adresse du premier élément du tableau et pointe sur le ***tableau*** M[0] qui a la valeur:

{0,1,2,3,4,5,6,7,8,9}.

L'expression (M+1) est l'adresse du deuxième élément du tableau et pointe sur M[1] qui a la valeur:

{10,11,12,13,14,15,16,17,18,19}.

L'arithmétique des pointeurs qui respecte automatiquement les dimensions des éléments conclut logiquement que: **M+I** désigne l'adresse du tableau **M[I]**

Accès aux éléments du tableau

```
int *P;
P = (int *)M;  /* conversion forcée */
```

Dû à la mémorisation ligne par ligne des tableaux à deux dimensions, il nous est maintenant possible traiter M à l'aide du pointeur P comme un tableau unidimensionnel de dimension 4*10.

```
/Calculer la somme de tous les éléments du tableau*/

int *P;
int I, SOM;
P = (int*)M;
SOM = 0;
for (I=0; I<40; I++)
     SOM += *(P+I);
```

Exercices

1) Ecrire un programme qui lit une matrice A de dimensions N et M au clavier, affiche A et sa transposée en utilisant le formalisme pointeur.

2) Ecrire un programme qui lit 5 mots d'une longueur maximale de 50 caractères et les mémorise dans un tableau de chaînes de caractères TABCH. Inverser l'ordre des caractères à l'intérieur des 5 mots à l'aide de deux pointeurs P1 et P2. Afficher les mots.

3.5. Tableaux de pointeurs

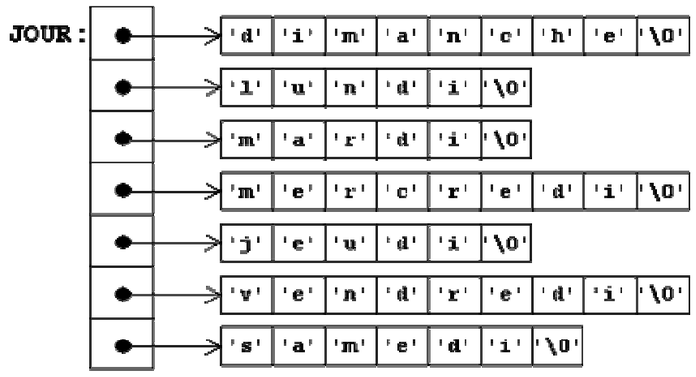
Déclaration d'un tableau de pointeurs

<Type> *<NomTableau>[<N>]
déclare un tableau <NomTableau> de <N> pointeurs sur des données du type <Type>.

Exemple

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

déclare un tableau **JOUR[]** de 7 pointeurs sur **char**. Chacun des pointeurs est initialisé avec l'adresse de l'une des 7 chaînes de caractères.




```
 /*Afficher les 7 jours de la semaine*/
  int I;
  for (I=0; I<7; I++) printf("%s\n", JOUR[I]);

/*Afficher les premières lettres des 7 jours de la semaine*/
  int I;
  for (I=0; I<7; I++) printf("%c\n", *JOUR[I]);

/*Afficher la troisième lettre de chaque jour de la semaine*/
  int I;
  for (I=0; i<7; I++) printf("%c\n",*(JOUR[I]+2));
```

Exercice

Considérez les déclarations de **NOM1** et **NOM2**:

```
char *NOM1[] =    {"Marc",  "Jean-Marie",  "Paul",
                   "François-Xavier",    "Claude" };

char NOM2[][16] = {"Marc",  "Jean-Marie",  "Paul",
                   "François-Xavier",    "Claude" };
```

Représenter graphiquement la mémorisation des deux variables **NOM1** et **NOM2**.

3.6. Allocation dynamique de mémoire

Déclaration statique de données

Dans ce type de déclaration, le nombre d'octets à réserver est déjà connu pendant la compilation.

Exemples

```
float A, B, C;        /* réservation de 12  octets */
short D[10][20];      /* réservation de 400 octets */
char E[] = {"Bonjour !"};
                      /* réservation de 10  octets */
char F[][10] = {"un", "deux", "trois", "quatre"};
                      /* réservation de 40  octets */
```

Allocation dynamique

Nous voulons lire 10 phrases au clavier et mémoriser les phrases en utilisant un tableau de pointeurs sur **char**. Nous déclarons ce tableau de pointeurs par:

```
char *TEXTE[10];
```

Il nous est impossible de prévoir à l'avance le nombre d'octets à réserver pour les phrases elles-mêmes qui seront introduites lors de l'exécution du programme ...

La réservation de la mémoire pour les 10 phrases peut donc seulement se faire *pendant l'exécution du programme*. Nous parlons dans ce cas de l'***allocation dynamique*** de la mémoire.

La fonction malloc et l'opérateur sizeof

La fonction **malloc** de la bibliothèque <*stdlib*> nous aide à localiser et à réserver de la mémoire au cours d'un programme. Elle nous donne accès au tas *(heap)*; c.-à-d. à l'espace en mémoire laissé libre une fois mis en place le DOS, les gestionnaires, les programmes résidents, le programme lui-même et la pile *(stack)*.

malloc(<N>) 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 1

```
char *T ;
T = malloc(4000);
```

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.

Exemple 2

Nous voulons réserver de la mémoire pour X valeurs du type **int**; la valeur de X est lue au clavier:

```
int X;
int *PNum;
printf("Introduire le nombre de valeurs :");
scanf("%d", &X);
PNum = malloc(X*sizeof(int));
```

Exemple 3

Le programme suivant lit 10 phrases au clavier, recherche des blocs de mémoire libres assez grands pour la mémorisation et passe les adresses aux composantes du tableau **TEXTE[]**. S'il n'y a pas assez de mémoire pour une chaîne, le programme affiche un message d'erreur et interrompt le programme avec le code d'erreur -1.

Nous devons utiliser une variable d'aide **INTRO** comme zone intermédiaire (non dynamique). Pour cette raison, la longueur maximale d'une phrase est fixée à 500 caractères.

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

main()
{
 /* Déclarations */
 char INTRO[500];
 char *TEXTE[10];
 int I;
```

```
 /* Traitement */
 for (I=0; I<10; I++)
    {
     gets(INTRO);
     /* Réservation de la mémoire */
     TEXTE[I] = malloc(strlen(INTRO)+1);
     /* S'il y a assez de mémoire, ... */
     if (TEXTE[I])
          /* copier la phrase à l'adresse */
          /* fournie par malloc, ...      */
          strcpy(TEXTE[I], INTRO);
     else
        {
         /* sinon quitter le programme */
         /* après un message d'erreur. */
         printf("ERREUR: Pas assez de mémoire \n");
         exit(-1);
        }
    }
 return 0;
}
```

Exercice

Ecrire un programme qui lit 10 phrases d'une longueur maximale de 200 caractères au clavier et qui les mémorise dans un tableau de pointeurs sur **char** en réservant dynamiquement l'emplacement en mémoire pour les chaînes. Ensuite, l'ordre des phrases est inversé en modifiant les pointeurs et le tableau résultant est affiché.

La fonction free

free(<Pointeur>)
libère le bloc de mémoire désigné par le <Pointeur>; n'a pas d'effet si le pointeur a la valeur zéro(NULL).

Attention !

* La fonction **free** peut aboutir à un désastre si on essaie de libérer de la mémoire qui n'a pas été allouée par **malloc**.

* La fonction **free** ne change pas le contenu du pointeur; il est conseillé d'affecter la valeur zéro au pointeur immédiatement après avoir libéré le bloc de mémoire qui y était attaché.

* Si nous ne libérons pas explicitement la mémoire à l'aide **free**, alors elle est libérée automatiquement à la fin du programme.

Exercices

1) Déclarer dynamiquement un tableau de N entiers (N étant introduits au clavier), saisir le contenu des éléments du tableau.

Afficher la valeur minimale et maximale du tableau.

2) Allocation dynamique d'un tableau de pointeurs :

Reprendre l'exemple de la section 3.5 du chapitre en déclarant le tableau JOUR de manière dynamique.

3) Allocation dynamique d'une matrice :

Ecrire un programme qui permet de saisir une matrice d'entiers de dimensions L et C et d'afficher cette matrice.