

8. Les Pointeurs

Les Pointeurs

Notion de pointeur

Intérêt des pointeurs en C

- La plupart des langages de programmation offrent la possibilité d'accéder aux données dans la mémoire de l'ordinateur à l'aide de **pointeurs**, c.-à-d. à l'aide de variables auxquelles on peut attribuer les **adresses d'autres variables**.
- 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**.
 - De même le traitement de tableaux et de chaînes de caractères dans des fonctions serait impossible sans l'utilisation de pointeurs

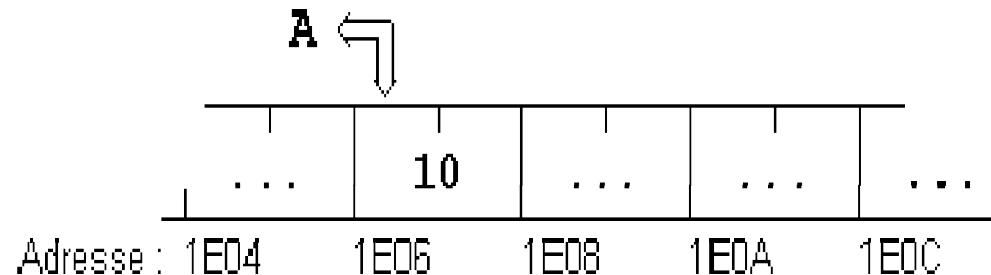
Adressage de variables

- Une variable est caractérisée par:
 - **son adresse**: c.à.d l'adresse mémoire à partir de laquelle l'objet est stocké.
 - **sa valeur**, c.à.d ce qui est stocké à cette adresse.
- Deux modes d'adressage:
 - **Adressage direct**
 - **Adressage indirect**

Adressage direct

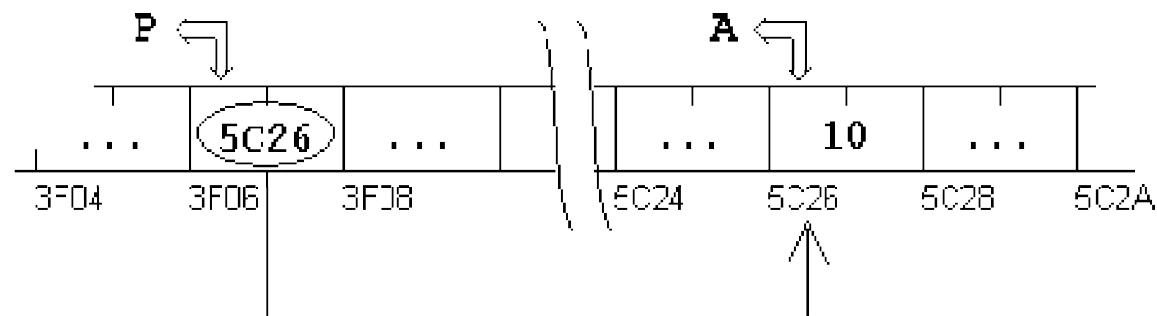
- La valeur d'une variable se trouve à un endroit spécifique dans la mémoire interne de l'ordinateur. **Le nom de la variable** nous permet alors d'accéder *directement* à cette valeur.
- **Adressage direct:** Accès au contenu d'une variable par le nom de la variable.
- **Exemple:**

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



Adressage indirect

- On peut copier l'adresse d'une variable dans une variable spéciale P, appelée **pointeur**. Ensuite, on peut retrouver l'information de la variable A en passant par le pointeur P.
- **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:



Pointeur

- 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.
- Si un pointeur P contient l'adresse d'une variable A, on dit que '**P pointe sur A**'.
- Remarque : Les pointeurs et les noms de variables ont le même rôle: Ils donnent accès à un emplacement dans la mémoire interne de l'ordinateur. Il faut quand même bien faire la différence:
 - Un pointeur est une variable qui peut 'pointer' sur différentes adresses.
 - Le nom d'une variable reste toujours lié à la même adresse.

Déclaration d'un pointeur

<Type> *<NomPointeur>

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

- Exemple:

int *PNUM;

Signifie que

→ **PNUM est du type int*

ou

PNUM est un pointeur sur int

ou

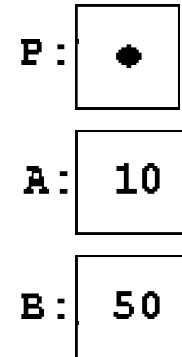
PNUM peut contenir l'adresse d'une variable du type int

Les opérateurs de base(1)

- Lors du travail avec des pointeurs, nous avons besoin:
 - d'un opérateur 'adresse de': **&** pour obtenir l'adresse d'une variable. **&<NomVariable>** fournit l'adresse de la variable <NomVariable>
 - d'un opérateur 'contenu de': ***** pour accéder au contenu d'une adresse. ***<NomPointeur>** désigne le contenu de l'adresse référencée par le pointeur <NomPointeur>
 - d'une syntaxe de déclaration pour pouvoir déclarer un pointeur.

Les opérateurs de base(2)

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



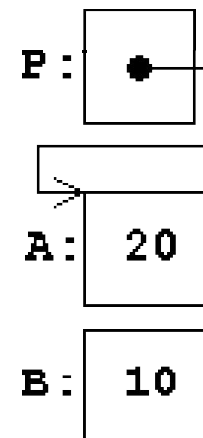
- Après les instructions,

P = &A;

B = *P;

***P = 20;**

- P pointe sur A,
- le contenu de A (référéncé par *P) est affecté à B, et
- le contenu de A (référéncé par *P) est mis à 20.



Les opérateurs de base(3)

Le programme complet effectuant les transformations de l'exemple ci-dessus peut se présenter comme suit:

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

Ou bien

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

Les opérations élémentaires sur un pointeur(1)






*Priorité de * et &*

- Les opérateurs * et & ont la **même priorité que les autres opérateurs unaires** (la négation !, l'incrémentation ++, la décrémentation --). Dans une même expression, les opérateurs unaires *, &, !, ++, -- sont évalués de droite à gauche.
- Si un pointeur P pointe sur une variable X, alors *P peut être utilisé partout où on peut écrire X.

Les opérations élémentaires sur un pointeur(2)

Exemple

Après l'instruction **P = &X;** les expressions suivantes, sont équivalentes:

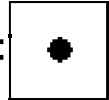
Y = *P+1	Y = X+1	
		
*P = *P+10		X = X+10
		
*P += 2		X += 2
		
++*P		++X
		
(*P)++		X++
		

 les parenthèses sont nécessaires

Les opérations élémentaires sur un pointeur(3)

Le pointeur NUL

- *Seule exception:* La valeur numérique 0 (zéro) est utilisée pour indiquer qu'un pointeur ne pointe 'nulle part'.

P :  **int *P;**
P = 0;

- Finalement, les pointeurs sont aussi des variables et peuvent être utilisés comme telles. Soit P1 et P2 deux pointeurs sur **int**, alors l'affectation

P1 = P2;

copie le contenu de P2 vers P1. P1 pointe alors sur le même objet que P2.

Les opérations élémentaires sur un pointeur(4)

Résumé:

- Après les instructions:

```
int A;  
int *P;  
P = &A;
```

A désigne le contenu de A et **&A** désigne l'adresse de A
P désigne l'adresse de A et ***P** désigne le contenu de A

- En outre:

&P désigne l'adresse du pointeur P

***A est illégal** (puisque A n'est pas un pointeur)

Les Pointeurs

Pointeurs et tableaux

Adressage des composantes d'un tableau(1)

- Le nom d'un tableau représente l'adresse de son premier élément. En d'autres termes:
&tableau[0] et **tableau**
sont une seule et même adresse.
- En simplifiant, nous pouvons retenir que *le nom d'un tableau est un **pointeur constant** sur le premier élément du tableau.*
- **Exemple:** 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];**

Adressage des composantes d'un tableau(2)

- Si **P** pointe sur une composante quelconque d'un tableau, alors **P+1** pointe sur la composante suivante et:
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]

Adressage des composantes d'un tableau(3)

- **Exemple**

Soit A un tableau contenant des éléments de 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.

Adressage des composantes d'un tableau(4)

Différence essentielle entre un pointeur et le nom d'un tableau:

- Un *pointeur* est une variable, donc des opérations comme **P = A** ou **P++** sont permises.
Le *nom d'un tableau* est une constante, donc des opérations comme **A = P** ou **A++** sont impossibles.
- Lors de la première phase de la compilation, toutes les expressions de la forme A[i] sont traduites en *(A+i). En multipliant l'indice i par la grandeur d'une composante, on obtient un indice en octets:

$$\langle \text{indice en octets} \rangle = \langle \text{indice élément} \rangle * \langle \text{grandeur élément} \rangle$$

Cet indice est ajouté à l'adresse du premier élément du tableau pour obtenir l'adresse de la composante i du tableau. Pour le calcul d'une adresse donnée par une adresse plus un indice en octets, on utilise un mode d'adressage spécial connu sous le nom '**adressage indexé**':

$$\langle \text{adresse indexée} \rangle = \langle \text{adresse} \rangle + \langle \text{indice en octets} \rangle$$

Adressage des composantes d'un tableau(5)

Formalisme tableau et formalisme pointeur

- Exemple: 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;
for (J=0,I=0 ; I<10 ; I++)
    if (T[I]>0) {
        POS[J] = T[I];
        J++;
    }
return 0;
}
```

Formalisme pointeur

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

Résumons

- **Les variables et leur utilisation** **int A;** déclare une **variable simple** du type **int**

A désigne *le contenu de A*

&A désigne *l'adresse de A*

int B[]; déclare un **tableau** d'éléments de type **int**

B désigne *l'adresse de la première composante de B.*
(Cette adresse est toujours constante)

B[i] désigne le contenu de la composante *i* du tableau

&B[i] désigne l'adresse de la composante *i* du tableau

en utilisant le formalisme pointeur:

B+i désigne l'adresse de la composante *i* du tableau

***(B+i)** désigne le contenu de la composante *i* du tableau

Résumons

int *P;

déclare un *pointeur* sur des éléments du type **int**.

P peut pointer sur des variables simples du type **int** ou sur les composantes d'un tableau du type **int**.

P

désigne *l'adresse contenue dans P*
(Cette adresse est variable)

***P**

désigne le contenu de l'adresse dans P

Si P pointe dans un tableau, alors

P

P+i

***(P+i)**

désigne l'adresse de la première composante

désigne l'adresse de la i-ième composante derrière P

désigne le contenu de la i-ième composante derrière P

Les Pointeurs

Arithmétique des pointeurs

Arithmétique des pointeurs(1)

- **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]

Arithmétique des pointeurs(2)

- **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]

Arithmétique des pointeurs(3)

- **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.
 - *Seule exception: Il est permis de 'pointer' sur le premier octet derrière un tableau (à condition que cet octet se trouve dans le même segment de mémoire que le tableau). Cette règle, introduite avec le standard ANSI-C, légalise la définition de boucles qui incrémentent le pointeur avant l'évaluation de la condition d'arrêt.*

- **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 */
```

Arithmétique des pointeurs(4)

- **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 précède 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.

Arithmétique des pointeurs(5)

- **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).

Les Pointeurs

Pointeurs et chaînes de caractères

Pointeurs sur char et chaînes de caractères constantes(1)

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.

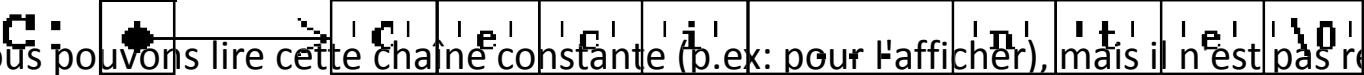
- **Affectation**

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 constante";
```

-  Nous pouvons lire cette chaîne constante (p.ex: pour l'afficher), mais il n'est pas recommandé de la modifier, parce que le résultat d'un programme qui essaie de modifier une chaîne de caractères constante n'est pas prévisible en ANSI-C.

Pointeurs sur char et chaînes de caractères constantes(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 !";
```

Remarque:

- 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 autre chose. La chaîne constante peut être lue, copiée ou affichée, mais **pas modifiée**

A:

'B'	'o'	'n'	'j'	'o'	'u'	'r'	' '	'!'	'\0'
-----	-----	-----	-----	-----	-----	-----	-----	-----	------

B:

•	→	'B'	'o'	'n'	'j'	'o'	'u'	'r'	' '	'!'	'\0'
---	---	-----	-----	-----	-----	-----	-----	-----	-----	-----	------

Pointeurs sur char et chaînes de caractères constantes(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:

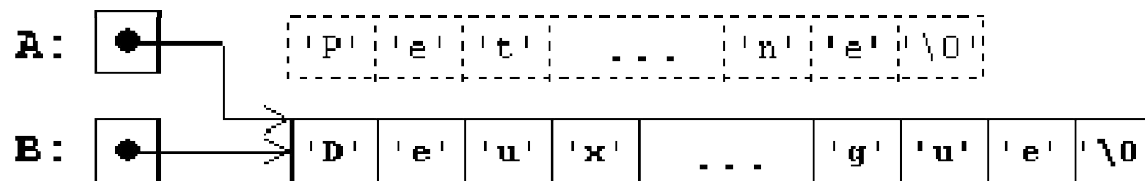
Exemple

```
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:



Pointeurs sur char et chaînes de caractères constantes(4)

- **Remarque:**

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

Exemple

```
char A[45] = "Petite chaîne";  
char B[45] = "Deuxième chaîne un peu plus longue";  
char C[30];  
A = B; /* IMPOSSIBLE -> ERREUR !!! */  
C = "Bonjour !"; /* IMPOSSIBLE -> ERREUR !!! */
```

A:

'P'	'e'	't'	...	'n'	'e'	'\0'
-----	-----	-----	-----	-----	-----	------

B:

'D'	'e'	'u'	x'	...	'g'	'u'	'e'	'\0'
-----	-----	-----	----	-----	-----	-----	-----	------

- Dans cet exemple, nous essayons de copier l'adresse de B dans A, respectivement l'adresse de la chaîne constante dans C. Ces opérations sont impossibles et illégales parce que ***l'adresse représentée par le nom d'un tableau reste toujours constante.***
- Pour changer le contenu d'un tableau, nous devons changer les composantes du tableau l'une après l'autre (p.ex. dans une boucle) ou déléguer cette charge à une fonction de <stdio> ou <string>.

Pointeurs sur char et chaînes de caractères constantes(5)

- ***Conclusions:***

- Utilisons des *tableaux de caractères* pour déclarer les chaînes de caractères que nous voulons modifier.
- Utilisons des *pointeurs sur **char*** pour manipuler des chaînes de caractères constantes (dont le contenu ne change pas).
- Utilisons de préférence des *pointeurs* pour effectuer les manipulations à l'intérieur des tableaux de caractères.