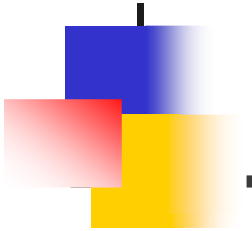


Chapitre 7:
Les Pointeurs

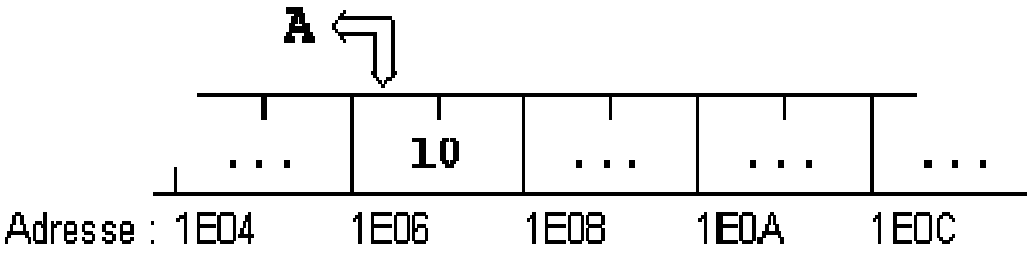
1. Adressage de variables

- Avant de parler de pointeurs, il est indiqué de brièvement passer en revue les deux modes d'adressage principaux
- **1.1 Adressage direct**

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

Exemple:

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



1. Adressage de variables

- **1.2 Adressage indirect**

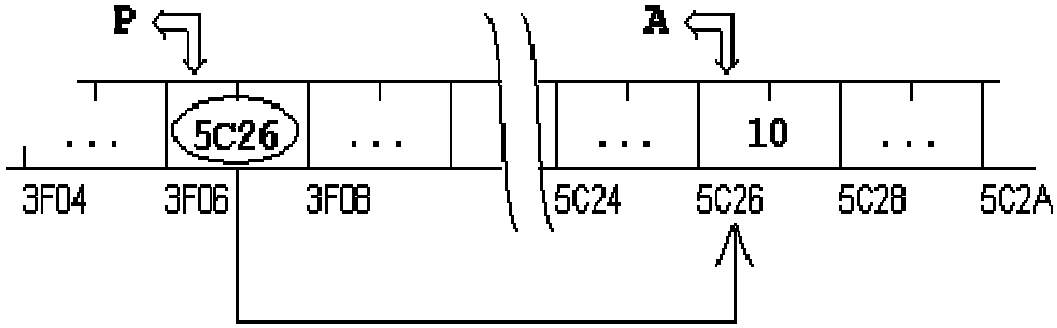
  Si nous ne voulons ou ne pouvons pas utiliser le nom d'une variable A, *nous pouvons copier l'adresse de cette variable dans une variable spéciale P, appelée pointeur*. Ensuite, nous pouvons retrouver l'information de la variable A en passant par le pointeur P.

Adressage indirect: Accès au contenu d'une variable, en passant par un pointeur qui contient l'adresse de la variable.

1. Adressage de variables

- ***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

- ***Définition:***

*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

- ***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.

2.1 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.

- ***L'opérateur*** '« adresse de » **&**

  & NomVariable fournit l'adresse de la variable NomVariable

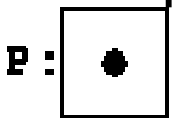
- ***Attention***

  L'opérateur **&** peut seulement être appliqué à des objets qui se trouvent dans la mémoire interne, c.-à-d. à des variables et des tableaux. Il ne peut pas être appliqué à des constantes ou des expressions.

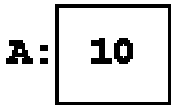
2.1 Les opérateurs de base

- *Représentation schématique*

Soit P un pointeur non initialisé

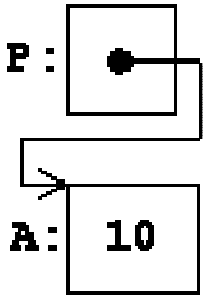


- et A une variable (du même type) contenant la valeur 10 :



Alors l'instruction

P = &A; affecte l'adresse de la variable A à la variable P 'P pointe sur A' par une flèche:



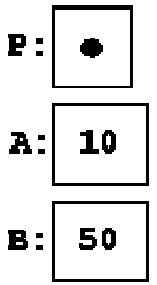
2.1 Les opérateurs de base

- ***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:

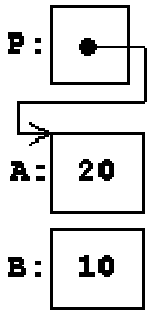


- Après les instructions

  P = &A; P pointe sur A

  B = *P; le contenu de A (référencé par *P) est affecté à B

  ***P = 20;** le contenu de A (référencé par *P) est mis à 20



2.1 Les opérateurs de base

- ***Déclaration d'un pointeur***

Type *NomPointeur

exemple

int *P;

peut être interprétée comme suit:

**P est du type int*

Ou

P est un pointeur sur int

Ou

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

2.1 Les opérateurs de base

```
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;
 }
```

2.2 Les opérations élémentaires sur pointeurs

- ***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
- ***Exemple***

Après l'instruction **P = &X;**

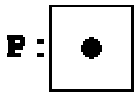
les expressions suivantes, sont équivalentes:

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

2.2 Les opérations élémentaires sur pointeurs

- **pointeur NUL**

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




```
int *P;
P = 0;
```

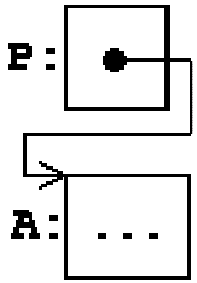
Finalement, les pointeurs sont aussi des variables et peuvent être utilisés comme telles. Soit P1 et P2 deux pointeurs sur **int**, alors l'affectation

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

2.2 Les opérations élémentaires sur pointeurs

- Après les instructions:

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



A ⟶ désigne le contenu de A
&A ⟶ désigne l'adresse de A
P ⟶ désigne l'adresse de A
***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)

Exercice

```
main() {
        int A = 1;
        int B = 2;
        int C = 3;
        int *P1, *P2;
      P1=&A;
      P2=&C;
      *P1=(*P2)++;
       P1=P2;
       P2=&B;
       *P1-=*P2;
       ++*P2;
       *P1*=*P2;
        A=++*P2**P1;
        P1=&A;
       *P2=*P1/=*P2;
          return 0;  }
```

Exercice -suite

- Copiez le tableau suivant et complétez-le pour chaque instruction du programme ci-dessus.

	A	B	C	P1	P2
Init.	1	2	3	/	/
P1=&A	1	2	3	&A	/
P2=&C					
***P1=(*P2)++**					
P1=P2					
P2=&B					
***P1-=*P2**					
++*P2					
***P1*=*P2**					
A=++*P2P1**					
P1=&A					
***P2=*P1/=*P2**					

Solution

	A	B	C	P1	P2
Init.	1	2	3	/	/
P1=&A	1	2	3	&A	/
P2=&C	1	2	3	&A	&C
*P1=(*P2)++	3	2	4	&A	&C
P1=P2	3	2	4	&C	&C
P2=&B	3	2	4	&C	&B
*P1-=*P2	3	2	2	&C	&B
++*P2	3	3	2	&C	&B
P1=*P2	3	3	6	&C	&B
A=++*P2**P1	24	4	6	&C	&B
P1=&A	24	4	6	&A	&B
*P2=*P1/=*P2	6	6	6	&A	&B

3. Pointeurs et tableaux

- Adressage des composantes d'un tableau

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

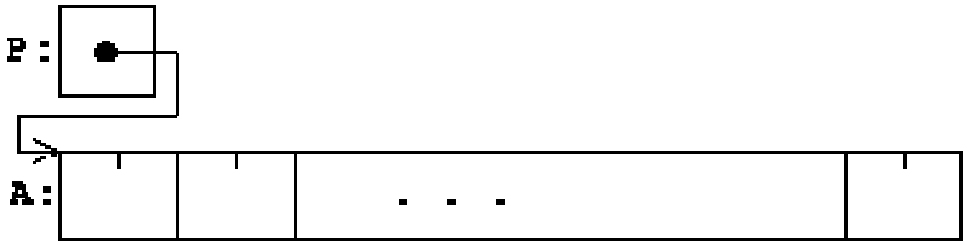
&tableau[0] et **tableau**

- ***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];**



3. Pointeurs et tableaux

- 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
- **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]

3. Pointeurs et tableaux

- ***Exemple***
- 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.

3. Pointeurs et tableaux

- **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]**

- Lors de la première phase de la compilation, toutes les expressions de la forme A[i] sont traduites en *(A+i).

3. Pointeurs et tableaux

- ***Résumons*** 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]**

3. Pointeurs et tableaux

- ***Formalisme tableau et formalisme pointeur***

A l'aide de ce bagage, il nous est facile de 'traduire' un programme écrit à l'aide du *'formalisme tableau'* dans un programme employant le *'formalisme pointeur'*

Exemple

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

3. Pointeurs et tableaux

- ***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;}
```

3. Pointeurs et tableaux

- Nous pouvons remplacer systématiquement 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;}
```

Exercice 2

- 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])

Solution 2

a) *P+2 => la valeur 14

b) *(P+2) => la valeur 34

c) &P+1 => l'adresse du pointeur derrière le pointeur P

d) &A[4]-3 => l'adresse de la composante A[1]

e) A+3 => l'adresse de la composante A[3]

f) &A[7]-P => la valeur (indice) 7

g) P+(*P-10) => l'adresse de la composante A[2]

h) *(P+*(P+8)-A[7]) => la valeur 23

Exercice 3

- 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 les pointeurs P1 et P2 pour parcourir le tableau.

```
#include <stdio.h>
main() {        /* Déclarations */
  int A[50]; N;  X;  *P1, *P2;                /* pointeurs d'aide */
 /* Saisie des données */
    printf("Dimension du tableau (max.50) : ");
   scanf("%d", &N );

   for (P1=A; P1<A+N; P1++)
     {
        printf("Elément %d : ", P1-A);
        scanf("%d", P1);
     }
   printf("Introduire l'élément X à éliminer du tableau : ");
   scanf("%d", &X );
 /* Affichage du tableau */
  for (P1=A; P1<A+N; P1++)
    printf("%d ", *P1); printf("\n");
 /* Effacer toutes les occurrences de X et comprimer : */
  /* Copier tous les éléments de P1 vers P2 et augmenter */
 /* P2 pour tous les éléments différents de X. */
for (P1=P2=A; P1<A+N; P1++)
   {
      *P2 = *P1;
      if (*P2 != X)
        P2++;
 }
/* Nouvelle dimension de A */
N = P2-A;
/* Edition du résultat */
for (P1=A; P1<A+N; P1++)
      printf("%d ", *P1);
}
```

Exercice 4

- Ecrire un programme qui range les éléments d'un tableau A du type **int** dans l'ordre inverse. Le programme utilisera des pointeurs P1 et P2 et une variable numérique AIDE pour la permutation des éléments.

```
#include <stdio.h>
main() {      /* Déclarations */
  int A[50];          /* tableau donné */
  int N;              /* dimension du tableau */
  int AIDE;           /* pour la permutation */
  int *P1, *P2;      /* pointeurs d'aide */
/* Saisie des données */
   printf("Dimension du tableau (max.50) : ");
  scanf("%d", &N );
  for (P1=A; P1<A+N; P1++)
     {
        printf("Elément %d : ", P1-A);
        scanf("%d", P1);
     }
/* Affichage du tableau */
  for (P1=A; P1<A+N; P1++)
     printf("%d ", *P1);
   printf("\n");
/* Inverser la tableau */
   for (P1=A,P2=A+(N-1); P1<P2; P1++,P2--)
     {
        AIDE = *P1;
        *P1 = *P2;
        *P2 = AIDE;
     }
/* Edition du résultat */
for (P1=A; P1<A+N; P1++)
      printf("%d ", *P1);
printf("\n");
}
```