UNIVERSITÉ PRIVÉE DE MARRAKECH

UNIVERSITÉ RECONNUE PAR L'ÉTAT

جامعة معترف بها من طرف الدولة

Module: Structures de données

Chapitre 2: Les pointeurs

F. OUAKASSE

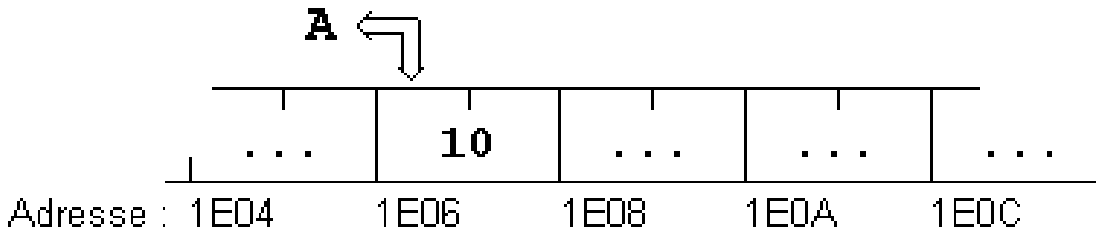
Pointeurs

Adressage direct

- Dans la programmation, nous utilisons des variables pour stocker des informations. 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:

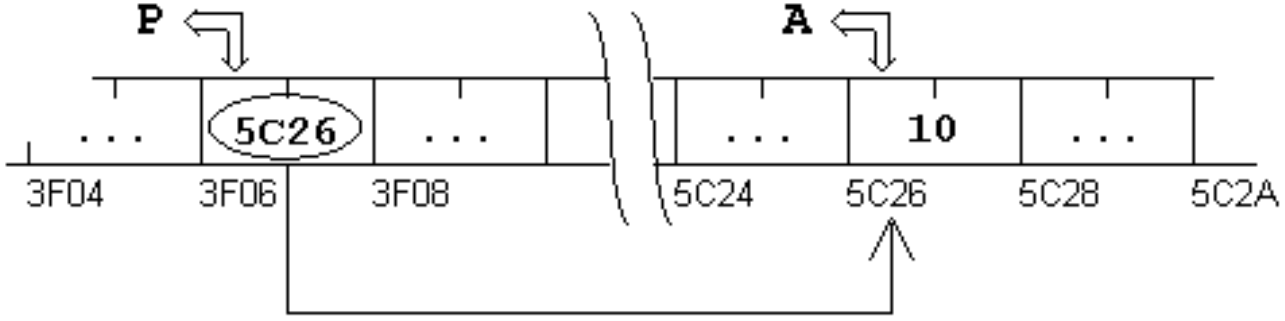
```
short A;
A = 10;
```



Pointeurs

Adressage indirect

- Si on ne veut 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.
- ***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·



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.
- 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 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.

Pointeurs

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.

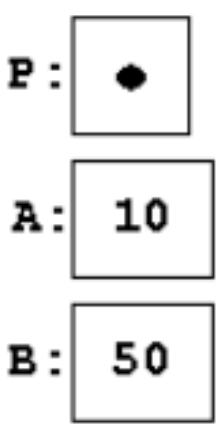
L'opérateur 'adresse de' : &

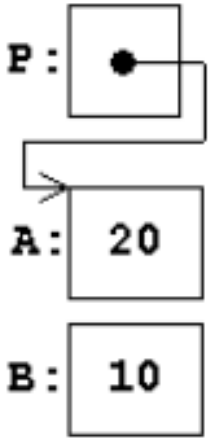
- **&<NomVariable> fournit l'adresse de la variable <NomVariable>**
  L'opérateur **&** est déjà utilisé par la fonction **scanf,** qui a besoin de l'adresse de ses arguments pour pouvoir leur attribuer de nouvelles valeurs.
- ***Exemple***
- **int N;**
- **printf("Entrez un nombre entier : ");**
- **scanf("%d", &N);**

Pointeurs

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

Pointeurs

- En travaillant avec des pointeurs, nous devons observer les règles suivantes:
- ***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.

Pointeurs

- ***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++
```

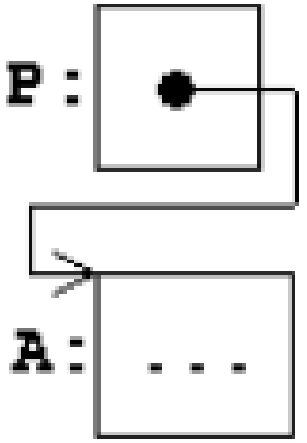
Dans le dernier cas, les parentheses sont nécessaires:

- Comme les opérateurs unaires * et ++ sont évalués *de droite à gauche*, sans les parenthèses le *pointeur* P serait incrémenté, *non pas l'objet* sur lequel P pointe.
- On peut uniquement affecter des adresses à un pointeur.

Pointeurs

Résumé

- 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)

Pointeurs

Exercice: à vérifier

	A	B	C	P1	P2
Init.	1	2	3	/	/
P1=&A	1	2	3	&A	/
P2=&C	1	2	3	&A	&C
***P1=(*P2)++**	4	2	3	&A	&C
P1=P2	4	2	3	&C	&C
P2=&B	4	2	3	&C	&B
***P1- =*P2**	4	2	1	&C	&B
++*P2	4	3	2	&C	&B
***P1*=*P2**	4	3	6	&C	&B
A=++*P2P1**	19	3	6	&C	&B
P1=&A	19	3	6	&A	&B
***P2=*P1/=*P2**	19	6	6	&A	&B

Pointeurs

- Tableau pointeur

- 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]`

Pointeurs

Tableau dynamique

Déclaration d'un pointeur :

Type * ptr; déclare un pointeur sur des données du type Type.

Allocation dynamique :

L'allocation dynamique de mémoire pour un tableau de nb éléments de type Type se fait à l'aide de **malloc:**

```
Type *ptr = (Type *)malloc(nb*sizeof(Type));
```

La libération de mémoire se fait à l'aide de **free** : **free**(ptr);

Accès aux éléments : On accède à l'élément d'indice i du tableau par: ptr[i] ou *(ptr + i)

Et on accède à son adresse par : &ptr[i] ou (ptr + i)

- Résumé

`int B[];`
déclare un ***tableau*** d'éléments du 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

`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`	désigne l'adresse de la première composante
`P+i`	désigne l'adresse de la i-ième composante derrière P
`*(P+i)`	désigne le contenu de la i-ième composante derrière P

Pointeurs

Déclaration d'une chaîne de caractères

On peut déclarer une chaîne de caractères comme n'importe quel tableau mais en prévoyant une place pour le '\0'.

Version allocation statique :

```
char chaine[100];
```

Version allocation dynamique :

```
char *chaine;
chaine = (char *)malloc(100*sizeof(char));
```