Intérêt des pointeurs

esprit
Se former autrement

Utilité des pointeurs

- On peut accéder aux données en mémoire à l'aide de pointeurs i.e. des variables pouvant contenir des adresses d'autres variables.

- Les pointeurs sont le seul moyen de changer le contenu de variables déclarées dans d'autres fonctions.

- Les pointeurs nous permettent d'écrire des programmes plus compacts et plus efficaces.

Les différents types d'adressage

Adressage direct

• C’est l’adressage standard que nous avons utilisé jusqu’à présent.

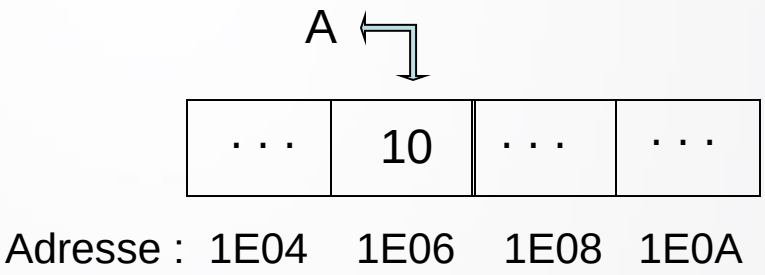
• Dans la programmation, on utilise des variables pour stocker des informations.

• La valeur d’une variable se trouve à un endroit spécifique dans la mémoire de l’ordinateur.

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

Exemple:

```
int A;
A = 10;
```



• Le nom de la variable A nous permet d’accéder directement au contenu de l’adresse 1E06.

Adressage indirect

• Le contenu d'une variable est obtenu en passant par une variable **P** qui contient l'adresse de la variable **A**.

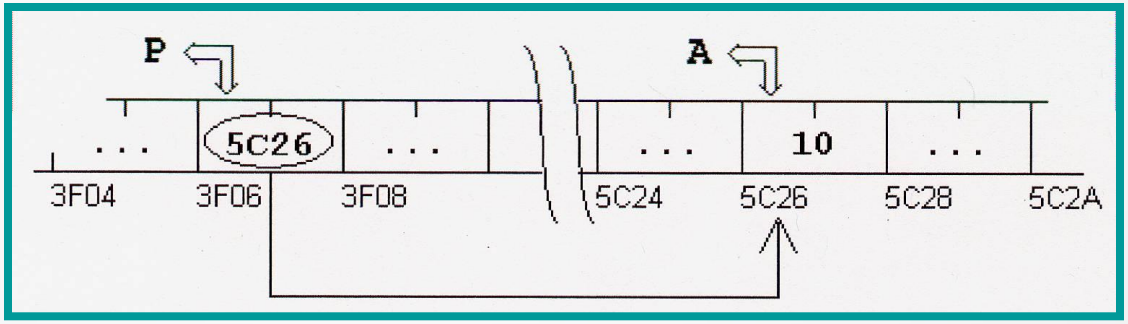
• Dans ce cas précis, P est une variable d'un type particulier que l'on appelle un **pointeur**.

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



Définition

Définition

- Un pointeur est une **variable** spéciale qui peut contenir l'**adresse** d'une autre variable.
- Chaque pointeur est **limité** à un type de données.
- Si un pointeur P contient l'adresse d'une variable A, on dit que P **pointe** sur A.

Définition

• Les pointeurs et les noms de variables ont le même rôle: ils donnent accès à un emplacement en mémoire.

• Par contre, un pointeur peut contenir différentes adresses mais le nom d’une variable reste toujours lié à la même adresse.

Syntaxe

Déclaration

Syntaxe:

<type> *P;

- D'après la déclaration, P est un pointeur qui pourra recevoir des adresses de variables qui auront uniquement le type <type> .
- Un pointeur sur int ne peut pas contenir l'adresse d'une variable de type float.

Exemple :

int* pNombre;

- pNombre désigne une variable pointeur pouvant contenir uniquement l'adresse d'une variable de type int.

- Bonne pratique de programmation : choisir des noms de variable appropriés: (Ex. : pNombre, NombrePtr).

Manipulation d’un pointeur

Initialisation d'un pointeur

- Le symbole **NULL** permet d'initialiser un pointeur qui ne pointe sur rien. Cette valeur NULL peut être affectée à tout pointeur quel que soit le type.
- Le symbole NULL est défini dans la librairie stdlib.h.

- **Exemple:**

```
int *pNombre=NULL;
```

Dès qu'on déclare un pointeur, il est préférable de l'initialiser à NULL.

Attribuer une valeur à un pointeur

- Pour attribuer une valeur à un pointeur, il faut le faire pointer sur une variable précise.
- **Exemple:**

```
int A;
int* p1 = NULL;
int* p2 = NULL;
p1=&A;     /* p1 contient l'adresse de A */
p2=p1;     /* copie le contenu de p1 dans p2 → p2 et p1 pointent sur la même variable A*/
```

- L'opérateur & permet d'obtenir l'adresse d'une variable.
- L'opérateur & ne peut pas être appliqué à des constantes ou des expressions.
- p1 et p2 désignent une variable pointeur initialisée à l'adresse de la variable A de type int.

Accès au contenu d'un pointeur

- L'opérateur *

- Pour avoir accès au contenu d'un pointeur, on utilise l'opérateur * suivi du nom du pointeur.
- **Exemple:**

```
int *pNombre;
int n=20;
pNombre=&n; /* pNombre pointe sur l'entier n
                qui contient la valeur 20 */
printf (" %d ", *pNombre); /*affiche la valeur 20 */
*pNombre=40;
printf(" %d ", *pNombre); /*affiche la valeur 40 */
printf(" %d ", n); /*affiche la valeur 40 */
```

Priorité des opérateurs

- Les opérateurs ***** et **&** ont la même priorité que les autres opérateurs unaires **(!, ++, --).**
- Si les parenthèses ne sont pas utilisées, les expressions sont évaluées de droite à gauche.

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++
```