ANNÉE SCOLAIRE 2023/2024

COURS D'ALGORITHMIQUE

Pape Abdoulaye **BARRO**
Docteur en Informatique et Télécommunications
Spécialiste en Télémétrie & Systèmes Intelligents

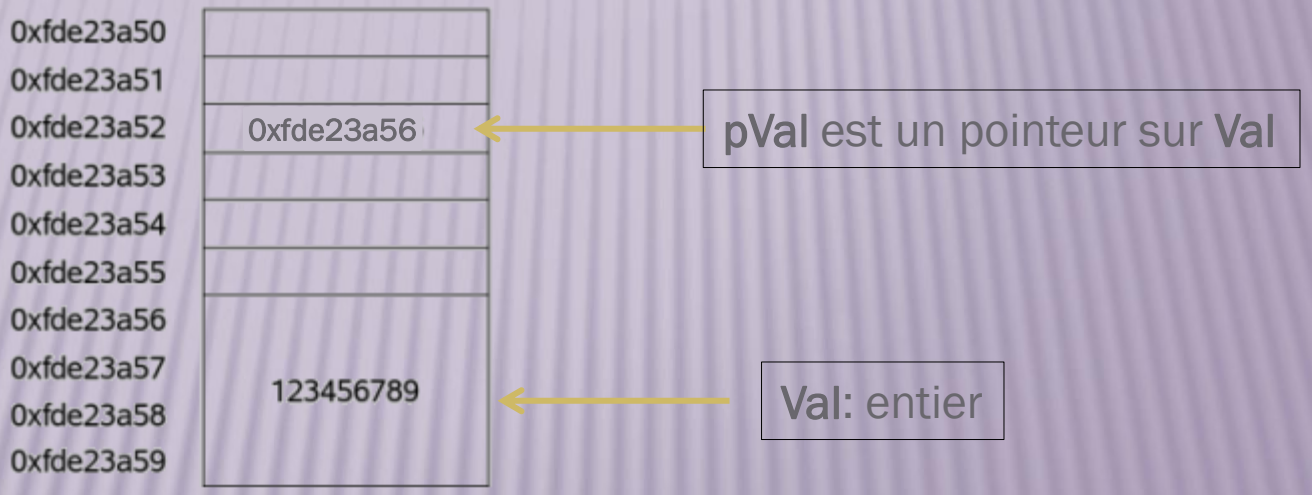
POINTEURS

POINTEURS
PRINCIPE ET DÉFINITION

- **En algorithmique**, il peut être utile de trouver des moyens de manipuler directement ou indirectement des adresses de variable.
- Dans les langages supportant les manipulations d'adresses mémoire, il est courant de manipuler ces adresses au travers de variables particulières qui ne contiennent non pas une donnée, mais une adresse mémoire. Ce sont des **pointeurs.**
- **Un pointeur est une variable qui contient l'adresse d'une autre variable.**
  - En affichant le contenu d'une variable, on obtient l'adresse de la variable pointée.
  - Par contre, la variable pointée contient la valeur associée à cette dernière.

POINTEURS
PRINCIPE ET DÉFINITION

- Un **pointeur** est donc une variable. Il doit donc être déclaré, dispose lui-même sa propre adresse en mémoire et se voit définir un type.
- Le **type d'un pointeur** ne décrit pas ce qu'il contient mais le type de la variable qu'il pointe.
- Un pointeur sur une variable de type entier devrait donc être déclaré avec un type entier.



- Si un pointeur P contient l'adresse d'une variable A, on dit que **'P pointe sur A'**.
- Il se peut qu'un pointeur n'ait pas encore reçu d'adresse et donc pointe sur nille part. c'est embêtant car c'est le plantage assuré si vous tentez de l'utiliser. Pour éviter ce problème, vous lui affecterez une valeur générique, qui ne représente rien, mais qui pourra cependant être testée. C'est la valeur **NIL** (Not Identidied Link).

POINTEURS
DÉCLARER ET UTILISER

- Les pointeurs et les noms de variables ont donc le même rôle: ils donnent accès à un espace mémoire. Il faut quand même faire la différence:
  - Un pointeur peut 'pointer' sur différentes adresses.
  - Le nom d'une variable reste toujours lié à la même adresse.
- Déclaration
  - **Variable pointeur : ^TypePointé**
  - **Remarque :** Le symsbole ↑ est souvent utilisé à la place de ^
  - Un pointeur ne pourra contenir que des adresses mémoire de type **TypePointé**
  - **Exemple :** Variable pointeurEntier : ^entier

POINTEURS
DÉCLARER ET UTILISER

- Affectation
  - **pointeur ← adresse(variablePointee)**
    - La fonction **adresse** renvoie l'adresse mémoire d'une variable
    - La variable pointée et le pointeur doivent être de même type
- **Manipulation**
  - **Pointeur^← valeur**
  - **Lire(pointeur^)**
  - **Ecrire(pointeur^)**

Quelle différence entre ?
- ✓ **adresse(pointeur)**
- ✓ **pointeur**
- ✓ **pointeur^**

POINTEURS

DÉCLARER ET UTILISER > EXEMPLE

- **Exemple:**

```
Algorithme pointeur_sur_type_pointe
Variable
        txt: chaine de caractére
        ptxt:^chaine de caractére
        cpt: entier
        pcpt: ^entier
Début
        txt← "Hello World"
        ptxt← adresse(txt)
        cpt← 10
        pcpt← adresse(cpt)
        Ecrire(ptxt^)
        ptxt^ ← "Salut tout le monde"
        Ecrire(txt)
        pcpt^ ← 20
        cpt ← cpt+1
        Ecrire(cpt)
  Fin
```

POINTEURS
ALLOCATION DYNAMIQUE

- Jusqu'ici nous affectons aux pointeurs l'adresse d'une variable existant. Il existe une possibilité de réserver un emplacement mémoire pour une donnée pointée directement: **C'est le principe de l'allocation dynamique de mémoire.**
- La syntaxe est la suivante:

  pointeur←nouveau type
  - Ici le type doit bien être celui de la valeur qui sera contenue à l'emplacement mémoire réservé.
  - Le pointeur recevra alors l'adresse mémoire de la zone réservée.
  - S'il n'y a plus de mémoire disponible, il reçoit la valeur NIL.

POINTEURS

ALLOCATION DYNAMIQUE>EXEMPLE

- **Exemple:**

```
Algorithme alloc
Variable
    pt: ^entier
Début
    pt← nouveau Entier
    pt^← 12345
    Ecrire(pt^)
 Fin
```

POINTEURS

ALLOCATION DYNAMIQUE>LIBÉRATION

- Dans la plupart des langages disposant de pointeurs, il est possible de **préciser la taille de la mémoire allouée**, par exemple allouer un espace pour dix entiers. Dans ce cas, c'est l'équivalent d'un tableau d'entiers et l'adresse retournée sera celle du premier entier. **Ajouter 1 au pointeur décalera celui-ci d'un élément.** Cette syntaxe n'est pas utilisée en **algorithmique** où on préfère allouer la mémoire élément par élément, quitte à les chaîner ensuite.
- Dès que le ou les pointeurs ne sont plus utiles, on libère la mémoire associée. Pour ce faire; on doit utiliser la syntaxe suivante:

  Libérer pointeur
  - La **zone mémoire** sur laquelle le pointeur pointait sera alors libérée. Elle sera donc utile à d'autre fin.
  - **Désallouer** un pointeur ne signifie nullement qu'il est vide. Ce pointeur peut arbitrairement pointer sur une zone éventuellement réaffectée à autre chose et peut donc entrainer des complications. **Le mieux est de replacer une valeur NIL après la libération et de penser à tester le pointeur avant de l'utiliser.**

POINTEURS

ALLOCATION DYNAMIQUE>LIBÉRATION > EXEMPLE

- **Exemple:**

```
Algorithme libere
Variable
    pt: ^entier
Début
    pt← nouveau Entier
    { Instructions}
    …
    Libérer pt
    pt ← NIL
 Fin
```

POINTEURS
CAS PRATIQUES N°6

Application 26:

Ecrivez un programme déclarant une variable i de type entier et une variable p de type pointeur sur entier. Affichez les dix premiers nombres entiers en :

- n'incrémentant que i
- n'affichant que p^

Application 27:

Même exercice en

- n'incrémentant que p^
- n'affichant que i

Application 28:

Ecrire un programme qui effectue les traitements suivants. Après chaque traitement et pour chaque pointeur ou valeur, afficher le contenu du pointeur, son adresse, la valeur pointée et l'adresse de la valeur pointée.

– déclarer un pointeur p sans l'initialiser et un pointeur q initialisé à NIL ;
– le pointeur p pointe sur une valeur entière v initialisée à 10 ;
– à l'aide du pointeur, modifier cette valeur entière v "à partir du clavier" ;
– utiliser seulement le pointeur p pour initialiser une valeur entière w à la valeur de v ;
– utiliser seulement un pointeur r qui pointe vers p pour modifier la valeur entière v "à partir du clavier"

Application 29:

Déclarer un tableau de réel et un pointeur sur son premier élément. Parcourir les éléments du tableau à l'aide de ce pointeur. Afficher les valeurs du tableau et leurs adresses, en utilisant le tableau et le pointeur.

Application 30:

Ecrire un programme qui place dans un tableau T les N premiers nombres impairs, puis qui affiche le tableau. Vous accèderez à l'élément d'indice i de t avec l'expression (t + i)^.

Affaires à suivre