Université Abdelmalek Essaâdi
Ecole nationale des sciences appliquées d'Al Hoceima
(ENSAH)

Algorithmique

Préparé et présenté par
Mr. Ouazzani Chahdi

Année universitaire: 2018/2019

Les pointeurs et la gestion dynamique de la mémoire

1-Notion d'adresse mémoire

- Toute variable manipulée dans un programme est stockée quelque part dans la mémoire.
- Au moment de sa déclaration, l'ordinateur lui réserve un emplacement mémoire dont la taille dépend de son type.
- La déclaration d'une variable se réalise en précisant l'identificateur et le type de celui-ci.
- Cet identificateur sert à identifier cette variable le long du programme d'une façon unique.
- Mais au niveau de la mémoire, comment peut-on identifier un objet d'une façon unique ?
- C'est en connaissant l'adresse de son emplacement mémoire.

- Pour cela, le compilateur se charge de faire le lien entre l'identificateur d'un objet et l'adresse de son emplacement mémoire.
- Comme ça, on peut connaître et manipuler l'adresse d'un objet à partir de son identificateur.
- En algorithmique, pour connaître l'adresse mémoire d'un objet, on utilise l'opérateur **@** suivant la syntaxe : **`@NomIdentificateur`**.

Variable	Adresse	Valeur
n	0120303	3
x	0120312	13.2

Adresse	Valeur
0120303	3
0120312	13.2

Représentation au niveau de la mémoire

Exemple

Ecrire un algorithme **Adresse** qui déclare trois variables **n**, **x** et **c** respectivement de type Entier, Réel, Caractère, les initialise et ensuite affiche leurs contenus et leurs adresses.

```
Algorithme Adresse
Variable n : Entier
         x : Réel
         c : Caractère
Début
  n ← 10
  x ← 0.5
  c ← '?'
  Ecrire("n = ", n, "@n = ", @n)
  Ecrire("x = ", x, "@x = ", @x)
  Ecrire("c = ", c, "@c = ", @c)
Fin
```


- L'adresse mémoire représente aussi une valeur entière, cette valeur est représentée sous forme binaire ou hexadécimale dont la taille est toujours fixe et ne dépend pas de l'objet référencé.
- Dans l'exemple précédent, **@n**, **@x** et **@c** représentent des valeurs entières de même taille.
- La taille d'une adresse mémoire dépend de la taille de la mémoire elle-même.

Exemple

Si on dispose d'une mémoire de 256 case, alors on a 2^8 cases à adresser, alors la taille des adresses mémoire sera de 1 octet.

2-Notion de pointeur

- Pour pouvoir manipuler un objet à partir de son adresse mémoire, nous somme amené à enregistrer celle-ci.
- Pour cela, on doit disposer d'un moyen qui permet de faire cela.
- Tout simplement, on utilise une variable dédiée dont le contenu ne peut être qu'une adresse mémoire d'un objet donné.
- Cette variable est appelé ***pointeur***.
- Alors, un pointeur est une variable qui contient l'adresse d'une autre variable.

- P est un pointeur qui contient la valeur **@V**.

7

2.1- Déclaration

Syntaxe :

`Variable NomPointeur : ^TypeVraiablePointee`

- Avec cette syntaxe, on déclare un pointeur identifié par **NomPointeur**, qui pointe vers une variable de type **TypeVraiablePointee**.
- Un pointeur est typé, c'est-à-dire, il est lié au type de l'objet vers lequel il pointe.

Exemple :

Variable p1 : ^Entier : déclaration d'un pointeur vers des variables de type **Entier**.

p2 : ^Réel : déclaration d'un pointeur vers des variables de type **Réel**.

- Comme la taille des adresses mémoire est fixe, alors la taille des pointeurs est aussi fixe.

8

2.2- Initialisation

- L'initialisation d'un pointeur se fait comme si on initialise une variable ordinaire, sauf ici on affecte au pointeur une adresse mémoire.
- L'initialisation peut être faite par trois façons :
  - **`NomPonteur ← AdresseMemoire`**
  - **`NomPonteur ← @Identificateur`**
  - **`NomPointeur1 ← NomPointeur2`**
- On peut initialiser un pointeur par une valeur nulle en utilisant le mot clé **NULL** :
  - **`NomPointeur ← NULL`**
- Généralement, la valeur NULL, représente une adresse mémoire de valeur 0.

9

Exemple

```
Algorithme Pointeur
Variable n : Entier
         x : Réel
         p, p1 : ^Entier
         p2 : ^Réel
Début
  n ← 10
  x ← 1.5
  p ← 1234575
  p1 ← @n
  p2 ← @x
  Ecrire("L'adresse de n est ", p1)
  Ecrire("L'adresse de x est ", p2)
  Ecrire("La valeur de p est ", p)
Fin
```

10

2.3- Contenu d'un pointeur

- Le contenu d'un pointeur est le contenu de l'emplacement mémoire pointé par celui-ci.
- Pour accéder au contenu d'un pointeur, on utilise la syntaxe : **`NomPointeur^`**.

Exemple

```
Algorithme Contenu_Pointeur
Variable n : Entier
         p : ^Entier
Début
n ← 10
p ← @n
Ecrire("La valeur de n est ", p^)
Fin
```

- Cet algorithme affiche à l'écran : ***La valeur de n est 10***

11

2.3- Opération sur les pointeurs

- La valeur d'un pointeur est de type entier, alors on peut appliquer aux pointeurs des opérations d'affectation, des opérations arithmétiques ou de comparaison.

2.3.1-Les opérations arithmétiques

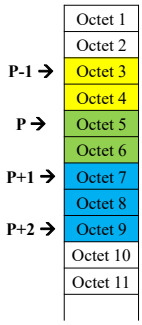
Seules les opérations suivantes ont un sens :

- Addition et de soustraction d'un entier à un pointeur;
- Différence de deux pointeurs de même type.

- ***Addition et soustraction***
  - L'addition (soustraction) d'un entier à un pointeur ne s'interprète pas comme une simple addition car, cet entier représente le nombre d'unité à ajouter et pas un simple scalaire.
  - Cette unité dépend du type de pointeur, par exemple, si on a un pointeur p de type X, et on suppose que la taille du type X en mémoire est 2 octets, alors l'instruction p+1 incrémente p de 2.

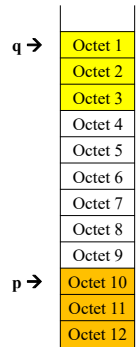
12

- P est un pointeur de type X, et X et un type dont la taille est 2 octets.



13

- ***Différence***
  - Si **p** et **q** sont deux pointeurs de même type X de taille ***n*** octets, alors l'expression **p-q** désigne le nombre d'éléments de taille n situés entre **p** et **q**.
  - Si **n** égale à 3 octets, alors le schéma ci-contre montre que l'expression **p-q** va fournir la valeur 3, car on a **p** pointe sur l'élément 4, et **q** pointe sur l'élément 1.
  - On peut utiliser aussi le résultat de la différence pour savoir :
    - Si p précède q ou l'inverse;
    - Si p égal à q.



14

2.3.2-Les opérations de comparaison

- Pour que la comparaison soit possible, il faut que les deux pointeurs soient de même type.
- Dans ce cas, on peut utiliser les opérateurs de comparaison `<, >, <=, >=, =, <>`.
- La valeur **NULL**, peut être comparer à toutes les variables pointeurs quelque soit leurs types.
- Les tests entre pointeurs ne permet de savoir que :
  - Si deux pointeurs pointent sur la même variable, c'est-à-dire même emplacement mémoire.
  - Si une variable pointeur pointe sur NULL.
  - Si un pointeur P1 précède un pointeur P2

2.3.3-L' opération d'affectation

Comme un pointeur est une variable, alors on peut utiliser l'opérateur d'affectation à l'instar des autres variables.

15

2.4- Pointeur vers un pointeur (double pointeurs)

- Un pointeur vers un pointeur est une variable pointeur qui contient l'adresse d'un autre pointeur :

Syntaxe :

```
Variable NomDoublePointeur : ^^Type
```

- Ce type de pointeur est utilisé pour pouvoir manipuler un pointeur à partir de son adresse.

Exemple

Ecrire un algorithme **Double_Pointeur** qui déclare et initialise un pointeur vers un pointeur de type Entier.

16

```
Algorithme Double_Pointeur
Variable P1 : ^^Entier
         P2 : ^Entier
         n : Entier
Début
  n ← 10
  P2 ← @n
  P1 ← @P2
  Ecrire("La valeur de n est ", n)
  Ecrire("L'adresse de P2 est ", @P2)
  Ecrire("Le contenu de P1 est ", P1^)
Fin
```

17

2.5- Tableau de pointeurs

- La déclaration d'un tableau de pointeur se fait comme une déclaration d'un tableau ordinaire, sauf ici on a un tableau d'adresses mémoire.

Syntaxe :

```
Tableau NomTableau[N] : ^Type
Tableau NomTableau[N, M] : ^Type
```

- Ce type de tableau est utilisé par exemple pour stocker les adresses d'un ensemble d'objets reliés par des relations particulières.

Exemple

Ecrire un algorithme **Tableau_Pointeurs** qui déclare et initialise un tableau à une seule dimension de 3 pointeurs de type entier. Ensuite affiche son contenu.

18

```
Algorithme Tableau_Pointeur
Variable i : Entier
         n1, n2, n3 : Entier
Tableau T[3] : ^Entier
Début
  n1 ← 1
  n2 ← 2
  n3 ← 3
  T[0] ← @n1
  T[1] ← @n2
  T[2] ← @n3
  Pour i allant de 0 Jusqu'à 2 Faire
    Ecrire(T[i]^, " ")
  FinPour
Fin
```

19

3-Gestion dynamique de la mémoire

- Chaque variable utilisée dans un programme occupe un certain nombre d'octets en mémoire.
- La réservation de l'espace mémoire nécessaire pour une variable s'effectue au moment de la déclaration.
- Donc le nombre d'octets à réserver est connu au moment de la compilation.
- On parle ici d'**allocation statique** de la mémoire.
- Parfois, nous sommes amenés à travailler avec des données dont nous ignorons le nombre, comme par exemple le cas des tableaux.
- C'est pour cela qu'on réserve une taille maximale au moment de la déclaration.

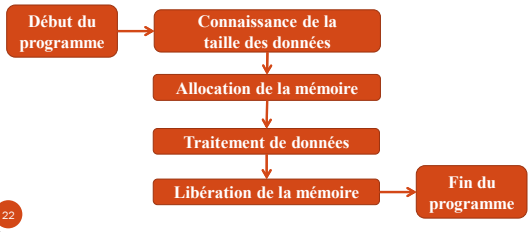
20

- Le problème c'est qu'on réserve une partie de la mémoire qui peut ne pas être utilisée, d'où un gaspillage de la mémoire.
- La solution c'est de laisser la réservation de l'espace mémoire jusqu'au moment où on connait la taille exacte des données à manipuler.
- Et dans ce cas, on réserve un espace mémoire qui correspond à la taille exacte de nos données.
- Une telle réservation se réalise au moment de l'exécution, on parle d'**allocation dynamique** de la mémoire.
- Une fois le traitement des données est terminé, alors pas la peine de les garder en mémoire, donc on procède à la **libération** de l'espace déjà alloué.
- Comme ça, on a géré notre mémoire d'une manière **dynamique**, c'est-à-dire **l'allocation** et la **libération** de l'espace mémoire nécessaire se fait au fur et à mesure de l'exécution du programme.

21

3.1- Allocation dynamique de la mémoire

- Pour la gestion de la mémoire, le langage algorithmique offre deux instructions :
  - L'instruction **Allouer** : permet de réserver l'espace mémoire nécessaire pour un objet donné.
  - L'instruction **Libérer** : permet de libérer l'espace mémoire allouer par l'instruction Allouer.



22

3.1.1-Cas d'une simple variable

Syntaxe : `Allouer(NomPointeur)`
`Libérer(NomPointeur)`

- Avec **NomPointeur** est pointeur de type X, et X un type dont la taille est **n** octets.
- Cette syntaxe réserve un espace mémoire de **n** octets pour stocker que des valeurs de type X.
- Après la réservation on peut manipuler le contenu de notre pointeur **`NomPointeur^`** comme une simple variable ordinaire.

Exemples

- Ecrire un algorithme **Allocation1** qui réserves deux espaces mémoires, l'un pour le type Entier et l'autre pour le type Réel.
- Ecrire un algorithme **Allocation2** qui réserve un espace mémoire de type Entier et le remplit avec une valeur saisie au clavier.

23

```
Algorithme Allocation1
Variable P1 : ^Entier
         P2 : ^Réel
Début
Allouer(P1)
Allouer(P2)
P1^ ← 12
P2^ ← P1^
Ecrire("Le contenu de P1 est ",P1^, et le
contenu de P2 est ", P2^)
P2^ ← P1^ + 0.5
P1^ ← P1^ + 1
Ecrire("Le contenu de P1 est ",P1^, et le
contenu de P2 est ", P2^)
Libérer(P1)
Libérer(P2)
Fin
```

24

```
Algorithme Allocation2
Variable P : ^Entier
Début
Allouer(P)
Ecrire("Donnez une valeur entière : ")
Lire(P^)
Ecrire("La valeur que vous avez saisie est ", P^)
Libérer(P)
Fin
```

25

3.1.2-Cas d'un double pointeur

- Dans ce cas, l'allocation se réalise en deux étapes :
  - **Etape 1** : l'allocation de la mémoire pour le premier pointeur(P1)
    Syntaxe : **`Allouer(P1)`**
  - **Etape 2** : l'allocation de la mémoire pour le deuxième pointeur (P2)
    Syntaxe : **`Allouer(P1^)`**
- La libération se réalise aussi en deux étapes mais dans le sens inverse de l'allocation :
  - **`Libérer(P1^)`**
  - **`Libérer(P1)`**

Exemple

Ecrire un algorithme **Allocation_Pointeur** qui déclare un double pointeur de type Entier, lui réserve de la mémoire. Et en suite initialise et affiche le contenu du deuxième pointeur.

26

```
Algorithme Allocation_Pointeur
Variable P : ^^Entier
         n : Entier
Début
Allouer(P)
Allouer(P^)
Ecrire("Donnez une valeur entière : ")
Lire((P^)^)
Ecrire("La valeur que vous avez saisie est ", (P^)^)
n ← 10
p^ ← @n
Ecrire("p^ =  ", p^)
Ecrire("(p^)^ = ", (P^)^)
Libérer(p^)
Libérer(p)
Fin
```

27

3.1.1-Cas d'un tableau à une seule dimension

Syntaxe : **`Allouer(NomPointeur, N)`**

- Avec **NomPointeur** est pointeur de type X dont la taille est **n** octets, et **N** est une expression entière.
- Cette syntaxe réserver un espace mémoire de **n×N** octets pour stocker que des valeurs de type X.
- Après la réservation, on dispose d'un tableau de N éléments de type X.
- Pour accéder à l'élément de l'indice i, on peut utiliser deux formalismes :
  - Formalisme tableau : dans ce cas on utilise la syntaxe **`NomPointeur[i]`**.
  - Formalisme pointeur : dans ce cas on utilise la syntaxe **`(NomPointeur+i)^.`**

28

Exemple

Ecrire un algorithme **Allocation_Tableau** qui permet de saisir et afficher un tableau de **n** éléments de type entier. L'algorithme doit tout d'abord demander la taille **n** et réserve l'espace mémoire nécessaire. Vous écrivez deux versions en utilisant les deux formalismes tableau et pointeur.

29

- **Formalisme pointeur**

```
Algorithme Allocation_Tableau
Variable T : ^Entier
         n, i : Entier
Début
    Ecrire("Donnes la taille n")
    Lire(n)
  Allouer(T, n)
  Pour i allant de 0 Jusqu'à n-1 Faire
    Ecrire("Donnez l'élément ", i+1)
    Lire((T+i)^)
  FinPour
  Ecrire("Affichage du tableau :")
  Pour i allant de 0 Jusqu'à n-1 Faire
    Ecrire((T+i)^, " ")
  FinPour
  Libérer(T)
Fin
```

30

❖ **Formalisme tableau**

```
Algorithme Allocation_Tableau
Variable T : ^Entier
         n, i : Entier
Début
    Ecrire("Donnez la taille n")
    Lire(n)
  Allouer(T, n)
  Pour i allant de 0 Jusqu'à n-1 Faire
    Ecrire("Donnez l'élément ", i+1)
    Lire(T[i])
  FinPour
  Ecrire("Affichage du tableau :")
  Pour i allant de 0 Jusqu'à n-1 Faire
    Ecrire(T[i], " ")
  FinPour
  Libérer(T)
Fin
```

31

3.1.1-Cas d'un tableau à deux dimensions

❖ Pour allouer dynamiquement de la mémoire on utilise un double pointeur.

❖ Alors l'allocation se réalise en deux étapes :

- **Etape 1** : on réserve le nombre de ligne **N**

  Syntaxe : **`Allouer(NomPointeur, N)`**

  → Alors, on obtient un tableau de pointeurs de taille N.

- **Etape 2** : on réserve le nombre de colonnes, pour cela, le processus d'allocation sera répété pour chaque élément du tableau ainsi obtenu dans la première allocation.

  Syntaxe :

```
Pour i allant de 0 Jusqu'à N-1 Faire
        Allouer(NomPointeur[i], M)
FinPour
```

32

❖ Pour accéder à l'élément de l'indice (i,j), on peut utiliser deux formalismes :

- Formalisme tableau : dans ce cas on utilise la syntaxe **`NomPointeur[i, j]`**.
- Formalisme pointeur : dans ce cas on utilise la syntaxe **`((NomPointeur+i)^+j)^`**.

❖ Pour la libération de la mémoire, on procède comme à l'allocation mais dans le sens inverse :

```
Pour i allant de 0 Jusqu'à N-1 Faire
     Libérer(NomPointeur[i])
FinPour
Libérer(NomPointeur)
```

33

Exemple

Ecrire un algorithme **Allocation_Tableau2** qui permet de saisir et afficher un tableau de type entier de **n** lignes et de **m** colonnes L'algorithme doit tout d'abord demander les nombres n et m et réserver l'espace mémoire nécessaire. Vous écrivez deux versions en utilisant les deux formalismes tableau et pointeur.

❖ **Formalisme pointeur**

```
Algorithme Allocation_Tableau
Variable T : ^^Entier
         n, m, i, j : Entier
Début
  Ecrire("Donnez le nombre de lignes n et le
  nombre de colonnes m")
  Lire(n, m)
```

34

```
  Allouer(T, n)
  Pour i allant de 0 Jusqu'à n-1 Faire
    Allouer((T+i)^, M)
  FinPour
  Pour i allant de 0 Jusqu'à n-1 Faire
    Ecrire("Donnez l'élément (",i+1,",",j+1, ")")
    Lire(((T+i)^+j)^)
  FinPour
  Ecrire("Affichage du tableau :")
  Pour i allant de 0 Jusqu'à n-1 Faire
    Ecrire(((T+i)^+j)^, " ")
  FinPour
  Pour i allant de 0 Jusqu'à n-1 Faire
    Libérer(T+i)
  FinPour
  Libérer(T)
Fin
```

35

❖ **Formalisme tableau**

```
Algorithme Allocation_Tableau
Variable T : ^^Entier
         n, m, i, j : Entier
Début
  Ecrire("Donnez le nombre de lignes n et le
  nombre de colonnes m")
  Lire(n, m)
  Allouer(T, n)
  Pour i allant de 0 Jusqu'à n-1 Faire
    Allouer(T[i], M)
  FinPour
  Pour i allant de 0 Jusqu'à n-1 Faire
    Ecrire("Donnez l'élément (",i+1,",",j+1, ")")
    Lire(T[i,j])
  FinPour
```

36

```
  Ecrire("Affichage du tableau :")
  Pour i allant de 0 Jusqu'à n-1 Faire
    Ecrire(T[i, j], " ")
  FinPour
  Pour i allant de 0 Jusqu'à n-1 Faire
    Libérer(T[i])
  FinPour
  Libérer(T)
Fin
```

37

Remarques

1. En général, le nom d'un tableau est un pointeur constant qui représente l'adresse du premier élément. C'est pour cela qu'on a les deux formalismes tableau et pointeur.
2. En algorithmique, on suppose que le système possède une place mémoire sans limite et que l'instruction Allouer se termine correctement. Mais en pratique, ce n'est pas le cas, alors des mécanismes existent dans les langages de programmation pour savoir si la réservation a échouée ou a réussie.

38

3.2- Le mot clé Redim

- Le mot clé **Redim** permet de redimensionner un tableau.

  Syntaxe : `Redim NomTableau[Taille]`

- L'utilisation de ce mot clé n'entraine pas la perte des données déjà enregistrées dans le tableau (en cas d'expansion du tableau).

Exemple

```
Algorithme Redimensionner_Tableau
Tableau T[4] : Entier
Début
  Ecrire("Donnez 4 valeurs entières :")
  Lire(T[0], T[1], T[2], T[3])
  Ecrire(T[0]," ", T[1], " ", T[2], " ", T[3])
  Redim T[6]
  T[4] ← 10
  T[5] ← 20
Fin
```

39

- L'utilisation du mot clé **Redim** permet de manipuler les tableaux d'une manière dynamique(Tableau dynamique), car l'espace mémoire est réservé selon le besoin.
- Alors dans le cas où l'on ignore la taille des données à manipuler, on peut procéder ainsi :
  - Déclarer le tableau sans préciser sa taille.
  - Savoir la taille des données.
  - Redimensionner le tableau avec la nouvelle taille.
  - A la fin du traitement on redimensionne le tableau avec une taille nulle pour libérer la mémoire.

40

Exemple

```
Algorithme Redimensionner_Tableau2
Variable N : Entier
Tableau T[] : Entier
Début
  Ecrire("Donnez la taille du tableau :")
  Lire(N)
  Redim T[N]
  .
  .
  .
Fin
```

Remarque

Le mot clé **Redim** n'existe pas dans tous les langages de programmation comme le langage C, mais on trouve d'autres alternatives qui réalisent la même chose.

41