Ministère de l'enseignement supérieur et de la recherche scientifique
Université Djillali BOUNAAMA - Khemis Miliana (UDBKM)
Faculté des Sciences et de la Technologie
Département de Mathématiques et d'Informatique

Chapitre 4

Les Listes Linéaires Chaînées

MI-L1-UEF221 : Algorithmiques et Structures de Données II

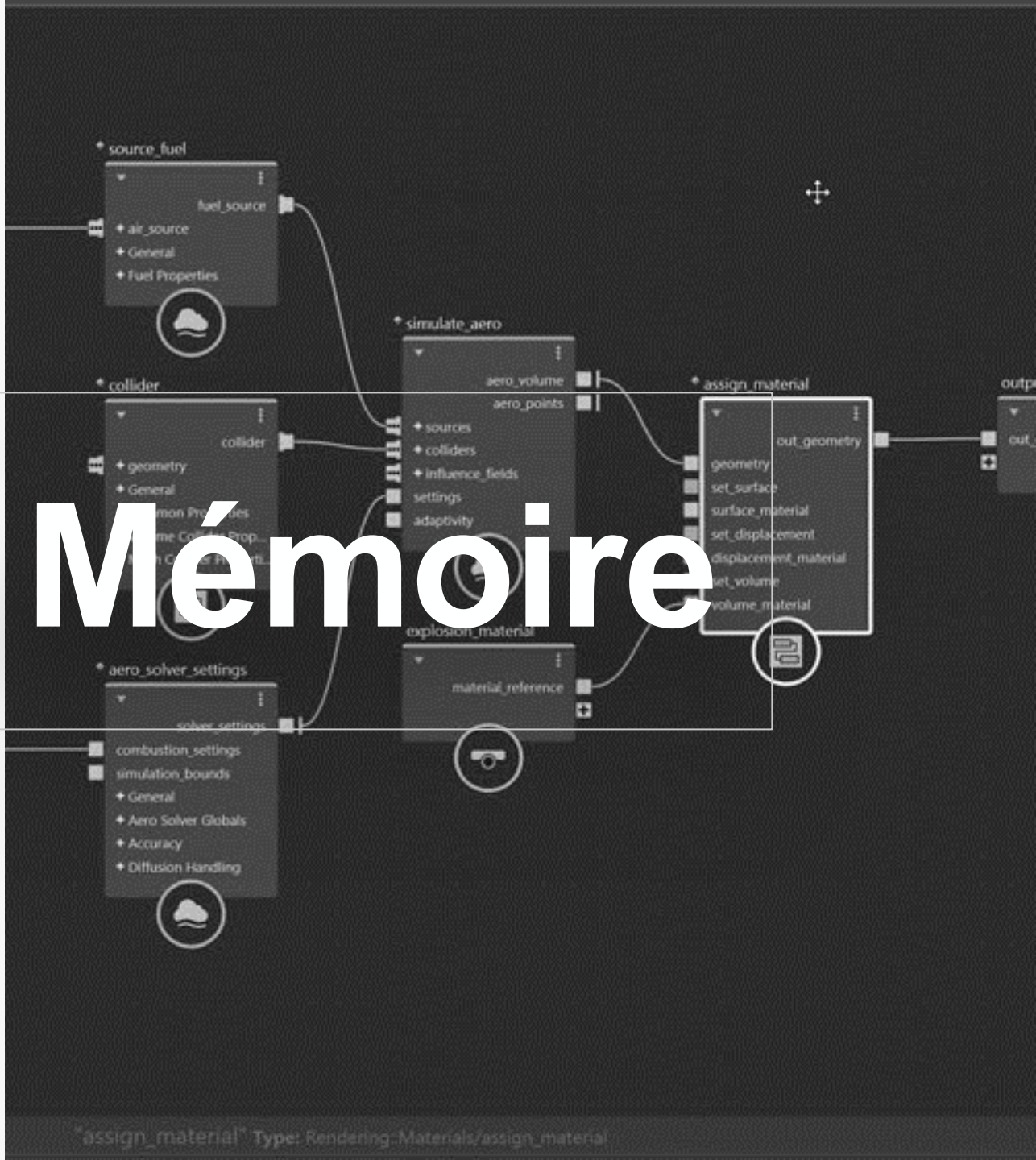
Noureddine AZZOUZA

n.azzouza@univ-dbkm.dz

Plan du Cours

1. Allocation mémoire
3. Les pointeurs
4. Allocation dynamique et Allocation statique
4. Les Listes Linéaires Chainées
    - 4.1 Définition
    - 4.2 Modèle des LLC
    - 1.3 Opérations sur les LLC
    - 1.3 LLC particulières

Allocation Mémoire



Introduction

- ✓ L'espace mémoire occupé par une structure de données dynamique est variable.
- ✓ C'est intéressant pour représenter des ensembles à tailles variables.
- ✓ On peut donc agrandir ou rétrécir la taille de l'ensemble durant l'exécution du programme.
- ✓ Certains problèmes nécessitent la gestion d'un ensemble dynamique.

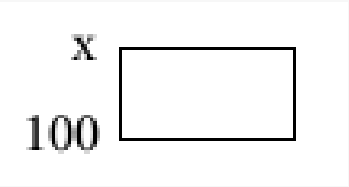
Notion d'allocation mémoire

- ✓ La **Mémoire Centrale (MC)** est formée par des cases numérotées. Chaque case peut stocker un octet (8 bits).
- ✓ Une **variable** est une zone contigüe en MC (une case ou un ensemble de cases qui se suivent).
  - ➢ Sa **taille** (en nombre de cases) dépend du type de la variable (ex: un entier occupe 4 cases, un réel occupe 8 cases, ...etc).
  - ➢ **L'adresse** d'une variable est le numéro de sa première case.

Notion d'allocation mémoire

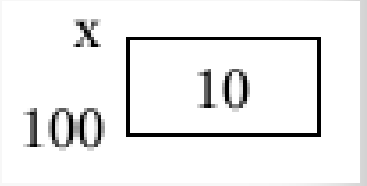
PASCAL	C
VAR x: integer;	*int x;*

✓ **x** est le **nom** donné pour référencer l'emplacement mémoire associé à la variable (la case d'adresse 100)

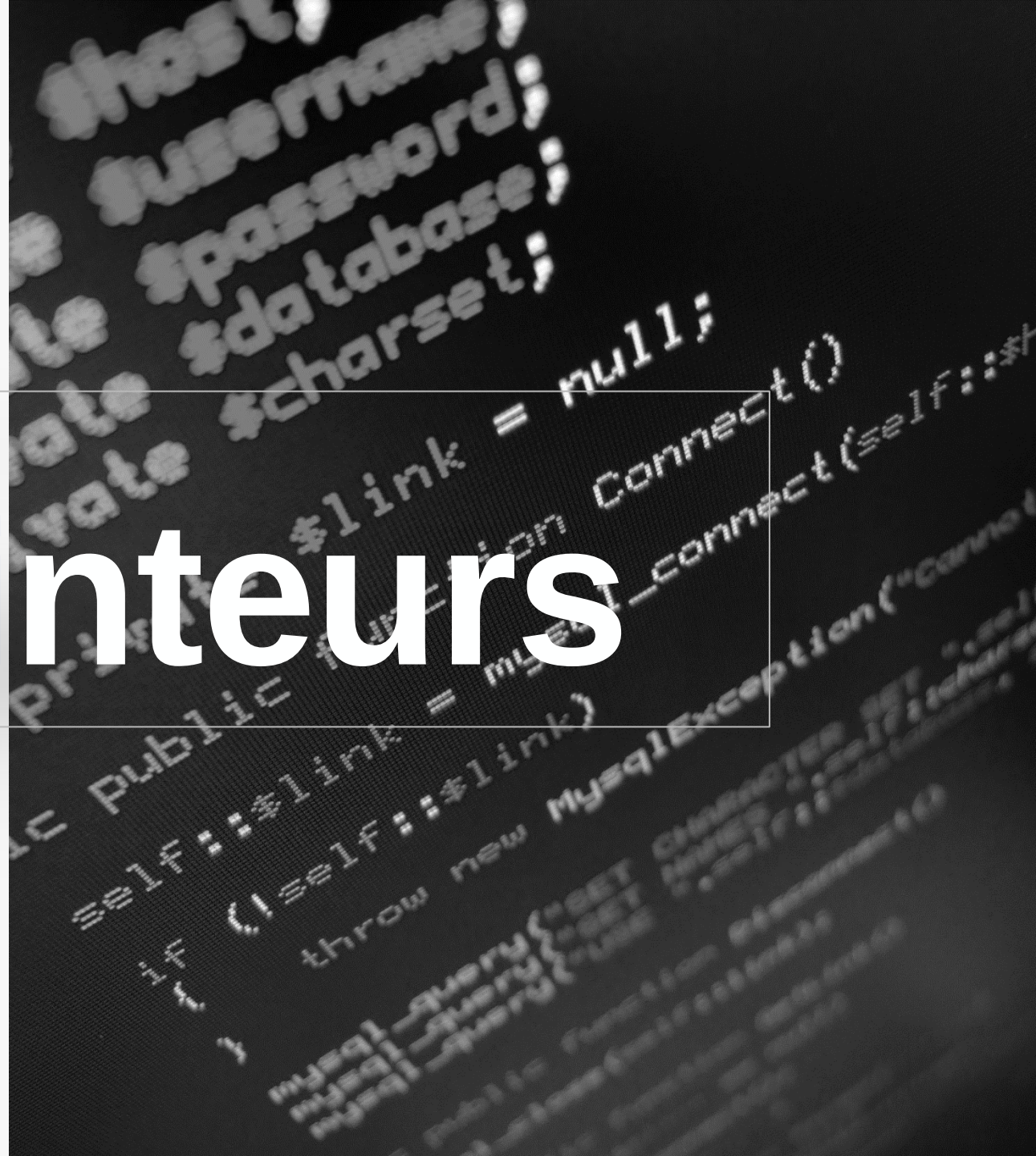


PASCAL	C
x := 10;	*x = 10;*

✓ quand on affecte une valeur (par ex 10) à x, On dit alors que le contenu de l'adresse 100 est 10



Les Pointeurs



Pointeurs : Déclaration

- ✓ **Un pointeur**: est une variable qui peut contenir des adresses de variables.
- ✓ on peut déclarer un pointeur vers un entier comme suit:

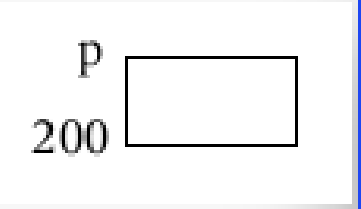
PASCAL

```
VAR p: ^integer;
```

C

```
int *p;
```

- ✓ d'après cette déclaration, on peut affecter à la variable p l'adresse d'une variable de type entier.



- ✓ En C, l'expression **&v** retourne l'adresse de la variable v. Certains compilateur Pascal (non standards) offrent un tel mécanisme (**Addr(v)** en turbo-pascal retourne l'adresse de la var v)

Pointeurs : Adresse d'une variable

- ✓ donc si on écrit par exemple :

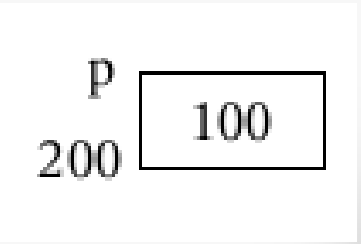
PASCAL

```
p := Addr(x);
```

C

```
p = &x;
```

- ✓ on aura dans p l'adresse de x :
- ✓ on dit alors que p **pointe** x.



Pointeurs : Modification indirecte

- ✓ on peut modifier la valeur de x indirectement (sans utiliser x):

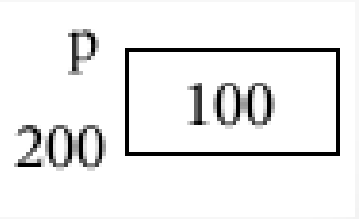
PASCAL

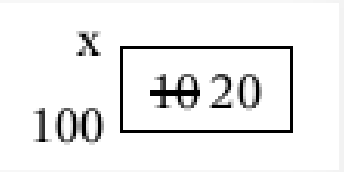
```
p^ := 20;
```

C

```
*p = 20;
```

- ✓ la valeur de x sera alors modifiée (par une affectation indirecte)





Allocation dynamique et Allocation statique

Types d'allocation des variables

✓ Allocation de variables veut dire création de variables. Donc réservation d'espace mémoire en associant à chaque variable l'adresse d'une zone vide en mémoire.

Allocation statique

- ✓ L'allocation de l'espace se fait au début d'un traitement.
- ✓ gérée automatiquement par le système
- ✓ l'espace est connu à la compilation.
- ✓ les variables déclarées représentent des variables allouées statiquement

Allocation dynamique

- ✓ L'allocation de l'espace se fait au fur et à mesure de l'exécution du programme.
- ✓ Gérée manuellement par le programmeur
- ✓ l'espace est connu à la compilation.
- ✓ L'utilisateur doit disposer des deux opérations : allocation et libération de l'espace.

Allocation Dynamique

PASCAL

Allocation

new (p) : *cette procédure alloue une nouvelle variable et affecte son adresse à la variable p.*

Libération

dispose(p) : *détruit la variable pointée par p.*

C

ALLOCATION

malloc(nb_octets): *fonction qui alloue une zone mémoire de taille nb_octets et retourne son adresse comme résultat.*

Libération

free(p) : *détruit la variable pointée par p.*

Allocation Dynamique

PASCAL

```
var
        p : ^char; { allocation statique d'une var ( p ) de type pointeur }
begin
        new(p);  { allocation dynamique d'une var caractère }
        p^ := 'A'; { utilisation indirecte de la var dynamique }
        dispose(p); { destruction de la var dynamique }
end.
```

Allocation Dynamique

C

```c
int main()
{
    char *p;                    /* allocation statique d'une var ( p ) de type pointeur */
    p = malloc( sizeof(char) ); /* allocation dynamique d'une var de même taille
    qu'un caractère : sizeof( type ) retourne le nb d'octets nécessaire pour
    représenter une var de ce type */
    *p = 'A';                   /* utilisation indirecte de la var dynamique */
    free(p);                    /* destruction de la var dynamique */
    return 0;
}
```

Dynamique / Statique

Remarques

- ✓ **Pointeur Vide**

PASCAL	C
$p := NIL;$	$p = \text{NULL};$

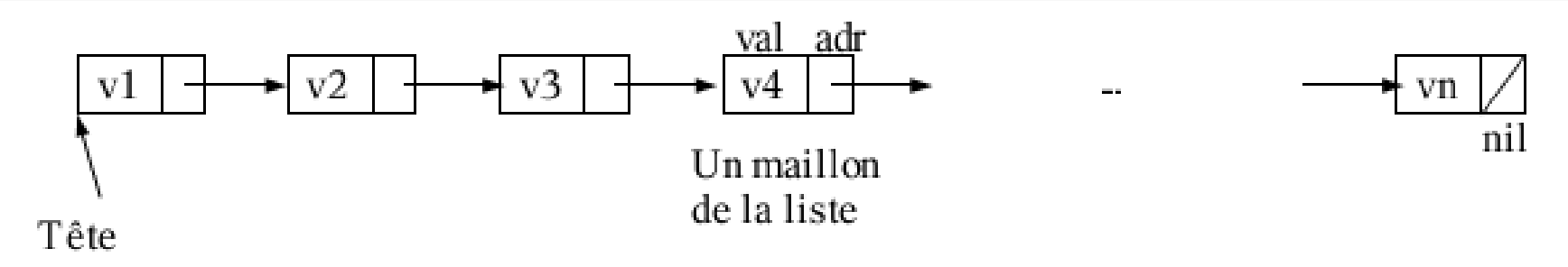
- ✓ Les constantes pointeurs NIL (en Pascal) ou bien NULL ou 0 (en C) indiquent l'absence d'adresse. Donc, par exemple en Pascal, l'affectation p := NIL, veut dire que p ne pointe aucune variable.

- ✓ Il ne faut jamais utiliser l'indirection (^ en pascal ou * en C) avec un pointeur ne contenant pas l'adresse d'une variable, il y aura alors une erreur de segmentation.

Les listes linéaires chainées

Définition

✓ Une Liste Linéaire Chaînée est une structure de données (le plus souvent dynamique) pour représenter un ensemble de valeurs. Ces valeurs sont chaînées entre elles formant une suite :



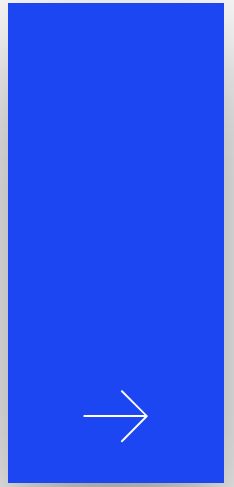
Propriétés

- ✓ Chaque valeur v de l'ensemble est stockée dans un maillon
- ✓ Un maillon est une structure à 2 champs :
  - ➢ **val** : de type quelconque ,
  - ➢ **adr** : pointeur vers le prochain maillon >
- ✓ Dans le dernier maillon de la liste, le champs **adr** contient le pointeur NIL (indiquant par convention la fin de la liste).
- ✓ L'adresse du 1er maillon (**la tête de la liste**) est importante. Elle doit toujours être sauvegardée dans une variable pour pouvoir manipuler la liste.
- ✓ Si la liste est vide (ne contient aucun maillon), la tête doit alors être positionnée à NIL.

Modèle des LLC : Structure d'un maillon

✓ Dans le langage algorithmique, on définira le type d'un maillon comme suit

```
Type    maillon = Enregistrement
                  Val : Typeqq /*type quelconque*/
                  Adr : Pointeur(maillon)
        Fin
```



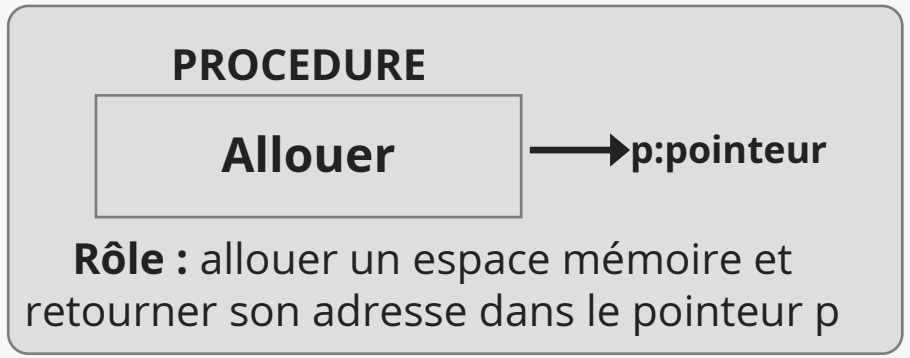
Modèle des LLC : Types Abstrait de Données

✓ Afin de développer des algorithmes sur les LLCs, on construit une machine abstraite (**Types Abstrait de Données - TAD**) avec les opérations suivantes :

- **Allouer(P)** :allocation d'un espace de taille spécifiée par le type de P. L'adresse de cet espace est rendue dans la variable de type Pointeur P.
- **Libérer(P)** :libération de l'espace pointé par P.
- **Valeur(P)** :consultation du champ Valeur du maillon pointé par P.
- **Suivant(P)** :consultation du champ Suivant du maillon pointé par P.
- **Aff_Adr(P, Q)** :dans le champ Suivant du maillon pointé par P, on range l'adresse Q.
- **Aff_Val(P,Val)** :dans le champ Valeur du maillon pointé par P, on range la valeur Val

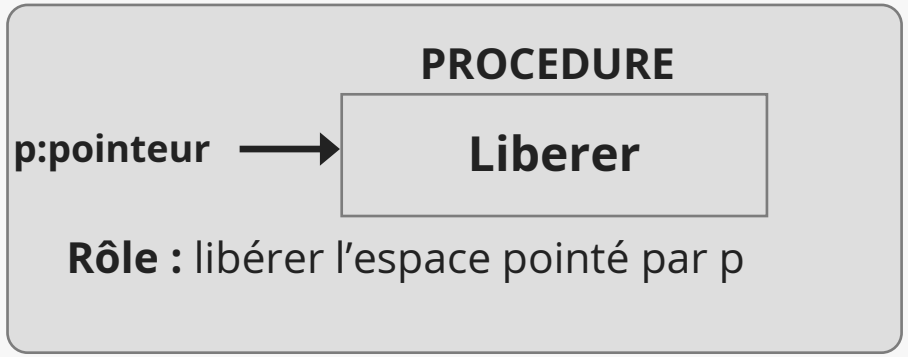
Modèle des LLC : Implémentation de la TAD

Allouer(P)




```
Procédure Allouer (VAR p: pointeur(maillon))
Début
    new(p); //ou malloc() en C
Fin;
```

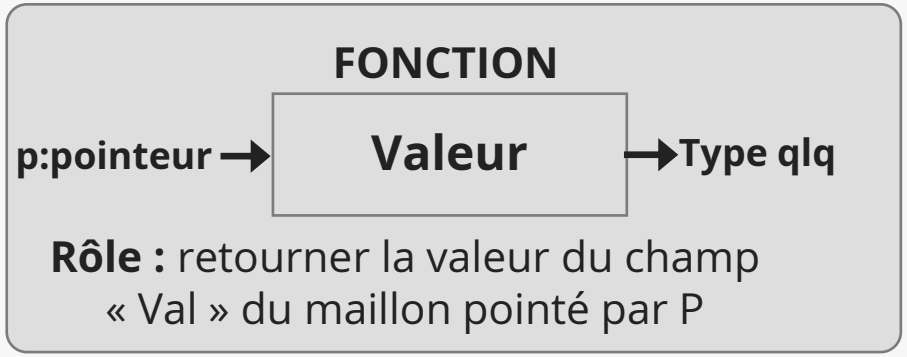
Liberer(P)




```
Procédure  Liberer (p: pointeur(maillon))
Début
    dispose(p); //ou free(p) en C
Fin;
```

Modèle des LLC : Implémentation de la TAD

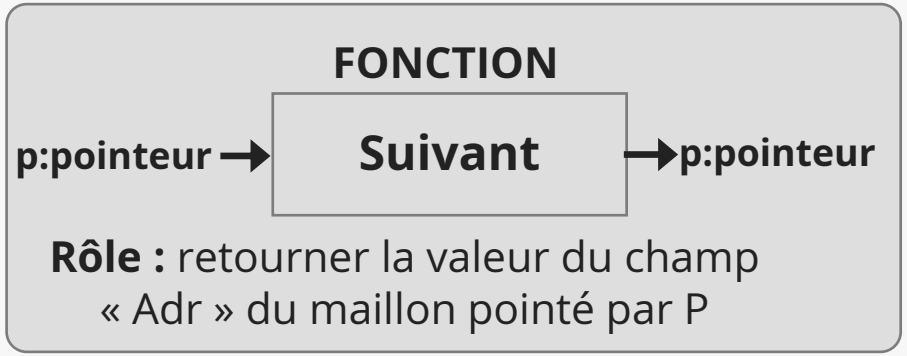
Valeur(P)



Rôle : retourner la valeur du champ « Val » du maillon pointé par P

```
Procédure Valeur (p: pointeur(maillon)): type qlq
Début
        Valeur ← p^.val;
Fin;
```

Suivant(P)

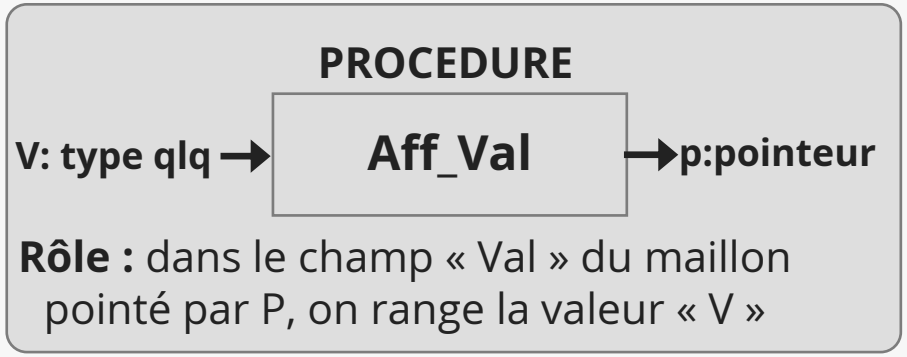


Rôle : retourner la valeur du champ « Adr » du maillon pointé par P

```
Procédure Suivant (p: pointeur(maillon))
Début
        Suivant ← p^.adr;
Fin;
```

Modèle des LLC : Implémentation de la TAD

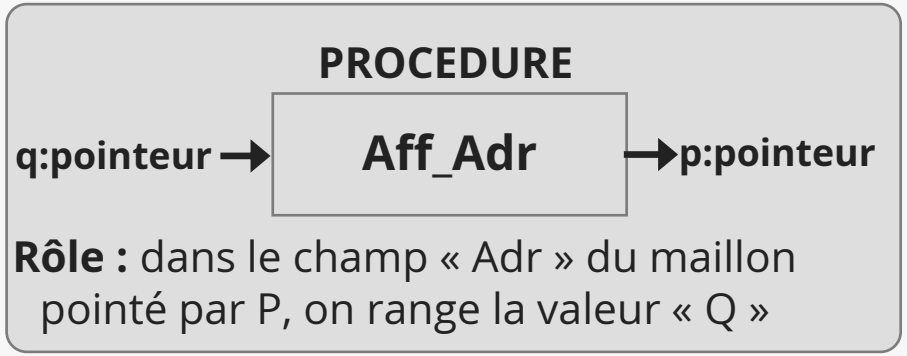
Aff_Val(P,V)



Rôle : dans le champ « Val » du maillon pointé par P, on range la valeur « V »

```
Procédure Aff_ Val (V:typeqlq; VAR p: pointeur(maillon) )
Début
        p^.val ← V;
Fin;
```

Aff_Adr(P, Q)



Rôle : dans le champ « Adr » du maillon pointé par P, on range la valeur « Q »

```
Procédure  Suivant (Q:pointeur(maillon);VAR P: pointeur(maillon))
Début
        p^.adr ← Q;
Fin;
```

Exemple : Créer une liste de N éléments

```
Algorithme  CreerListe;
   Type  maillon=enregistrement
            Val:entier;
            Adr:pointeur(maillon)
         Fin;
   Var   Tete, P, Q:u : pointeur(maillon);
         i, N, V : entier ;
Début
   Tete ← Nil;
   P ← Nil;
   Ecrire('Donner le d''elements de la liste');
   Lire (N);
```

```
Pour i de1 à N faire
   Lire(Val) ;
   Allouer(Q) ;
   Aff_ val(Q, val) ;
   Aff_ adr(Q, NIL) ;
   Si (Tete <> Nil) Alors
         Aff_ adr(P, Q)
   Sinon
         Tete←Q
   FSi;
   P←Q;
FPour;
```

Exemple : Parcourir et afficher les éléments de la liste

```
P ← Tete ;
TQ (P <> Nil) Faire
    Ecrire(Valeur(P)) ;
    P ← Suivant(P) ;
FTQ;

Fin.
```