Université Batna 2
Faculté des Mathématique et Informatique
Département d'Informatique
Master ISIDS - Semestre 1

Programmation Avancée

Préparé par Dr O.MESSAOUDI

2022/2023

Chapitre I
Rappels

- Généralité sur l'algorithmique
- Algorithmique et Programmation
- Langage algorithmique
- Tableaux
- Pointeurs

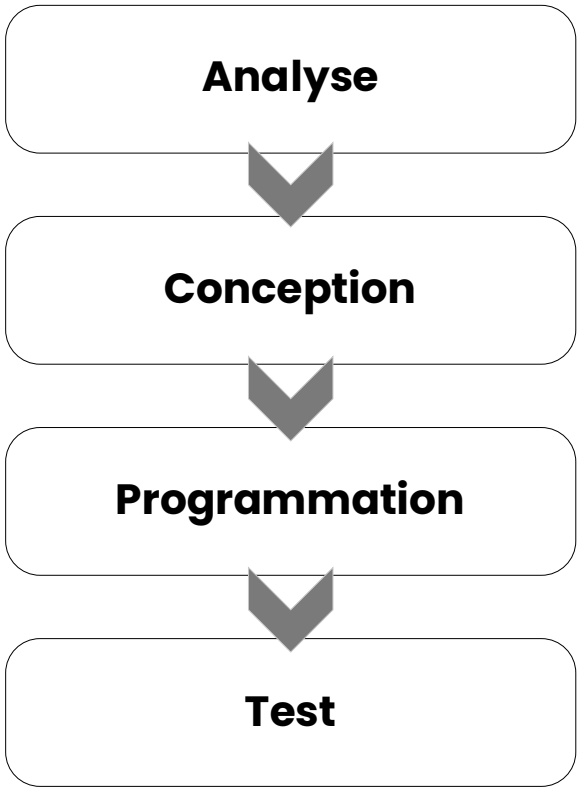
Généralité sur l'algorithmique

- **Historique:** L'algorithmique est un terme d'origine arabe, vient de Al Khawarizmi, un mathématicien persan du 9ième siècle.

- **Définition:** Un algorithme est une suite finie d'opérations (étapes) élémentaires constituant un schéma de calcul ou de résolution d'un problème.

Généralité sur l'algorithmique

Étapes de conception d'un algorithme:



Analyse des besoins définis dans un cahier de charges et définition d'une spécification claire de ce que doit faire le programme

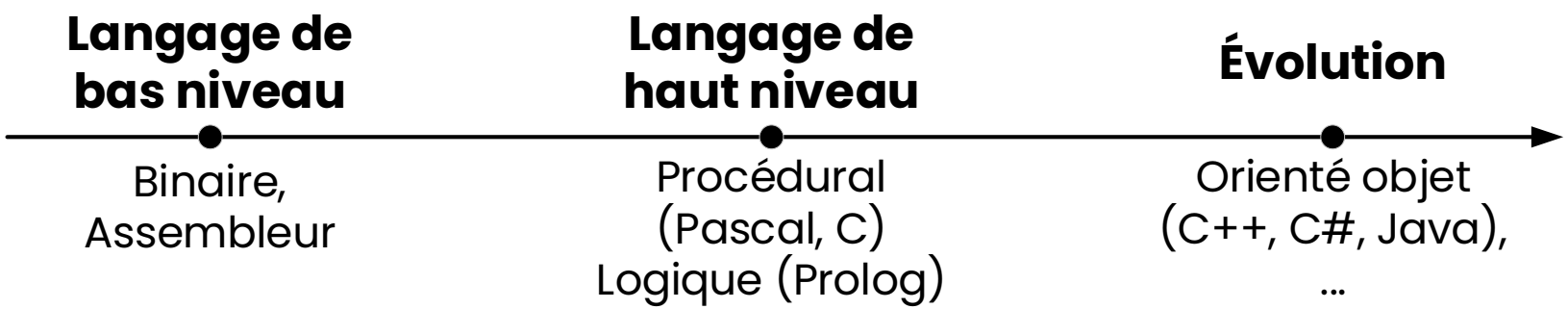
Conception de l'architecture du programme, ainsi la conception détaillée de chaque composant (sa contribution)

Réalisation et implémentation des composants définits pendant la conception, assemblage des différents composants

Test unitaire, test d'intégration, test système, et test d'acceptation

Algorithmique et Programmation

- **Définition:** Un programme est la description (traduction) d'un algorithme dans un langage de programmation.



Algorithmique et Programmation

Qualité d'un bon algorithme/programme

- **Correcte:** Il faut que le programme exécute correctement ses tâches.
- **Complet:** Il faut que le programme considère tous les cas possibles.
- **Efficace:** Il faut que le programme exécute sa tâche avec efficacité qui se mesure sur:
  - Temps (complexité temporelle),
  - Ressources (complexité spatiale),

Langage algorithmique

Structure d'un algorithme

```
Algorithme NomDeLAlgorithme ;
Const   c1= ...  c2=... ;
Var     var1 : Type ;  var2 : Type2 ;
        Procedure P1(....)
           Debut
            .........
           fin
        fonction F1(....) : Type ;
           Debut
           .........
           fin
debut
  ...
  P1(...) ;
  Var1 ← F1(...) ;
  ...
fin.
```

Langage algorithmique

Structures de données

- **Simples:** entier, réel, booléen, caractère, tableau, etc.
- **Complexes:** incluent:
  - Structures séquentielles: Listes, Piles, Files
  - Structures hiérarchiques: Arbres

Tableaux

- C’est un objet décomposé en plusieurs **éléments** de même type,
- Chaque élément est repéré par un **indice** (index),
- Le nombre d’éléments constitue sa **taille**,
- Le nombre d’indices qui permet de désigner un élément est appelée **dimension** du tableau,
- Le type de l’indice est un **intervalle** [0..taille -1]

Tableaux

Déclaration: se fait en précisant le mot **TABLEAU**, suivi par sa **taille** et par le **type** de ses éléments.

Tableau nom_tableau[**taille**]:**type**

- L'accès à un élément s'effectue en précisant le nom du tableau suivi par l'indice entre crochets: **Tab[1]**

Tableaux

Algorithmique	Langage C
Algorithm exemple; **Var** Tableau tab[10]:entier; **Début** tab[0] ← 0; affichier(tab[0]); affichier(*tab); affichier(tab); **Fin**	**int main()** **{** int tab[4] = {10, 23,505,8}; tab[0] = 10; printf("%d", tab[0]); printf("%d", *tab); printf("%p", tab); **}**

Tableaux

Tableaux multidimensionnels

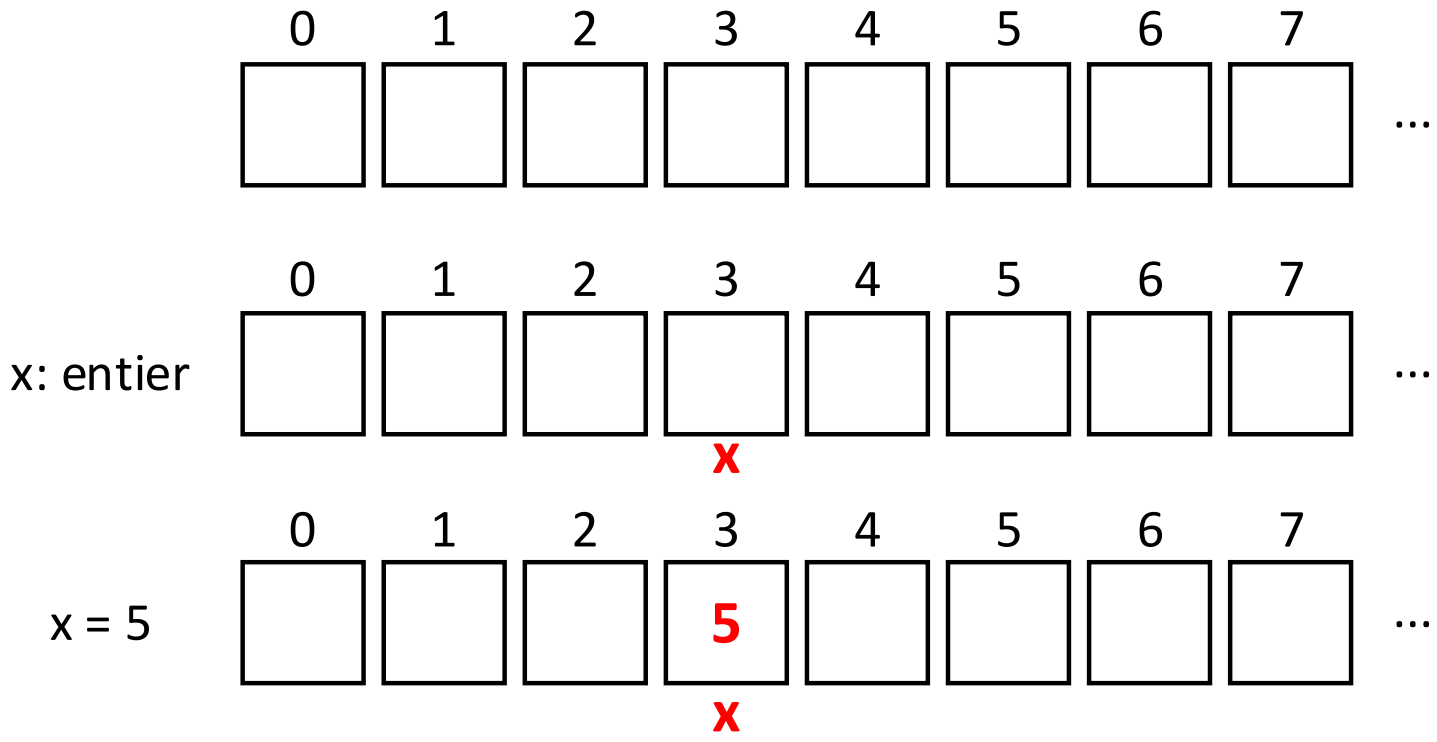
Un tableau multidimensionnel est considéré comme un tableau dont les éléments sont eux mêmes des tableaux.

Tableau nom [**taille_dim_1, taille_dim_2, …**]: **type**

- L'accès à un élément s'effectue en précisant le nom suivi par l'indice des dimensions: **Tab[1,2]**

Les pointeurs

Variable: est destinée à contenir une valeur du type avec laquelle elle est déclarée. Physiquement cette valeur se situe en mémoire.



Les pointeurs

Un pointeur est une variable destinée à contenir une ***adresse mémoire***, c-à-d une valeur identifiant un emplacement en mémoire.

Tout pointeur est associé à un type d'objet.

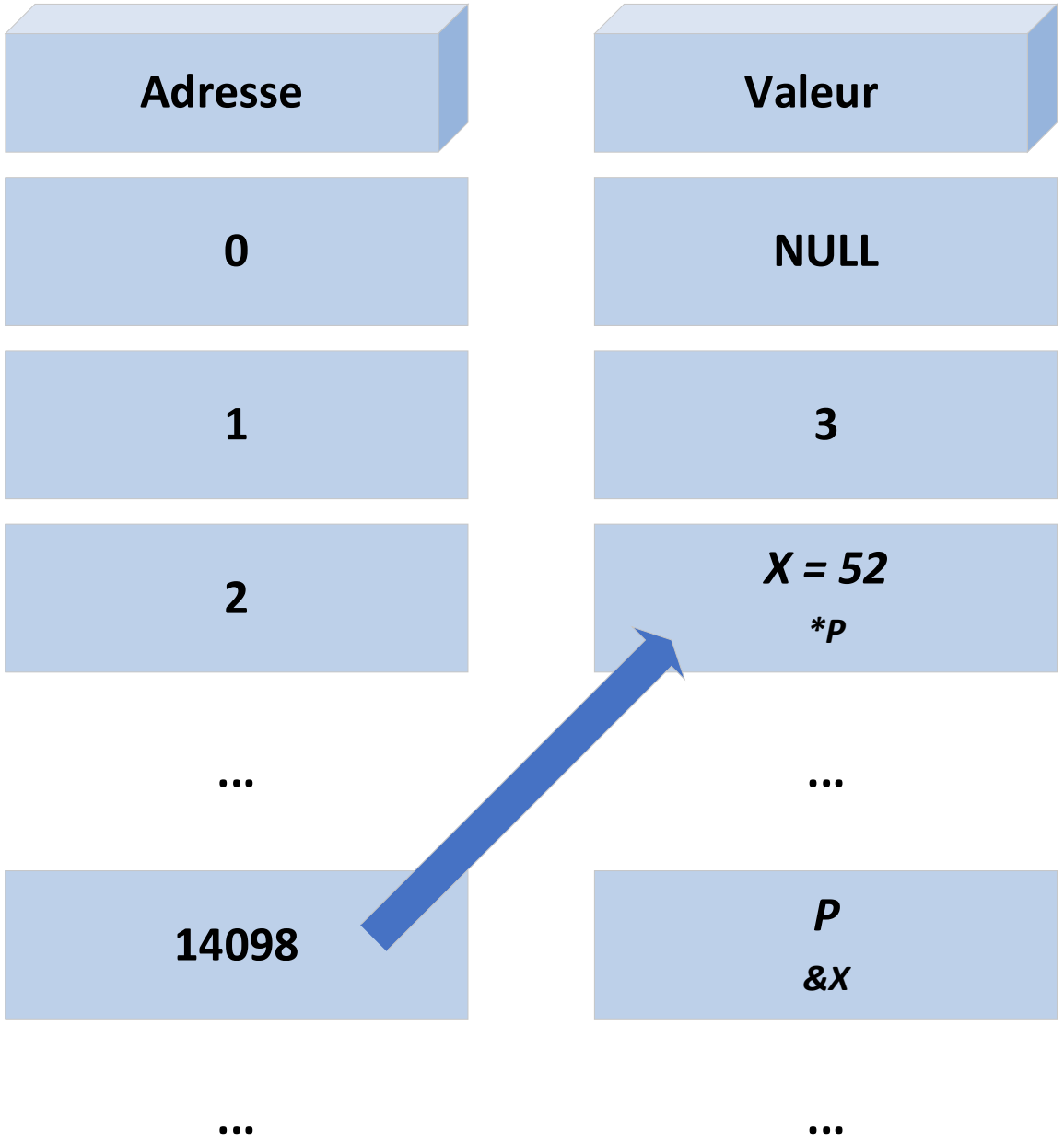
Opération sur les pointeurs:

- ***Affectation*** d'une adresse au pointeur
- ***Utilisation*** du pointeur pour accéder à l'objet dont il contient l'adresse

Les pointeurs

Exemples

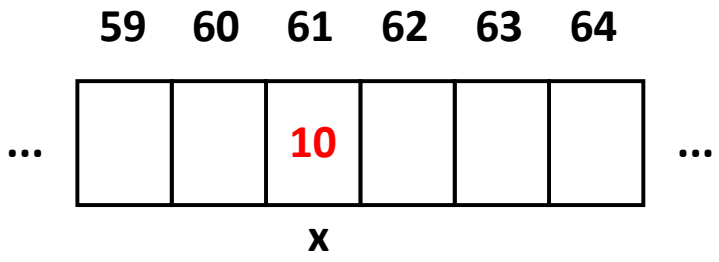
```
Var X:entier;
      P:*entier;
...
x=52;
P=&X;
...
```



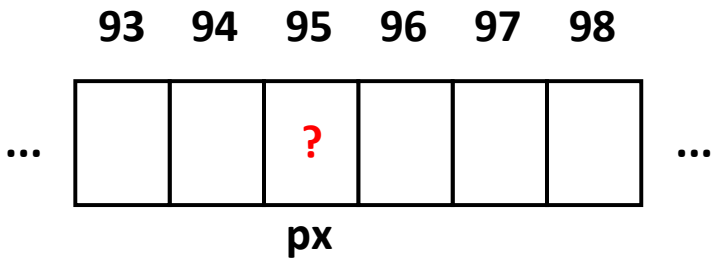
Les pointeurs

Exemples

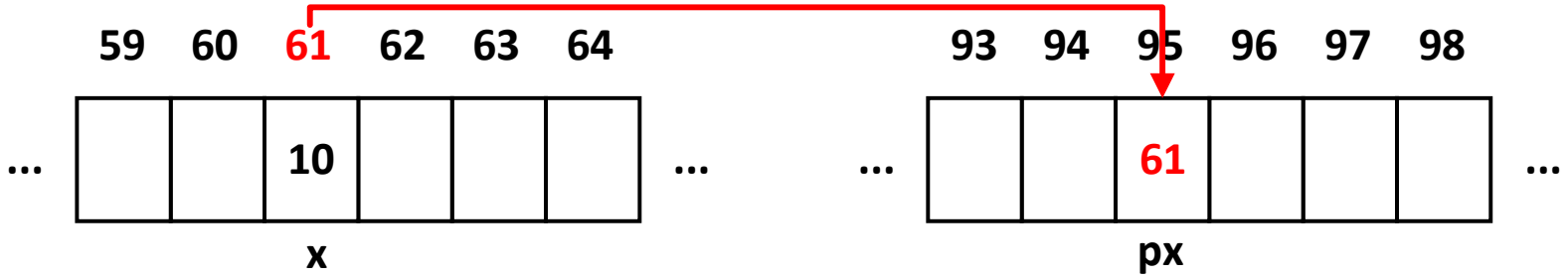
x:enter; x = 10;



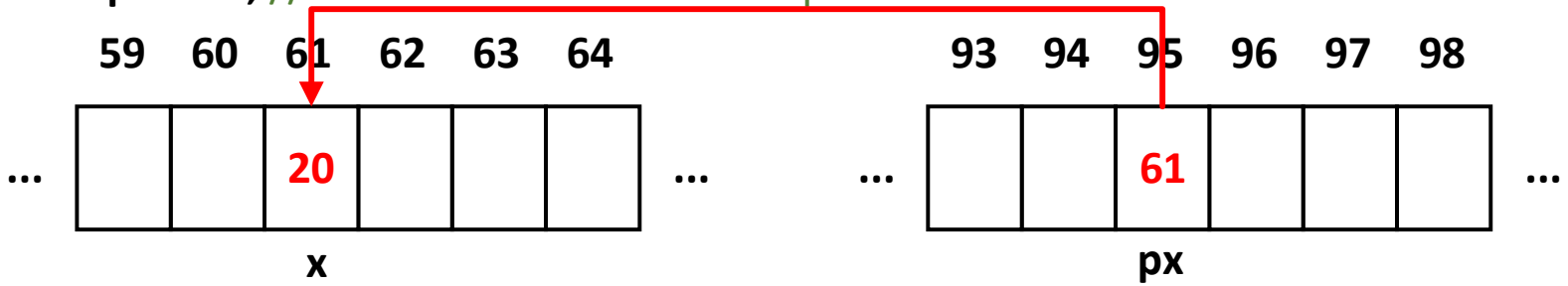
px:*enter;



px = &x; // affectation de @ de x au pointeur



***px = 20;** // affectation en utilisant le pointeur



Les pointeurs

Allocation dynamique du mémoire

- **Allocation statique:** la déclaration des variables réserve de l'espace mémoire pour ces variable.
- *Limitation:* connaitre au début l'espace nécessaire au stockage des variables.
- **Allocation dynamique:** l'espace nécessaire (ex. les tableaux) peut varier d'une exécution à une autre.

Les pointeurs

Algorithmique

- La **déclaration**:

$$ptr{:}*\,type;$$
$$ptr \leftarrow Nil;$$

- **Réserver** un espace mémoire/retourner un pointeur vers type:

$$ptr \leftarrow (*\,type)\ allouer()$$

- **Libérer**:

$$Liberer(ptr)$$

Les pointeurs

Langage C

- La **déclaration**:

$$type \ * ptr = NULL;$$

- **Réserver** un espace mémoire/retourner un pointeur vers type:

$$ptr = (* type)\boldsymbol{malloc}\big(\boldsymbol{sizeof}(type)\big);$$

- **Libérer**:

$$\boldsymbol{free}(ptr)$$

Les pointeurs

Les pointeurs et les tableaux

- Par défaut, le tableau est de **grandeur statique**, c-à-d qu'il impossible de les changer de taille après la compilation.
- Cependant, il est possible de changer la taille des **tableaux dynamiques** après la compilation.
- Pour faire des tableaux dynamique, il faut **réserver** un espace mémoire d'une taille donnée, puis d'assigner un **pointeur** à cet espace mémoire.

Les pointeurs

Les pointeurs et les tableaux (Algorithmique)

- La **déclaration**:

$$TAB{:}*\,type;$$
$$TAB \leftarrow Nil;$$

- **Réserver** un espace mémoire/retourner un pointeur vers type:

$$TAB \leftarrow (*\,type)\ \boldsymbol{allouerTab}(N)$$

- **Libérer**:

$$\boldsymbol{Liberer}(TAB)$$

Les pointeurs

Les pointeurs et les tableaux (Langage C)

- La **déclaration**:

$$type\ *TAB = NULL;$$

- **Réserver** un espace mémoire/retourner un pointeur vers type:

$$TAB = (*type)\boldsymbol{malloc}\big(\boldsymbol{N} * \boldsymbol{sizeof}(type)\big);$$

$$TAB = (*type)\boldsymbol{calloc}\big(\boldsymbol{N}, \boldsymbol{sizeof}(type)\big);$$

- **Libérer**:

$$\boldsymbol{free}(ptr)$$

Les pointeurs

Les pointeurs et le passage par variable

- Une autre utilité des pointeurs dans le langage C est de permettre le passage par variable des paramètres dans les procédures.

Algorithmique	Langage C
Procedure permuter**(**var x,y:**entier)** **Var** temp:entier; **Début** temp←x; x←y; y←temp; **Fin**	**void** permuter(**int** *px,**int** *py) { int temp; temp = *px; *px=py; *py=*temp; }
x←5;y←80;permuter(x,y);	x=5;y=80;permuter(&x,&y);

Les pointeurs

Les pointeurs et les autoréférences

- **Un autoréférence** est une structure dont un de ces membres est un pointeur vers une autre structure du même modèle.
- Cette représentation permet de construire des listes chainées et des arbres.

Algorithmique	Langage C
module:structure { moy:reel; **suiv:*module;** **}**	**struct module** { float moy; **struct module** *suiv; }

Chapitre II
Les structures séquentielles

- Introduction
- Listes linéaires chainées
- Files (FIFO)
- Piles (LIFO)

Introduction

Structures séquentielles: est un ensemble de variables organisées séquentiellement auxquelles on peut accéder:

- Soit directement par leur numéro d'ordre
- Soit en les parcourant une par une dans l'ordre

Les listes

Définition

Une liste linéaire chainée (LLC) est un ensemble de **éléments reliés** entre eux.

Un **élément** est une entité renfermant de l'information et ayant un pointeur sur le maillon qui la suit.

C'est une structure avec deux champs:

- Un champ **Valeur** contenant l'information
- Un champ **Suivant** donnant l'adresse du prochain maillon.

Les listes

Définition

Une LLC est caractérisée par:

- L'adresse de son premier élément (la tête de liste)
- NIL constitue l'adresse qui ne pointe aucun élément (indiquant la fin de la liste)
- Si la liste est vide, la tète doit alors être positionnées à NIL

Les listes

Définition

Algorithmique	Langage C
type Element **= Structure** **val : types** **Suiv: *Element** **fin**	**struct** Element { types val; struct Element *suiv; }
Var tete: *Element	struct Element *tete;

Les listes

Modèle

Opération	Role
Allouer(P)	Allouer (dynamiquement) un nouveau élément et affecte son adresse dans le pointeur P.
Libérer(P)	Détruire l'élément pointé par P.
Valeur(P)	Renvoyer le contenu du champs « val » de l'élément pointé par P.
Suivant(P)	Renvoyer le contenu du champs « $suiv$ » de l'élément pointé par P.
AffSuiv(P,Q)	Affecter le pointeur Q dans le champs « $suiv$ » de l'élément pointé par P.
AffVal(P,v)	Affecter la valeur v dans le champs « val » de l'élément pointé par P.

Les listes

Opérations (langage C)

```c
struct element{
    int val;
    struct element * suiv;
};

struct element* Allouer(){
    return (struct element*) malloc(sizeof(struct element));
}
void liberer(struct element *P){
    free(P);
}
int Valeur(struct element *P){
    return (*P).val;
}
struct element* Suivant(struct element *P){
    return (*P).suiv;
}
void AffVal(struct element *P, int v){
    (*P).val = v;
}
void AffSuiv(struct element *P, struct element *Q){
    (*P).suiv = Q;
}
```

Les listes

Parcours

- **Accès par valeur:** il s'agit de rechercher (séquentiellement à partir de la tète) une valeur v dans la liste.
- **Accès par position:** il s'agit de rechercher (séquentiellement à partir la tête) l'élément (son adresse) qui se trouve à une position donnée. La position est le numéro d'ordre du maillon dans la liste.

Les listes

Mise à jour

- **Constructeur d'une liste** à partir de n valeurs données.
- **Insertion d'une valeur v à une position donnée i:** allouer un nouveau élémnt contenant v et l'insérer dans la liste à la i ième position.
- **Insertion d'une valeur v dans une liste ordonnée:** allouer un nouveau élément contenant v et l'insérer dans la liste de telle sorte que la liste reste ordonnée.

Les listes

Mise à jour

- **Suppression du maillon se trouvant à une position donnée:** rechercher par position et le libérer
- **Suppression d'une valeur dans la liste:** recherche par valeur et supprimer le maillon trouvé en metton à jour le précédent (s'il existe).
- **Destruction de tous les maillons d'une liste.**

Les files d'attente

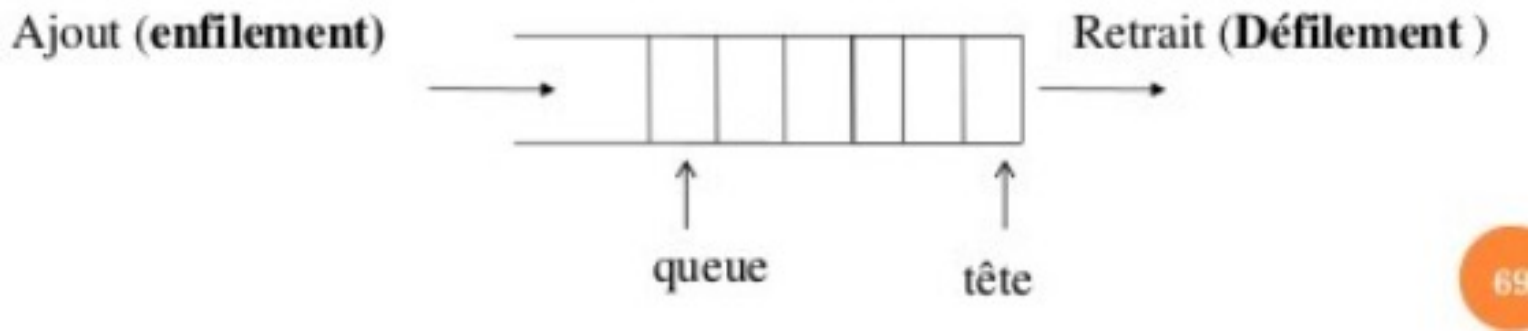
Définition

- est une structure de données basée sur le principe « premier arrivé, premier sorti »
- Le fonctionnement ressemble à une file d'attente : les premiers personnes à arriver sont les premiers personnes à sortir de la file

Les files d'attente

Définition

- Est une collection d'éléments dans laquelle tout nouvel élément est inséré à la fin (queue) et tout élément ne peut être supprimé que du début (tête)



Les files d'attente

Utilisation

- Les buffers: gérer des objets qui sont en attente d'un traitement ultérieur
  - La gestion des documents à imprimer
  - Des programmes à exécuter
  - Des messages reçus
- Elle sont utilisées également dans le parcours des arbres.

Les files d'attente

Modèle

- Les opération habituelles sur les files:
  - Initialisation
  - Vérification du contenu (vide ou pleine)
  - Enfilement (ajout)
  - Défilement (retrait)

Opération	Role
Initfile(F)	Créer une file vide
Enfiler(F,Val)	Ajouter Val à la queue de la file
Défiler(F,Val)	Retirer dans Val l'élément en tête de file
FileVide(F)	Tester si la file est vide
FilePleine(F)	Tester si la file est pleine

Les files d'attente

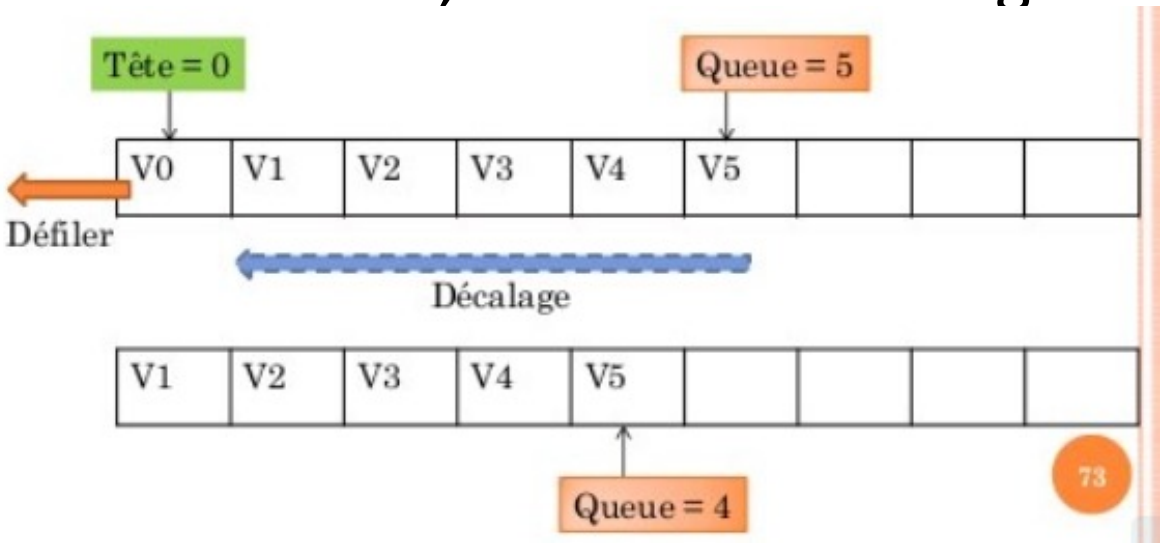
Implémentation

- Les files d'attente peuvent être présentées en deux manières:
  - Statique en utilisant les tableaux
  - Dynamique en utilisant les listes linéaires chainées

Les files d'attente

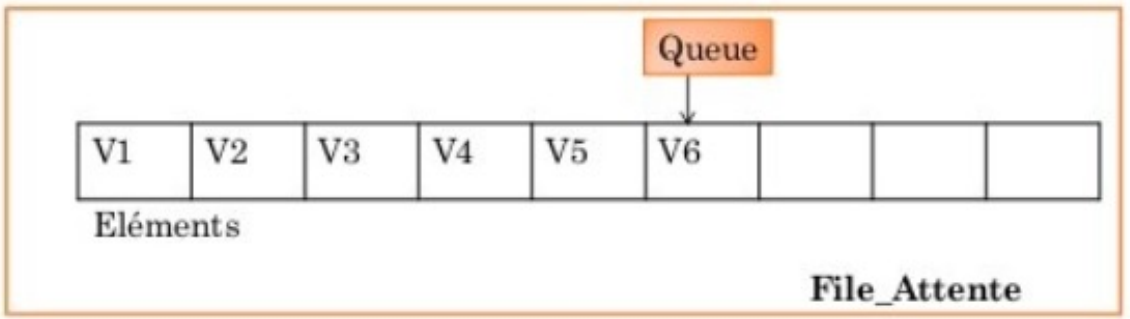
Implémentation Statique

- L'implémentation statique peut être réalisée par:
  - Décalage en utilisant un tableau avec une tête fixe toujours à 0, et une queue variable
  - Tableaux circulaire où la tête et la queue sont toujours les deux variables
- À chaque défilement, on fait un décalage



Les files d'attente

Implémentation Statique



Définition de la structure

```
TYPE file_attente = STRUCTURE
    Elements : TABLEAU[MAX] de typeq
    Queue : ENTIER
FIN
Var F:file_attente
```

Les files d'attente

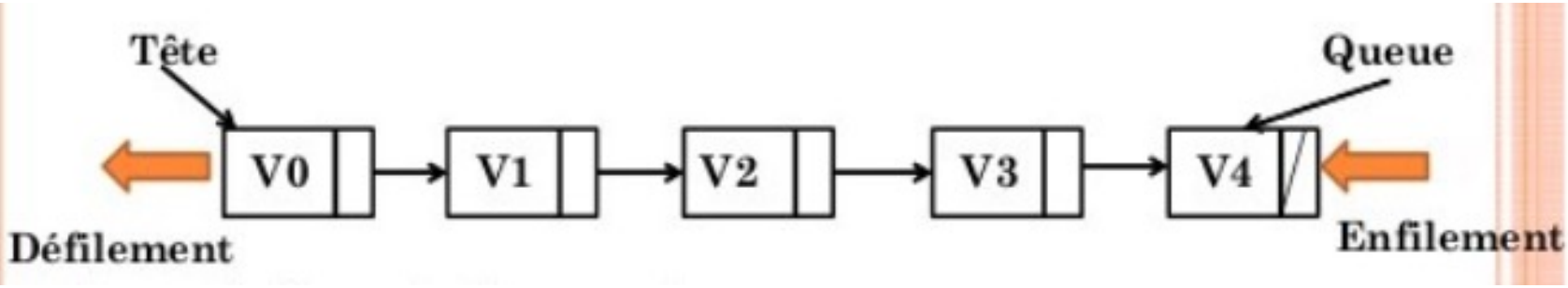
Implémentation statique par décalage

Opération	Rôle
Initfile(F)	$F.Queue \leftarrow -1$
FileVide(F)	$Retourner\ (F.Queue = -1)$
FilePleine(F)	$Retourner\ (F.Queue = Max - 1)$
Enfiler(F,Val)	$Si\ \big(Non\ Filevide(F)\big)$ $\quad F.Queue \leftarrow F.Queue + 1$ $\quad F.Elements[F.Queue] \leftarrow X$
Défiler(F,Val)	$Si\ \big(Non\ Filevide(F)\big)$ $\quad X \leftarrow F.Elements[0]$ $\quad Pour\ I \leftarrow 0$ à $F.Queue - 1$ $\quad\quad F.Elements[I] \leftarrow F.Elements[I + 1]$ $\quad F.Queue \leftarrow F.Queue - 1$

Les files d'attente

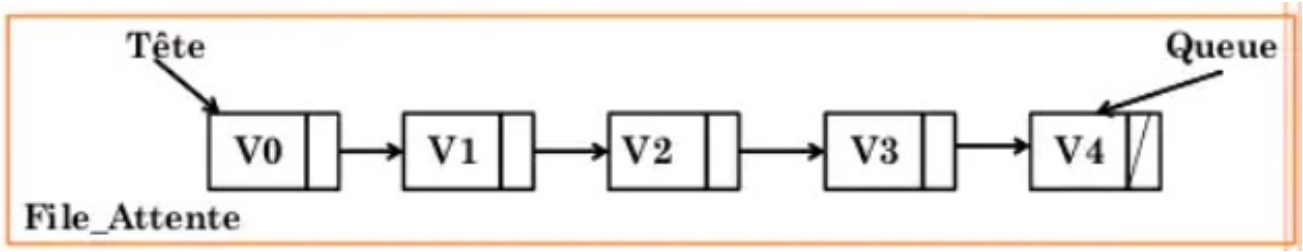
Implémentation Dynamique

- La représentation dynamique utilise une liste linéaire chaînée.
  - L'enfilement se fait à la queue de la liste.
  - Le défilement se fait de la tête de la liste.
  - La file d'attente peut devenir vide, mais ne sera jamais pleine.



Les files d'attente

Implémentation Dynamique



Définition de la structure

```
TYPE Maillon = STRUCTURE
    val: tupeq
    suiv:*Maillon
FIN
Type File_Attente = STRUCTURE
     Tête, Queue: *Maillon
FIN
Var F : File_Attente
```

Les files d'attente
Implémentation Dynamique

Opération	Rôle
Initfile(F)	$F.Tete \leftarrow NIL;\ F.Queue \leftarrow NIL$
FileVide(F)	$Retourner\ (F.Tete = NIL)$
Enfiler(F,X)	$Allouer(P); AffVall(P,X); AffSuiv(P,NIL);$ $Si\ \big(Non\ Filevide(F)\big)$ $\quad AffSuiv(F.Queue,P)$ $SINON$ $\quad F.Tete \leftarrow P$ $\quad F.Queue \leftarrow P$
Défiler(F,X)	$Si\ \big(Non\ Filevide(F)\big)$ $\quad P \leftarrow F.Tete$ $\quad X \leftarrow Valeur(F.Tete)$ $\quad F.Tete \leftarrow Suivant(F.Tete)$ $\quad Liverer(P)$