STRUCTURES DE DONNÉES
MIP — S4

Pr. K. Abbad & Pr. Ad. Ben Abbou & Pr. A. Zahi
Département Informatique
FSTF
2019-2020

Séance 6 - Les listes chainées

- Définition
- Opérations de base
- Représentation chainée(non contigüe)

Liste chainée — *Définition*

- Notion intuitive — *Liste chainée d'objets*
  - Une **liste chainée** est une **structure** destinée au stockage des données en cours de traitement par un programme.
  - On utilisera une liste pour stocker un nombre indéterminé
  - L'ordre pourra dépendre :
    - de la chronologie d'insertion des éléments
    - de la valeur des éléments insérés
    - d'autres critères peuvent être fixés par l'utilisateur

Liste chainée — *Opérations de base*

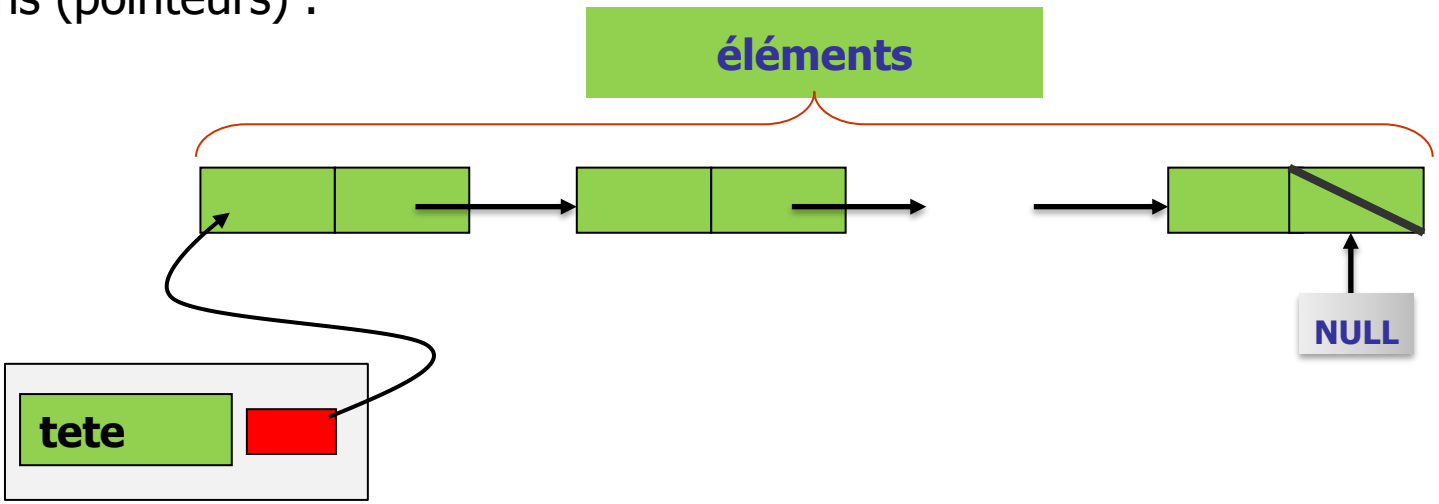
- **initListe** — *Crée une liste vide.*
- **estListeVide** — *Détermine si une liste est vide ou non.*
- **longListe**— *récupère le nombre des éléments de la liste chainée.*
- **afficherListe** — affiche *des éléments de la liste chainée.*
- **Recherche** — *recherche une valeur dans la liste chainée.*
- **Prédécesseur** — *détermine le prédécesseur d'un 'élément donné*

Liste chainée — *Opérations de base*

- **insererTete** — *insère un élément en tête de la liste chainée .*
- **insererQueue** — *insère un élément en queue de la liste chainée .*
- **insererMaillonOrd** — *insérer un élément à une position donnée dans la liste chainée .*
- **suprimerTete** — *supprime un élément situé en tête de la liste chainée .*
- **suprimerQueue** — *supprime un élément situé en queue de la liste chainée*
- **suprimerMaillonPos** — *supprime un élément situé à une position donnée dans la liste chainée .*

Liste chainée — *Représentation*

- Une Liste chainée est une suite de maillons distribués et organisés séquentiellement.
- Un maillon possède :
  - les données d'un élément.
  - un lien (un pointeur) vers son successeur.
- La Liste chainée est identifiée par sa tête (adresse du premier maillon)
- Le pointeur de dernier maillon est à NULL.
- Les opérations sur la liste chainée sont basées sur la manipulation des liens (pointeurs) .

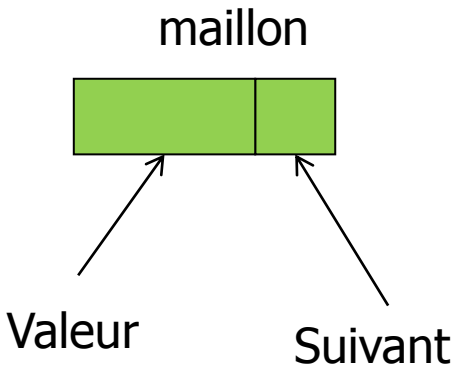


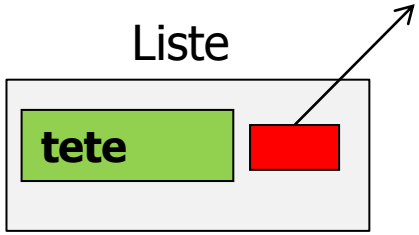
Liste chainée — *Représentation*

- Définition d'une liste chainée d'entiers

```
/* type maillon */
   typedef struct Maillon{
         int valeur;
         struct Maillon * suivant;
    } maillon;


/* type listechainee */
typedef struct Maillon* LISTE;
```





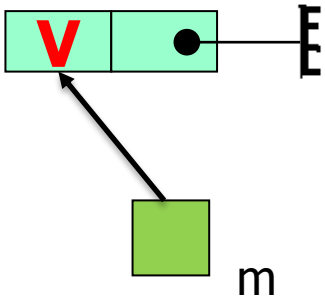

```
LISTE tete;
```

tete : pointe sur le premier élément de la liste chainée

Liste chainée — *Représentation*

- Créer maillon d'une liste chainée d'entiers
  - Réserver l'espace mémoire pour l'élément
  - Remplir les champs du maillon : valeur par V et le pointeur par NULL
  - La fonction retourne l'adresse de l'espace réservé et NULL si l'espace n'est pas réservé

```
maillon* creerMaillon(int V)
{
  maillon *m;
  m=(maillon *)malloc(sizeof(maillon);
     if(m!=NULL){
                m->valeur = V;
                m->suivant = NULL;
     }
        return m;
  }
```



Liste chainée — *Initialisation*

- Initialisation de la liste chainée
  - la tête de la liste est mise à NULL.
  - la fonction retourne L de type Liste.

```
LISTE InitListe()
{
        LISTE tete;
        tete=NULL;
        return tete;
}
```

Liste chainée — *Liste vide ?*

- Liste vide ?
  - *Teste si la tête de liste est mise à NULL*
  - *La fonction retourne 1 si la liste est vide et 0 sinon*

```
int EstListeVide(LISTE tete)
{
      if(tete== NULL  )
            return 1;
      return 0;
 }
```

Liste chainée — *Taille de liste chainée*

- Calculer la taille de liste chainée – méthode itérative
  - Parcourir des éléments de la liste jusqu'à la fin de la liste et à chaque passage d'un élément à l'autre on incrémente par 1.
  - La fonction retourne 0 si la liste est vide.

```
int longListe(LISTE tete)
{
        maillon* m;
        int nbr=0;
        m=tete;
        while(m!=NULL){
             nbr++ ;
              m=m->suivant;
        }

        return nbr;
}
```

Liste chainée — *Taille de liste chainée*

- Calculer la taille de liste chainée - méthode récursive

```
int longListe(LISTE tete)
{
        if (EstListeVide(tete)==1)
                return 0;
        else
                return 1+ longListe (tete->suivant);
}
```

  - Cas de base: liste vide taille(Liste)=0
  - Cas général : taille(Liste)= 1+ taille(Liste\{tête})

Liste chainée — *Affichage de liste chainée*

- Afficher les éléments d'une liste chainée
  - parcourt des éléments de la liste en commençant par la tète jusqu'à la fin de la liste,
  - affiche la valeur de chaque l'élément , puis passe au suivant jusqu' au dernier élément

```
void afficherListe(LISTE tete)
{
        maillon* m;
        m=tete;
        while(m!=NULL){
             printf("%d",m->valeur);
              m=m->suivant;
        }
}
```

Liste chainée — *Recherche*

- Recherche un élément dans une liste chainée
  - la fonction retourne l'adresse du maillon si la valeur existe et NULL sinon

```
maillon* Recherche(LISTE tete , int V)
{
        maillon* par;
        par=tete;
        while(par!=NULL){
                if(par->valeur==V)
                        return par;

                par=par->suivant;
        }
        return NULL;
}
```

Liste chainée — *Prédécesseur*

- Récupérer le prédécesseur d'un élément donné dans une liste non vide
  - Le prédécesseur du 1ere élément (tete) est NULL

```
maillon* Predecesseur (LISTE tete ,int v){
   maillon *pred=NULL;
   maillon *ptr=tete ;
   while(ptr!=NULL && ptr->valeur!=V)
   {
                        pred=ptr;
                        ptr=ptr->suivant;
   }
   if(ptr!= NULL)
                return pred;
   else
                return NULL;
}
```

Cette fonction retourne **NULL** dans deux cas:

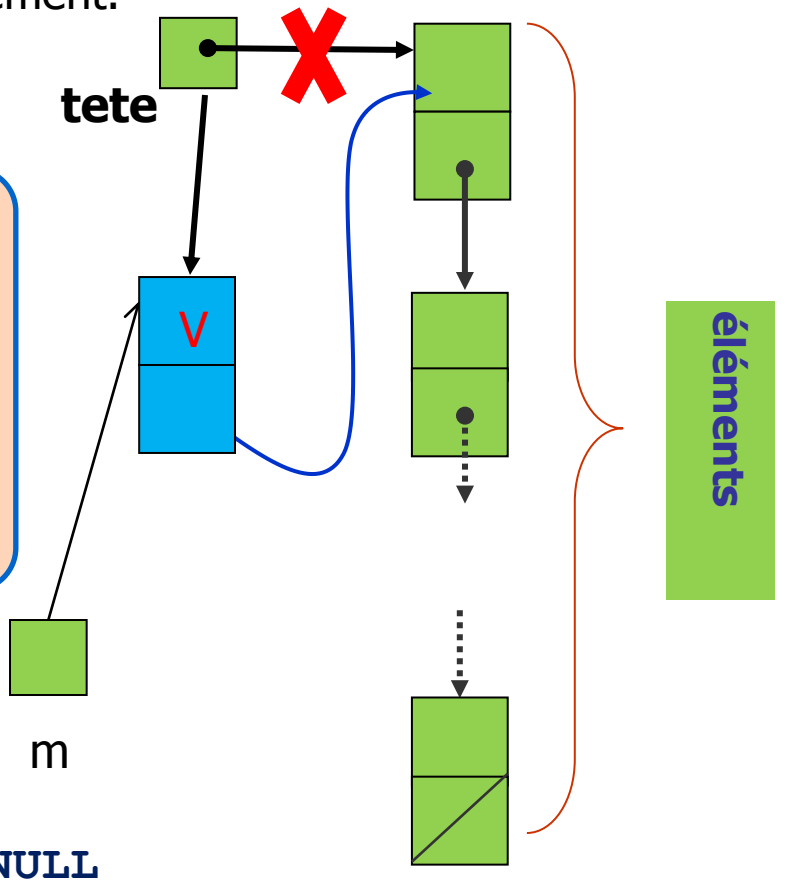
1) l'élément v existe dans le premier maillon

2) L'élément v n'existe pas dans la liste

Liste chainée — *Insertion en tête*

- Insérer un élément en tête de la liste chainée
  - créer un maillon avec la valeur de l'élément.
  - ajouter l'élément à la tête de la liste .

```
maillon* m;
m = creerMaillon(V) ;
If(m!=NULL){
        m->suivant = tete;
        tete = m;
}
```



Remarque:

Si la liste est vide `m->suivant = tete = NULL`

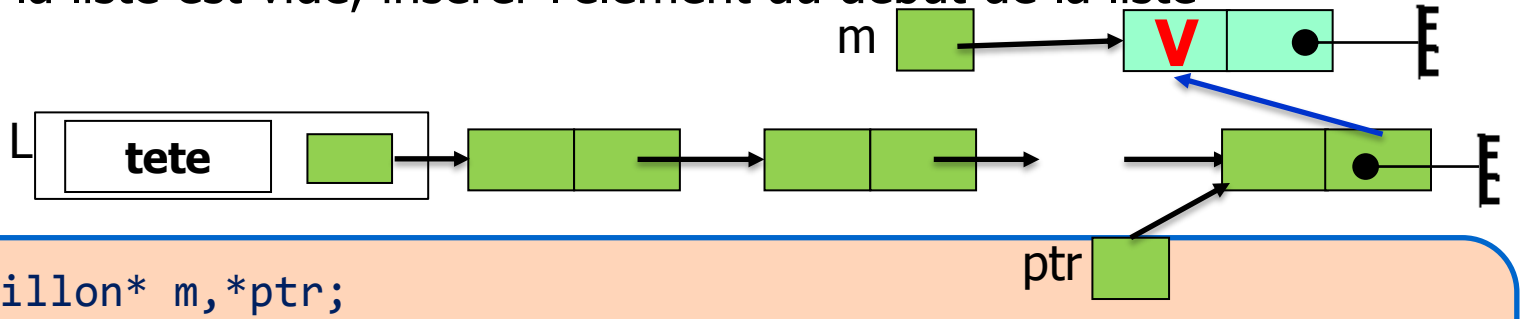
Liste chainée — *Insertion en tête*

- Insérer un élément en tête de la liste chainée
  - La fonction retourne 1 si la valeur est insérée et 0 sinon

```
int insererTete (Liste *tete, int V)
{
        maillon* m;
        m = creerMaillon(V) ;
        if(m!=NULL){
                m->suivant = *tete;
                *tete =m;
               return 1;
        }
        return 0;
  }
```

Liste chainée — *Insertion en queue*

- Insérer un élément V en queue de la liste chainée
  - créer un maillon avec la valeur de l'élément,
  - Cherche le dernier élément dans la liste chainée
  - Lier le nouveau maillon à la queue de la liste
- Si la liste est vide, insérer l'élément au début de la liste




```
maillon* m,*ptr;
m = creerMaillon(V)
if(!EstListeVide(tete))   { // recherche le dernier élément
        ptr=tete ;
        while(ptr->suivant!=NULL)
                ptr=ptr->suivant;
        ptr->suivant=m; // lier le maillon créé avec la queue
}
```

Liste chainée — *Insertion en queue*

- Insérer un élément V en queue de la liste chainée
  - La fonction retourne 1 si la valeur est insérée et 0 sinon

```
int insererQueue(Liste *tete, int V) {
     maillon* m,*ptr;
     m = creerMaillon(V)
    if(m!=NULL) {
        if(!EstlisteVide(*tete)){
                ptr=*tete ;
                while(Ptr->suivant!=NULL)  ptr=ptr->suivant;
                ptr->suivant =m;
         }else   *tete =m;
         return 1;
     }
     return 0;
}
```

La tête de la liste sera modifiée dans le cas liste vide →passage par adresse

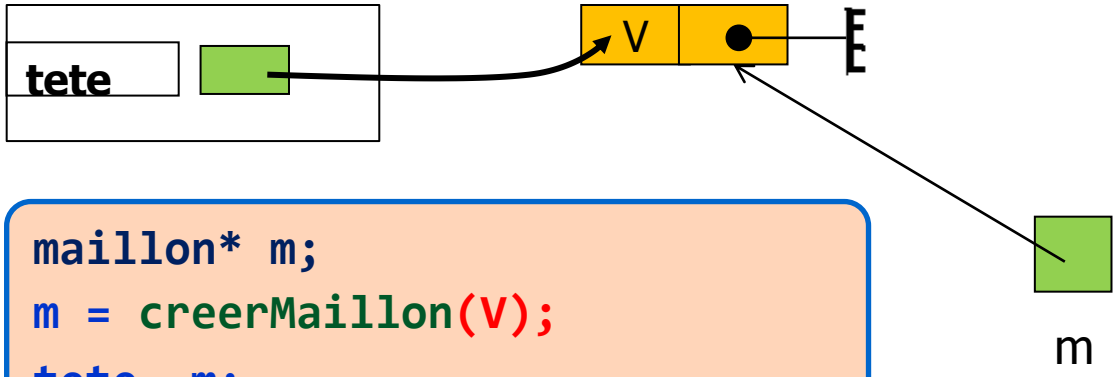
Liste chainée — *Insertion ordonnée*

- Insérer un élément V dans une liste chainée ordonnée
  - créer un maillon avec la valeur de l'élément V
  - Insérer l'élément dans la liste de telle façon que le nouveau chainage soit ordonné

- **Trois cas possibles**
  - Liste vide
  - Insérer au début
  - Insérer à une position donnée

Liste chainée — *Insertion ordonnée*

- Insérer un élément V dans une liste chainée ordonnée
- **Cas : Liste vide**
  - créer un maillon avec la valeur de l'élément,
  - Insérer l'élément dans la liste

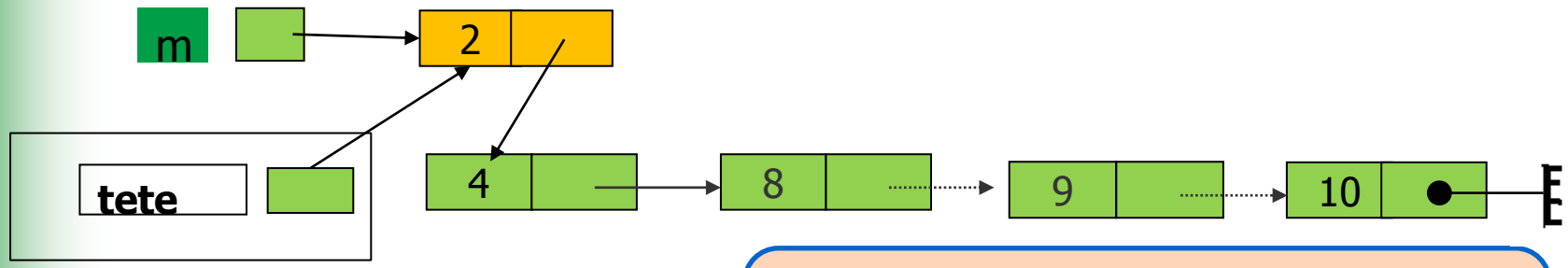



```
maillon* m;
m = creerMaillon(V);
tete =m;
```

La tête sera modifiée dans le cas liste vide (passage par adresse)

Liste chainée — *Insertion ordonnée*

- Insérer un élément V dans une liste chainée ordonnée
- **Cas : Liste non vide avec insertion au début**
  - créer un maillon avec la valeur de l'élément,
  - Insérer l'élément dans la liste

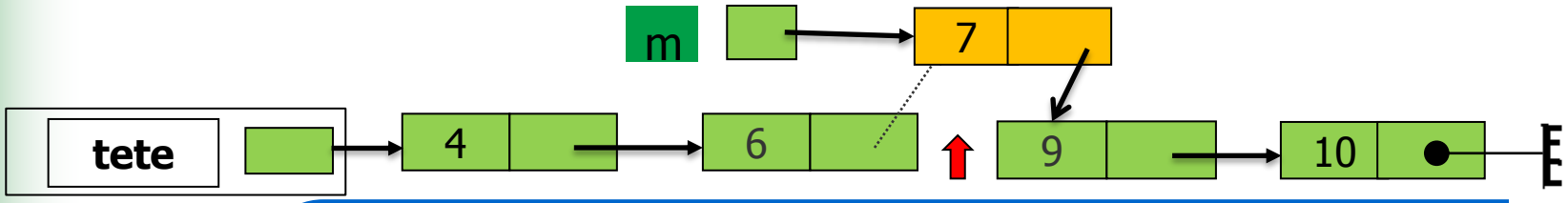



```
maillon* m;
m = creerMaillon(V)
m->suivant = tete
tete = m;
```

La tête sera modifiée (passage par adresse)

Liste chainée — *Insertion ordonnée*

- Insérer un élément V dans une liste chainée ordonnée
- **Cas : Liste non vide avec insertion à une position donnée**



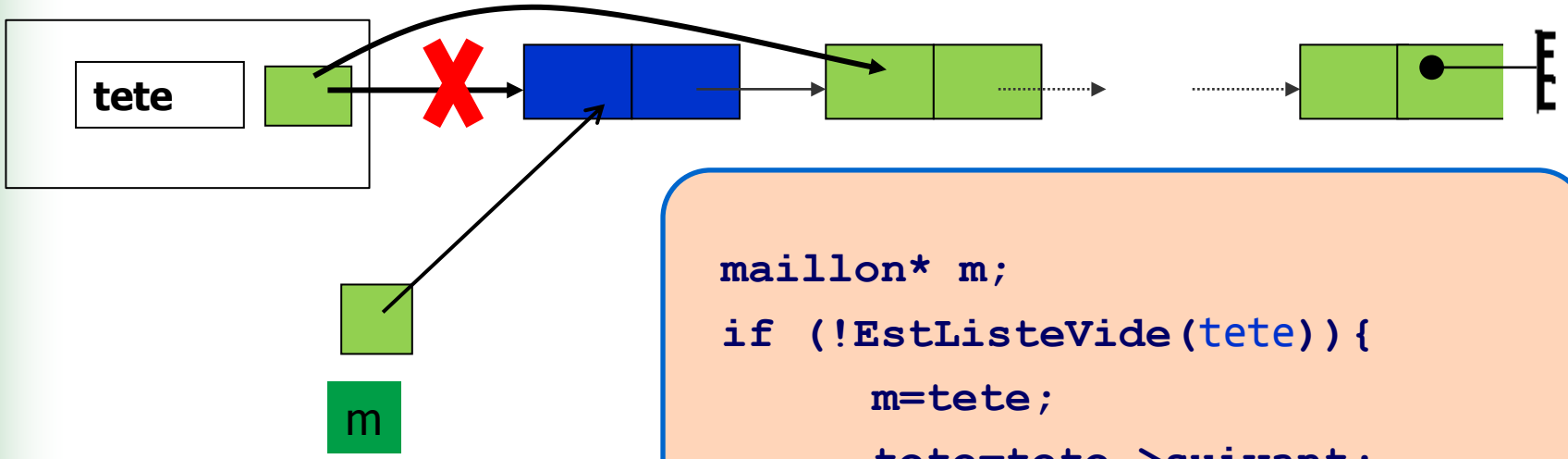

```
maillon* m,*ptr,*par;
m = creerMaillon(V);
par=tete ;
while(par!=NULL && par->valeur<V){/*chercher la position */
        ptr= par;
        par = par->suivant;
}
m->suivant = ptr->suivant;
ptr->suivant = m;
```

Liste chainée — *Insertion ordonnée*

```
int InsererMaillonOrd(Liste *tete, int V) {
     maillon* m,*ptr,*par;
     m = creerMaillon(V)
    if(m!=NULL) {
    if(EstListeVide(*tete)){*tete =m; // liste vide
    } else { if(*tete->valeur>V) { // inserer debut
                        m->suivant = *tete ; *tete = m;
             }else {     par=*tete;
                          while(par!=NULL && par->valeur<V){
                                 ptr= par; par = par->suivant;
                          }
                          m->suivant = ptr->suivant;
                          ptr->suivant = m;
                    }
           }
      return 1;
     }
    return 0;
}
```

Liste chainée — *Suppression en tête*

- Supprimer un élément situé en tête de la liste chainée




```
maillon* m;
if (!EstListeVide(tete)){
        m=tete;
        tete=tete->suivant;
        free(m);
}
```

Remarque:

Si la liste chainée est vide on fait rien

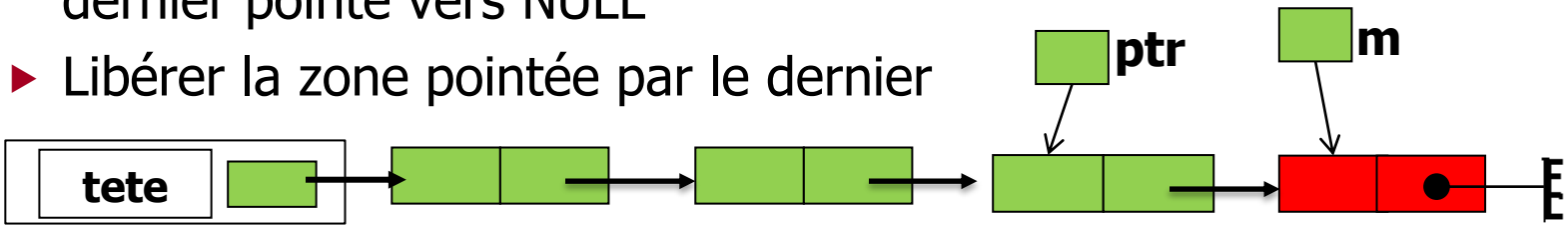
Liste chainée — *Suppression en tête*

- Supprimer un élément situé en tête de la liste chainée.
  - créer un pointeur m, tel que m et tete pointent sur la même zone,
  - Pointer la tête sur la zone pointée par le suivant de tête ,
  - Libérer la zone pointée par m ,
  - La fonction retourne 1 si l'élément est supprimé et 0 sinon

```
int SupprimerTete(Liste *tete){
  maillon* m;
  if (!EstlisteVide(*tete)){
        m=*tete;
        *tete=*tete->suivant;
        free(m);
        return 1;
  }
  return 0;
}
```

Liste chainée — *Suppression en queue*

- Supprimer un élément en queue de la liste chainée, la liste contient un deux éléments au moins
  - Chercher le dernier et l'avant dernier ,Le suivant de l'avant dernier pointe vers NULL
  - Libérer la zone pointée par le dernier




```
maillon *m,*ptr;
if (!EstListeVide(tete)){
        ptr=tete;
        while(ptr->suivant->suivant!=NULL)
            ptr=ptr->suivant;
        m=ptr->suivant;
        ptr->suivant=NULL;
        free(m);
}
```

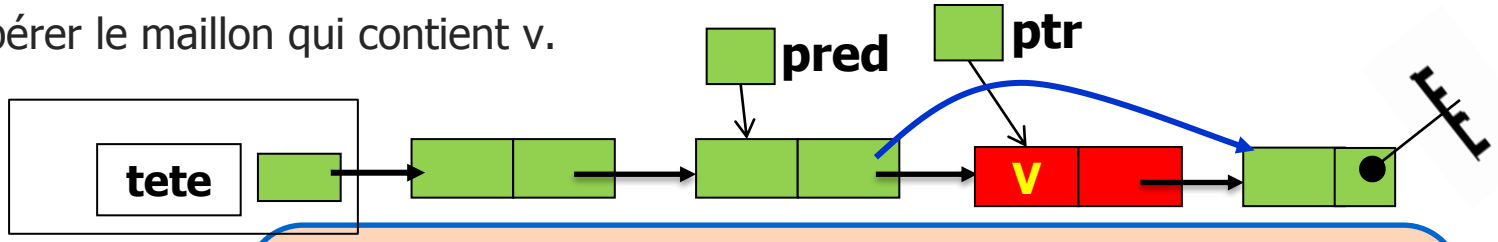
Liste chainée — *Suppression en queue*

- Supprimer un élément situé en queue de la liste chainée

```
int supprimerQueue(LISTE *tete) {
    maillon* m,*ptr;
    if(!EstlisteVide(*tete))
      if(*tete->suivant!=NULL)){/*plus de 2 elts dans la liste*/
            ptr=*tete;
            while(ptr->suivant->suivant!=NULL)
                 ptr=ptr->suivant;
            m=ptr->suivant;
            ptr->suivant=NULL;
            free(m); return 1;
      }
      else { /* 1 seul element dans la liste */
                m=*tete;
                *tete=NULL;
                free(m); return 1;
       }
    return 0;
}
```

Liste chainée — *Suppression Elément donné*

- Supprimer un élément donné **V** dans la liste chainée.
  - Chercher l'élément qui contient la valeur V **(ptr)** et son prédécesseur **(pred).**
  - Le suivant du prédécesseur pointe vers le suivant du maillon qui contient V.
  - Libérer le maillon qui contient v.




```
maillon *ptr,*pred;
if (!EstListeVide(tete)){
        ptr=tete;
         Pred=NULL;
         While(ptr -> valeur!=V && ptr!=NULL) {
             Pred= prt;
             ptr=ptr->suivant;
        }
        pred->suivant=ptr->suivant;
        free(ptr);
}
```

Cas particuliers:

Si V existe dans le premier maillon → **supprimerTete**

Si V n'existe pas, on ne fait rien

Liste chainée — *Suppression Elément donné*

- Supprimer un élément donné dans liste chainée.

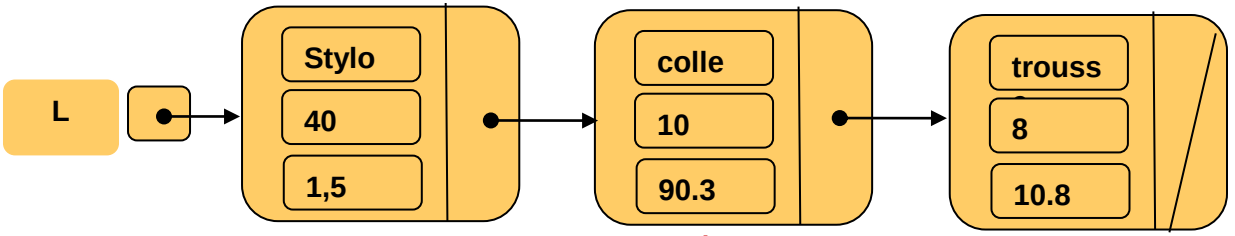
```
int supprimerMaillonPos(LISTE *tete,int v) {
     maillon *ptr,*pred=NULL;
     if(!EstListeVide(*tete)) {
        if (*tete->valeur==v){/*v existe dans le premier maillon*/
            supprimerTete(tete);
            return 1;
        }
        ptr=*tete;
        While(ptr -> valeur!=V && ptr!=NULL) {
               Pred= prt;
               ptr=ptr->suivant;
         }
         if(ptr!=NULL){
               pred->suivant = ptr->suivant;
               free(ptr);
               return 1;
         }
         Return 0;
     }
     return 0;
}
```

Liste chainée — *Suppression Elément donné*

```
int supprimerMaillonPos(LISTE *tete,int v) {
     maillon *pred,*m;
    if(!EstListeVide(*tete)) {
        pred=predecesseur(*tete,v);
        if(pred==NULL)
           if (*tete->valeur==v){/*v existe dans le premier
    maillon*/
               supprimerTete(tete);
               return 1;
            }
            else
                return 0;
       else{ m=Pred->suivant;
             pred->suivant = m->suivant;
             free(m);
             return 1;
        }
     }
     return 0;
}
```

Exercice1

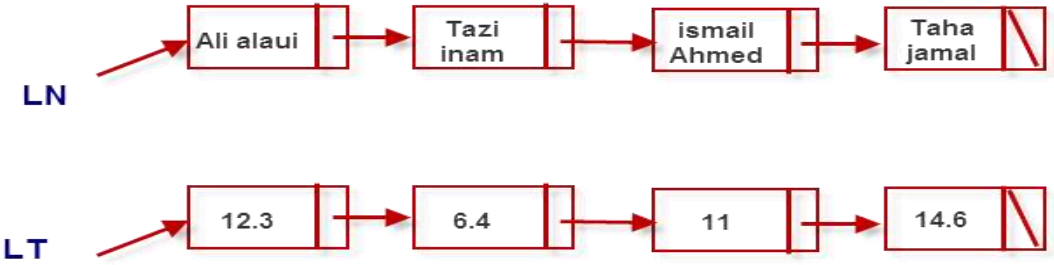
- Lors d'achat de produits, le client enregistre les informations relatives aux différents produits achetés dans une liste chaînée .
- Un produit est caractérisé par les informations suivantes :
  - Désignation (chaine de caractères),
  - Quantité en stock (entier)
  - Prix unitaire (réel)
- Ecrire un programme C qui :
  - Déclare les structures nécessaires.
  - Initialise la liste.
  - Saisit la liste.
  - Calcule et affiche le montant global dépensé lors de l'achat de produits.
  - Afficher le prix du produit le moins cher.
  - Libérer toute la mémoire occupée par une liste.



Exercice2

Etant données deux listes chaînées LN et LT contenant respectivement, les noms des étudiants et les moyennes générales du semestre S3.

- Ecrire un programme C qui :
  - Déclare les structures nécessaires
  - Initialise les deux listes
  - Saisit les deux listes
  - Vérifie que les deux listes ont le même nombre d'éléments
  - affiche les noms des étudiants qui ont validé le semestre S3
  - calcule le nombre des étudiants qui n'arrivent pas à valider le semestre S3



Exercice 3

- Ecrire un programme C qui saisit un entier n et l'affiche à l'envers. Le programme doit utiliser une pile pour stocker les différents chiffres de n.
- **Exemple**

  n = 4596