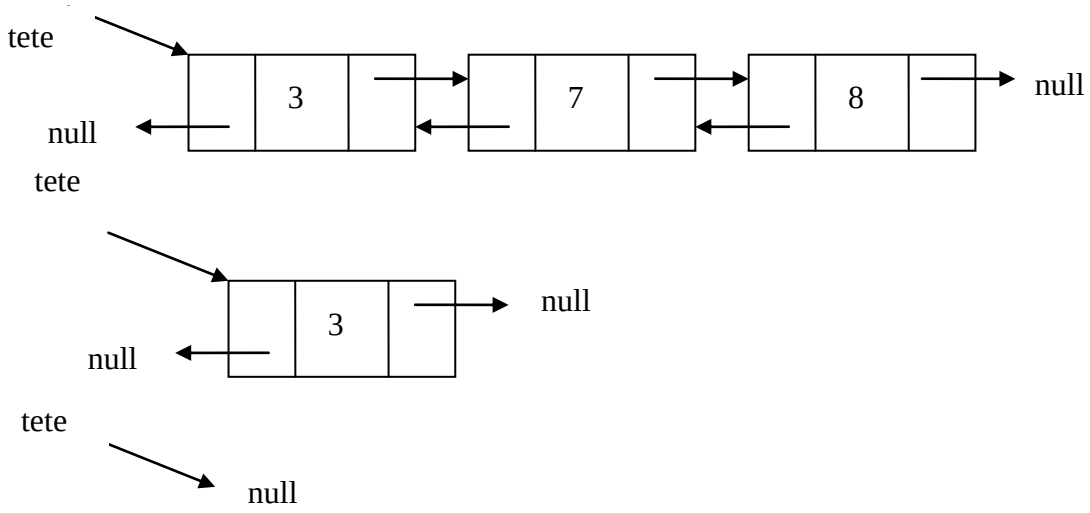
Liste doublement chainée

Solution :

Une liste doublement chaînée est une liste dont chaque cellule, en plus de pointer sur la cellule suivante, comme dans une liste simplement chaînée, dispose également d'un pointeur sur la cellule précédente.
La première cellule de la liste pointe sur la valeur précédente null, la dernière cellule pointe sur la valeur suivante null.
Suivent ci-après un ensemble d'exemples de listes doublement chaînées :

- *Une liste contenant 3 cellules*
- *Une liste contenant une seule cellule*
- *Une liste vide*



Nous allons nous intéresser dans le cadre de cet exercice à la suppression d'un élément dans une liste doublement chaînée ordonnée.

Question 1 *:*
Déclarer les types nécessaires à l'utilisation d'une liste doublement chaînée.

Type ptelement = ^ element
Type element = **Structure**
val : Entier
precedent : ptelement
suivant : ptelement
finstructure

Question 2 *:*
Citer différents cas de figures de positions remarquables de cellule à supprimer et qui vont devoir attirer notre attention dans la procédure de suppression.

Lors de la suppression d'un élément dans une liste, il faut être vigilant et surveiller différents cas de figures :

- La cellule que l'on veut supprimer peut ne pas exister dans la liste.
- La cellule peut être au cœur de la liste (elle aura une cellule précédente et une cellule suivante)

- La cellule peut être en première position.
- La cellule peut être en dernière position.

Attention,

- La liste elle-même peut être vide au moment de l'appel de la procédure.

Question 3 *:*
Pour chacun de ces cas, expliquer textuellement ce que la procédure de suppression devra réaliser.

- **La cellule que l'on veut supprimer peut ne pas exister dans la liste.**

Ce cas de figure correspond à un parcours de liste amenant à un pointeur suivant à vide ou à une valeur de cellule supérieure à la valeur recherchée (la liste étant triée dans l'ordre croissant).
Dans ce cas, la procédure s'achève sans action de suppression avec éventuellement un message d'alerte. On pourrait, au mieux, prévoir que la procédure renvoie un booléen indiquant si la cellule a été trouvé ou non.

- **La cellule peut être au cœur de la liste (elle aura une cellule précédente et une cellule suivante)**

Ce cas de figure correspond à un parcours de liste amenant à un pointeur sur la cellule contenant l'élément à supprimer. Cet élément possède des pointeurs précédent et suivant non null.
Dans ce cas, on doit faire pointer le pointeur « suivant » de la cellule précédent celle à supprimer sur la cellule suivant celle à supprimer et faire pointer le pointeur « précédent » de la cellule suivant celle à supprimer sur la précédente.
On doit restituer l'espace occupée par la cellule.

- **La cellule peut être en première position.**

Ce cas de figure correspond à un parcours de liste amenant à un pointeur sur la cellule contenant l'élément à supprimer. Cet élément possède un précédent pointant sur null et un suivant ne pointant pas sur null.
Dans ce cas, on doit faire pointer la tête de liste sur la cellule suivant celle à supprimer et la cellule de tête aura un pointeur « précédent » positionné sur null.
On doit restituer l'espace occupée par la cellule.

- **La cellule peut être en dernière position.**

Ce cas de figure correspond à un parcours de liste amenant à un pointeur sur la cellule contenant l'élément à supprimer. Cet élément possède un suivant pointant sur null et un précédent pointant sur une cellule.
Dans ce cas, on doit faire pointer le pointeur « suivant » de la cellule précédent celle à supprimer sur null.
On doit restituer l'espace occupée par la cellule.

Attention,

- **La liste elle-même peut être vide au moment de l'appel de la procédure.**

Cas à rapprocher du premier cas.
Dans ce cas, la procédure s'achève sans action de suppression avec éventuellement un message d'alerte. On pourrait, au mieux, prévoir que la procédure renvoie un booléen indiquant si la cellule a été trouvé ou non.

Question 4 *:*
Soit la procédure de suppression « supprimer_cellule », équipée de deux paramètres : l'adresse de tête de liste et l'élément entier à supprimer.
Ecrire l'entête de cette procédure et le commentaire qui l'accompagne.

Procédure supprimer_cellule (E/S tete : ptelement, x : entier)
// Procédure chargé de supprimer la cellule contenant l'entier x dans une liste doublement
// chaînée ordonnée (croissant).
// il faut restituer le chaînage des cellules autour
// Si la cellule n'est pas trouvée, un message sera affichée
// On pourra faire évoluer cette procédure en lui adjoignant un troisième paramètre en Sortie
// ok qui indiquera si l'opération s'est bien déroulée et lui affectera faux en lieu et heure
// du message d'erreur.

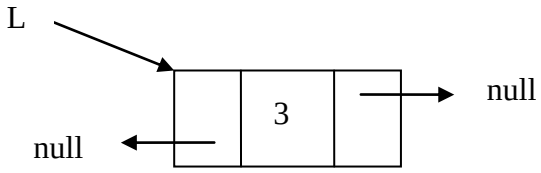
Question 5 *:*
Ecrire le corps de cette procédure.

```
Déclarations
        p : ptelement
        fin, trouvé : booléen
Début
trouvé ← faux
fin ← (tete = null)
p ← tete
Tant que non fin faire
        Si p^.val = x alors
                Si p^.precedent = null alors
                        Si p^.suivant = null alors      //on supprime la seule cellule en liste
                                tete ← null
                        Sinon                           //on supprime la cellule de tete
                                p^.suivant^.precedent ← null
                                tete ← p^ .suivant
                        Finsi
                Sinon
                        Si p^.suivant = null alors      //on supprime une cellule de fin de liste
                                p^.precedent^.suivant ← null
                        Sinon                           //cellule de cœur de liste
                                p^.suivant^.precedent ← p^.precedent
                                p^.precedent^.suivant ← p^.suivant
                        Finsi
                Finsi
                désallouer(p)  // restituer l'espace occupé par la cellule supprimée du chaînage
                trouvé ← vrai //la cellule a été trouvé et supprimée
                fin ← vrai
        Sinon
                p← p^.suivant //on avance sur la cellule suivante
                fin ← (p = null) // on arrête la boucle : dernière cellule
                Si non fin alors
                        fin ← (p^.val>x)        // on doit arrêter immédiatement, on ne trouvera
                Finsi                           // pas x , mais trouvé reste bien à faux
                // attention, on ne peut évaluer dans la même condition une liste vide
                // et la valeur contenue à l'intérieur. En effet, si la liste est vide l'évaluation
                // générera une erreur => d'où la séparation des tests
        Finsi
Fin Tant que
```

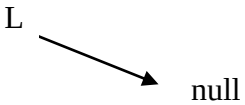
Si non trouvé **alors**
    Afficher (« élément non présent dans la liste »)
Finsi
Fin Procédure

<u>Question 6 (question subsidiaire mais recommandée)</u>
Tester la procédure selon les cas suivants :

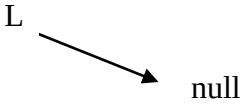
1. Supprimer_cellule (L,3)



On appelle la procédure avec L qui transmet sa valeur à tete
On passe par les différentes actions suivantes :
trouvé ← faux
fin ← (tete = null) => la tête n'est pas null
p ← tete
Tant que non fin **faire**
    Si p^.val = x **alors** => notre valeur x correspond
        Si p^.precedent = null **alors**
            Si p^.suivant = null **alors** => precedent et suivant sont null
                tete ← null
            Finsi
        Désallouer(p)
        trouvé ← vrai
        fin ← vrai => on arrête la boucle
Fin Tant que
Fin Procédure
L étant déclarée E/S, on récupère bien sa nouvelle valeur et on obtient après l'appel :



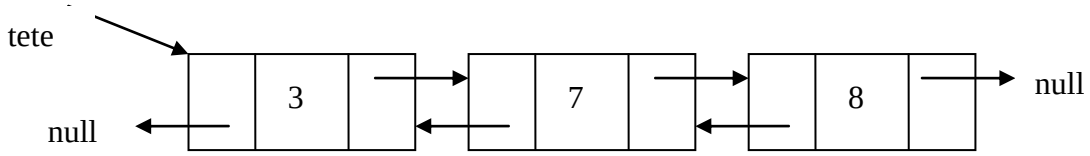
2. Supprimer_cellule (L,3)



On appelle la procédure avec tete qui rend donc la valeur null
trouvé ← faux
fin ← (tete = null) => fin passe à vrai (la liste est vide)
p ← tete
Tant que non fin **faire** => on ne rentre pas dans la boucle
Fin Tant que
Si non trouvé **alors**
    Afficher (« élément non présent dans la liste »)
Finsi

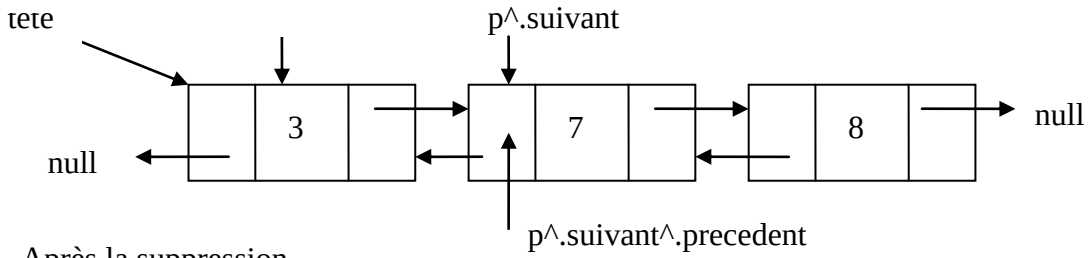
Fin Procédure
On affiche bien le message : l'élément n'a pas été trouvé et L garde sa valeur initiale

3. Supprimer_cellule (L,3)

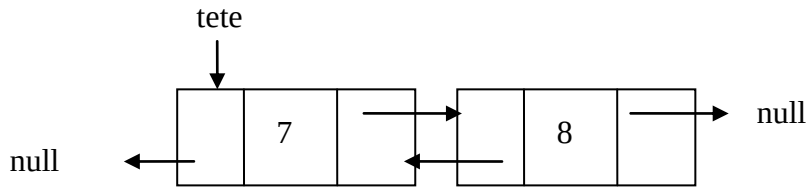


On appelle la procédure avec l'adresse de début de liste L transmis à tete
trouvé ← faux
fin ← (tete = null) => la liste n'est pas vide. Fin est à faux
p ← tete
Tant que non fin **faire** => on entre
Si p^.val = x **alors** => la valeur est égale à x
Si p^.precedent = null **alors**
Si p^.suivant = null **alors** => le suivant n'est pas null
Sinon => on entre donc ici
p^.suivant^.precedent ← null
tete ← p^ .suivant
Finsi
Finsi
Désallouer(p)
trouvé ← vrai
fin ← vrai => on arrête la boucle
Fin Tant que
Fin Procédure

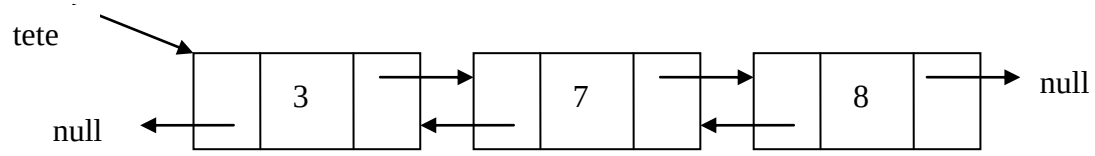
Avant la suppression : repères



Après la suppression



4. Supprimer_cellule (L,4)
On appelle la procédure avec l'adresse de début de liste L transmis à tete



Début
trouvé ← faux
fin ← (tete = null) => la liste n'est pas vide. Fin est à faux
p ← tete
Tant que non fin **faire** => on entre
    Si p^.val = x **alors** => la valeur (3) est différente donc direction sinon
    Sinon
        p← p^.suivant => on avance sur la cellule suivante (cellule de valeur à 7)
        fin ← (p = null) => fin reste à faux
        Si non fin **alors**
            fin ← (p^.val>x) => 7 > 4 fin reste à passe à vrai
        Finsi
    Finsi
Fin Tant que => fin est passée à vrai on sort de la boucle
Si non trouvé **alors** => trouvé est restée à faux
    Afficher (« élément non présent dans la liste »)
Finsi

On a ainsi affiché le message et la procédure s'arrête sans suppression