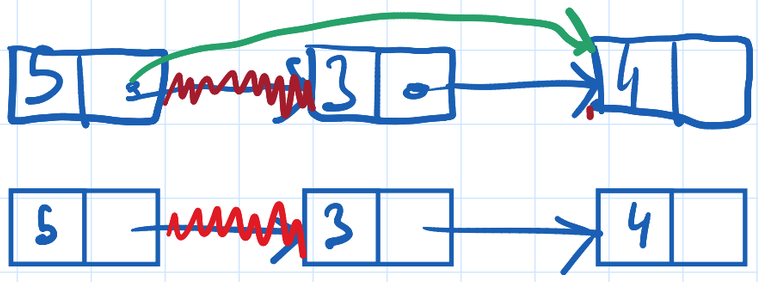
TD1: ASD

Exercice 1 : Suppression dans des listes chaînées

Nous représentons une liste simplement chaînée avec une classe `List` donnant accès à un attribut `head` de type `Cell`. Cette classe `Cell` possède deux attributs, un attribut `next` de type `Cell` et un attribue `value` stockant la valeur associée à cette cellule de la liste chaînée.

Nous souhaitons pouvoir supprimer une cellule de la liste chaînée.

Q 1.1 Commencer par dessiner une liste chaînée contenant, dans l'ordre, les valeurs 5, 3, 4. Dessiner les chaînages à modifier lors de la suppression de la cellule contenant la valeur 3. De même, en partant de la liste initiale, dessiner les chaînages à modifier lors de la suppression de la cellule contenant la valeur 5.



Nous souhaitons implémenter une méthode `delete` procédant à la suppression d'une cellule de la liste chaînée.

Q 1.2 Dans quelle classe devrait être cette méthode ?

Elle devrait être implémentée dans List, comme elle a accès à toutes les cellules.

Q 1.3 Pourquoi seulement passer la cellule à supprimer en paramètre de cette méthode n'est pas suffisant pour assurer une suppression efficace ?

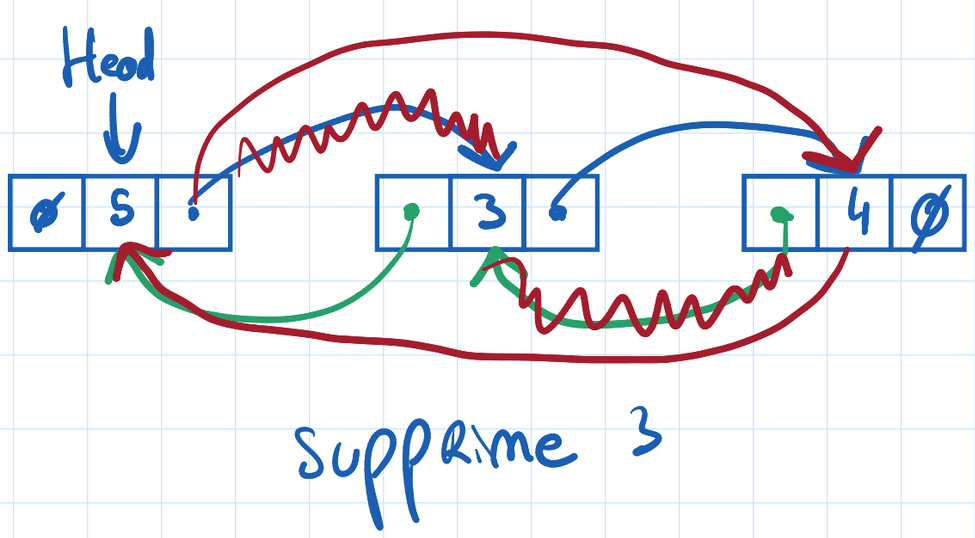
Ce n'est pas suffisant, car il est nécessaire de modifier l'état de la précédente.

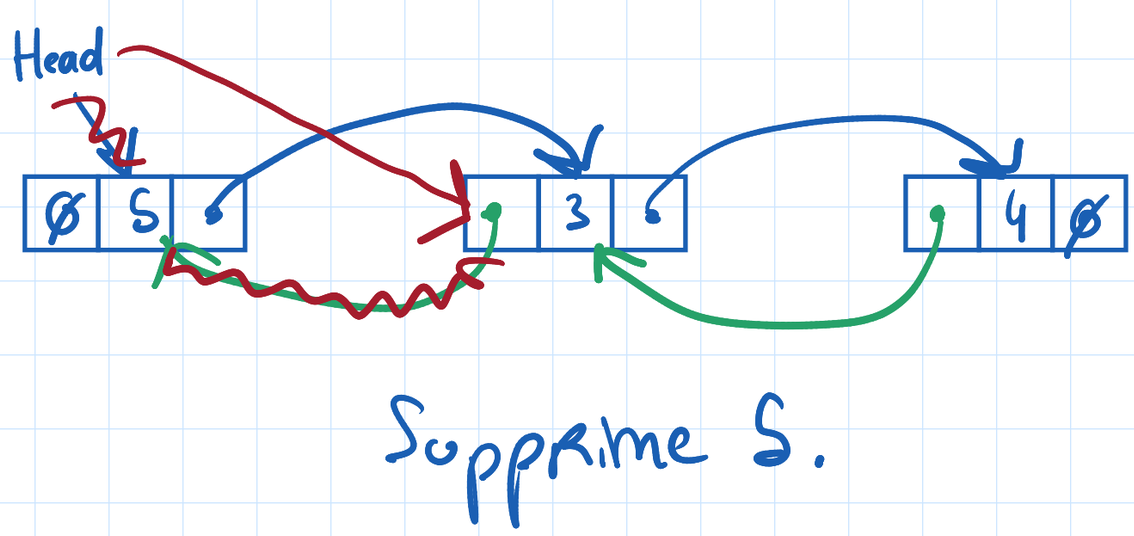
Q 1.4 Écrire le (pseudo-)code de la méthode `delete`, qui prend donc en paramètre la cellule courante, à supprimer, et la cellule qui la précède, si elle existe.

```
fct delete : Cell cell, Cell prec , List l
    Si prec est NULL :
        l.head ← cell.next;
    Sinon
        prec.next ← cell.next;
```

Intéressons-nous maintenant au cas où la liste est doublement chaînée. Nous avons une classe `DoubleList` ayant deux attributs `head` et `tail` faisant respectivement référence à la `DoubleCell` au début et à la fin de la liste. La classe `DoubleCell` possède trois attributs : `previous` et `next`, respectivement la `DoubleCell` qui précède et qui suit la cellule courante, et `value`, correspondant à la valeur associée à cette cellule de la liste chaînée.

Q 1.5 Commencer par dessiner une liste doublement chaînée contenant, dans l'ordre, les valeurs 5, 3, 4. Dessiner les chaînages à modifier lors de la suppression de la cellule contenant la valeur 3. De même, en partant de la liste initiale, dessiner les chaînages à modifier lors de la suppression de la cellule contenant la valeur 5. De même, en partant de la liste initiale, dessiner les chaînages à modifier lors de la suppression de la cellule contenant la valeur 4.





Q 1.6 Donner le code de la méthode `delete` pour des listes doublement chaînées

```
fct delete : Cell cell, Dlist l :
si cell.prec is NULL :
    l.head ← cell.next.
    cell.next.prec ← NULL          // cas de cell.next null manquant.
sinon :
    cell.prec.next ← cell.next
    cell.next.prec ← cell.prec.
```

Exercice 2 : Liste circulaire

On souhaite implanter une liste circulaire, c'est-à-dire que le suivant du dernier élément est le premier élément de la liste. On accédera à cette structure de donnée grâce à une primitive `valeur` qui prend en entrée la liste, retourne la valeur de l'élément courant et a pour effet de bord que le prochain appel à `valeur` donnera la valeur de l'élément suivant.

Par exemple, si la liste contient les éléments 6,9,0,3,1 alors si un appel à `valeur` retourne 3, l'appel suivant retournera 1, l'appel suivant retournera 6, etc.

Q 2.1 Proposer une définition d'une structure de données à base de tableau. On supposera que la fonction d'ajout est telle que les cases vides (sans élément) dans le tableau sont contiguës (modulo).

SD :

tableau

pointeur i

Q 2.2 Proposer avec cette structure un pseudocode de la fonction `valeur`.

```
fct valeur : Lcirc l :
    a ← l.get(i);
    l.i ← ((l.i+1) % l.taille();
    ret a;
```

Q 2.3 Proposer une définition d'une structure de données à base de liste simplement chaînée.

On prend une liste chaînée, mais on oblige que le dernier élément pointe sur head. On stocke tail aussi, et current :)

Q 2.4 Proposer avec cette structure un pseudocode de la fonction `ajouter` qui permet d'ajouter un élément dans la liste circulaire. Discuter de l'endroit où ajouter cet élément.

On l'ajoute après tail.

```
fct ajouter : Lcirc L, Cell elem :
    Si l.tail est NULL :
        l.head ← elem
        elem.next ← l.head ;
        l.tail ← elem ;
    Sinon :
        l.tail.next ← l.elem ;
        l.elem.next ← l.head ;
        l.tail ← elem.
```

Exercice 4 : Files et piles

En utilisant uniquement les opérations primitives sur chacune des deux structures de données.

Q 4.1 Montrer que l'on peut implanter une pile avec deux files.

Empiler → enfiler sur la file 1.

Dépiler : - défiler de la file 1 un élément et l'enfiler sur file 2 jusqu'à ce que la file soit de taille 1
- defiler le dernier élément et le retourner
- défiler la file 2 dans la file 1.

Q 4.2 Quel est le nombre d'itérations effectuées pour les opérations empiler et dépiler dans ce cas, en supposant que les opérations primitives sur les files ne font aucune itération ?

Empiler : 1 itération

Dépiler : 2(n-1) itérations.