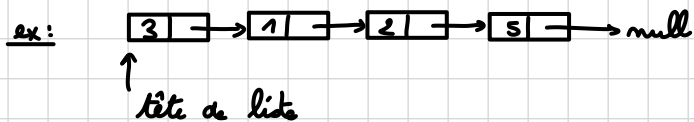


→ structure de données pour représenter une collection ordonnée d'objets du même type.

↳ un ensemble de cellules avec un contenu et un pointeur vers la cellule suivante.



Avantages : taille flexible → ajout et suppression en tête de liste en tps constant.

Désavantages : l'accès au i-ème élément nécessite de parcourir les éléments qui le précèdent.

→ recherche d'un élément  $x$  dans liste  $L$

```
search(l: list, x: key) {
    c = l.head
    while (c != null) { //until the end of the list
        if c.key == x return c
        c = c.next
    }
    return c; //null
}
```

→ recherche récursive

```
search-rec(l: list, x: key) {
    return (search-aux(l.head, x))
}

search-aux(c: cell, x: key) {
    if (c == null) return null
    if (c.key == x) return c
    else return search-aux(c.next, x)
}
```

→ insertion en tête de liste

```
insertion-head (l: list, x: key){  
    c = new cell()  
    c.key = x  
    c.next = l.head  
    l.head = c  
}
```

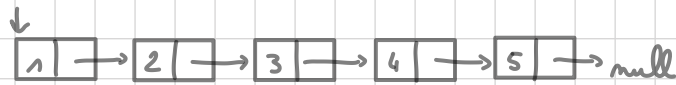
→ suppression en tête de liste

```
remove-head (l: list){  
    c = l.head  
    if (c != null) { l.head = c.next }  
    return c  
}
```

## TD

### Exercice 3

1) 

2) 

cell c2 = l.head.next  
c2.next = c2.next.next

3) cell c3 = new cell(3, null)  
cell c2 = l.head.next  
c3.next = c2.next  
c2.next = c3

### Exercice 4

1) rechercheMaximale(l: list){  
    c = l.head  
    if (c == null) return null  
    max = c  
    c = c.next  
    while (c != null)  
        if (c.key > max.key) max = c  
        c = c.next  
    }  
    return max

2) listeTriée(l: list){  
    c = l.head  
  
    while (c != null)  
        if (c.next == null) return 1 // fin de la liste  
        else if (c.key > c.next.key) return 0 // liste non triée  
        c = c.next  
    }  
    return 1 // liste vide et triée

### Exercice 2

1) comptageAux(t: tableau, x: element, m: entier)  
    si (m = 0) return 0  
    sinon si (t[m-1] == x) return 1 + comptageAux(t, x, m-1)  
    sinon return comptageAux(t, x, m-1)  
}

2) `comptage(t: tableau, x: element)`

`return comptageAux(t, x, t.length)`

}

3) initialisation  $n=0$  renvoie 0 car  $t[-1]$  n'existe pas.

hérédité On suppose vrai pour  $n$ . Montrons le pour  $n+1$ .

Le tableau  $t[0, n-1]$  contient `comptageAux(t, x, n)` occurrences de  $x$ .

Si  $t[n] == x$ , on ajoute 1.  
Sinon on renvoie le résultat.

Donc on a bien le nbr d'occurrences de  $x$  dans  $t[0, n]$ .

Donc `comptageAux` est correct.

`comptage(t, x, t.length)` renvoie le nbr d'occurrences de  $x$  dans  $t[0, t.length-1]$ ,  
c'est-à-dire  $t$  en entier. Donc `comptage` est correct.

### Exercice 5

`boolean palindromAux(String[] t, int n){`

`if (n == -1) return true;`

`if (t[n] != t[t.length-1-n]) return false;`

`return palindromAux(t, n-1);`

}

`boolean palindrome(String[] t){`

`int start = 0;`

`if (t.length % 2 == 0) start = t.length/2;`

`else start = t.length/2 - 1; // the letter in the middle can't be compared to another`

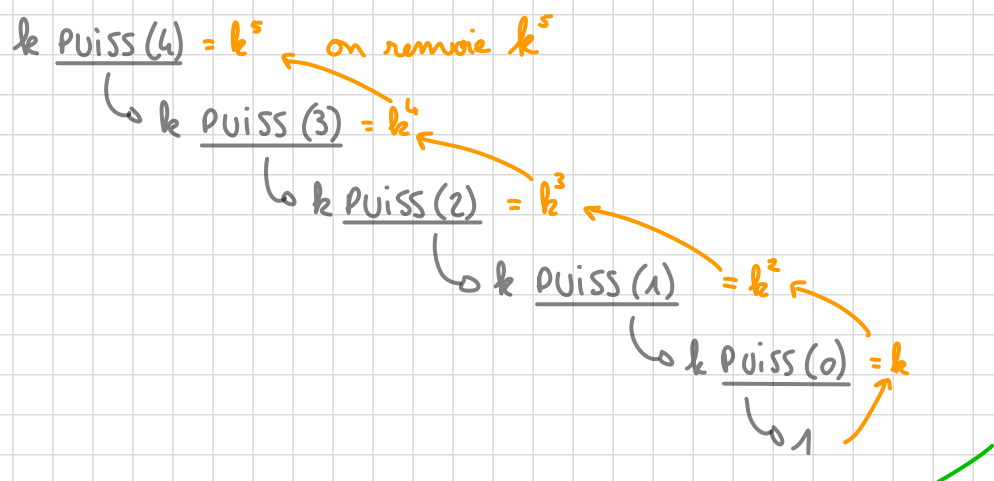
`return palindromAux(t, start);`

}

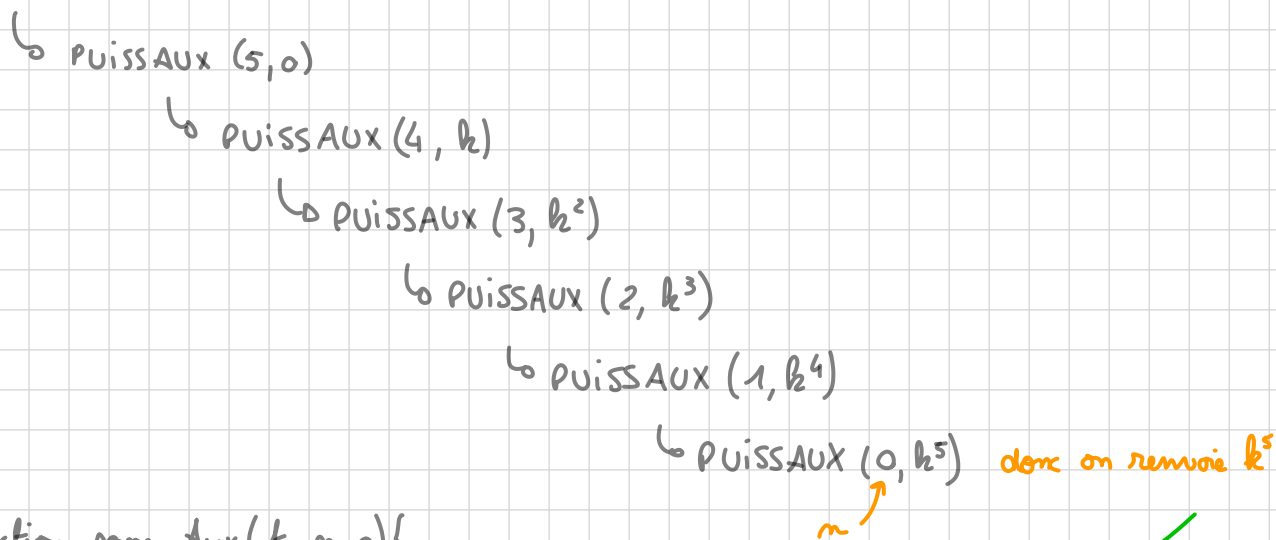
## Exercice 1

1) fonction `PUISSTER(k, n)` {  
    retourne `PUISSAUX(k, n, 1)` ✓  
}

2) `PUISS(5)`:



`PUISSTER(5)`:



3) fonction `sommeAux(k, m, a)` {  
    si ( $m = k$ ) return `a`  
    sinon return (`k, m+1, a+T[m]`) ✓  
}  
  
fonction `sommeTer(m, T)` {  
    return (`sommeAux(T.length, 0, 0)`) ✓  
}