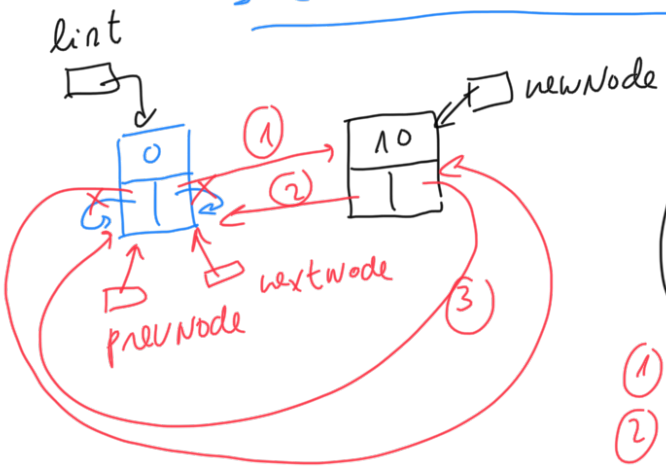
DList – addLast

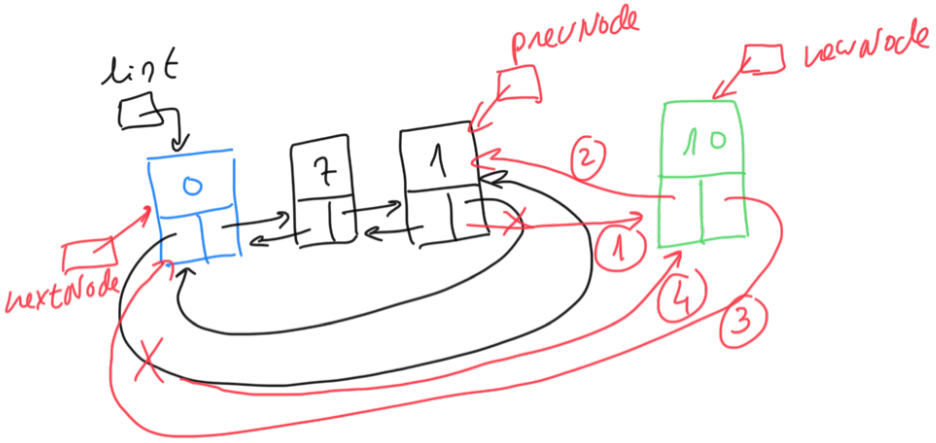
1: caso - lista vazia



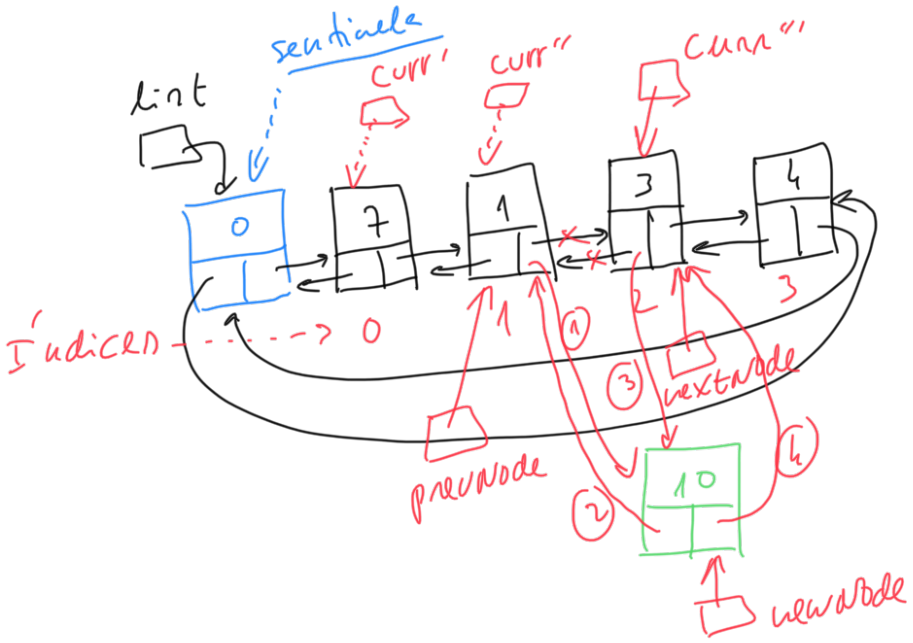

```
addLast:
prevNode = list.previous
nextNode = list
var prevNode: IntNode? = null
var nextNode: IntNode? = null
val newNode = IntNode(v)
```

1. prevNode.next = newNode
2. newNode.previous = prevNode
3. newNode.next = nextNode
4. nextNode.previous = newNode

2: caso: lista nao vazia



DList - insert (v, index)



Inserir elem (10) no índice (2)

⇒ Inserir novo nó atrás do curr

```
fun insert (v: Int, index: Int) {
    var prevNode : IntNode? = null
    var nextNode :    "        "
    var curr :        "
    curr = list.next // aponta o 1: elem.
    var i = 0
    while ( i < index ) {
            curr = curr.next
            ++i
    }
    prevNode = curr.previous
    nextNode = curr
    // ligar
    val newNode = IntNode (v)
(1) prevNode.next = newNode
(2) newNode.previous = prevNode
(3) nextNode.previous = newNode
(4) newNode.next = nextNode
    ++size
}
```

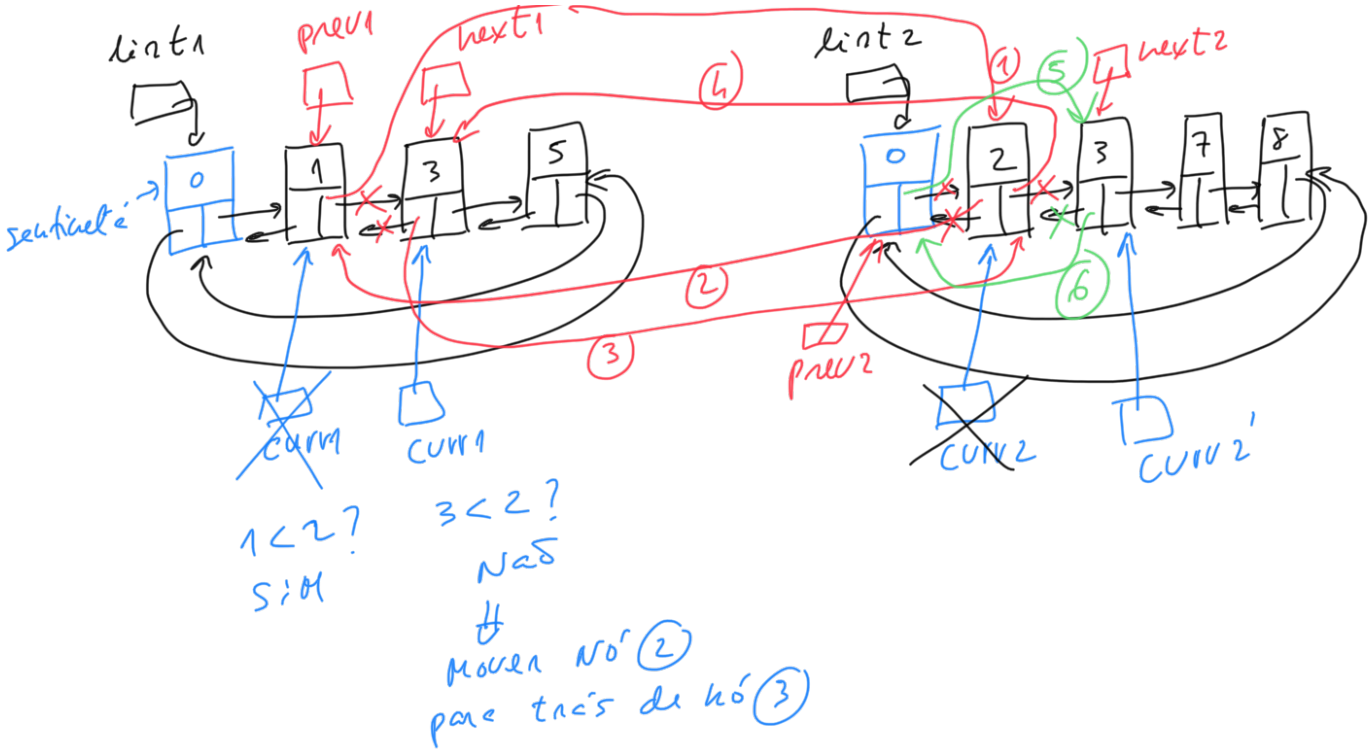
T.P.C.

Verificar que funciona caso a lista esteja vazia e índice == 0

DList - merge

```
fun <E> merge( list1 : Node<E>, list2: Node<E>,
               cmp: Comparator<E>) : Node<E> {
```

// listas duplamente ligadas, circulares, com
// sentinela, e ordenadas crescentemente pelo
// comparador cmp.
// o método deve fazer merge das duas listas,
// de forma ordenada, movendo os nós de list2
// para list1. No final, retorna a list1, e a
// list2 deve ficar vazia




```
Var   prev1 = Node<E>? = null
"     next1 = "        "
"     prev2 = "        "
"     next2 = "        "
"     curr1 = "        "
"     curr2 = "        "

curr1 = list1.next
curr2 = list2.next
while (curr1 != list1 && curr2 != list2) {
    if (cmp.compare(curr1.value, curr2.value)
                                <= 0) { // value1 <=
                                        value2
        curr1 = curr1.next
    }
    else { // value1 > value2
           // ligar nó com value2 na list1

        prev1 = curr1.previous
        next1 = curr1
        prev2 = curr2.previous
        next2 = curr2.next
```

5.

```
        (1) prev1.next = curr2
        (2) curr2.previous = prev1
        (3) next1.previous = curr2
        (4) curr2.next = next1
        (5) prev2.next = next2
        (6) next2.previous = prev2
        curr2 = next2
    }
}
if (curr2 != list2) { // list2 ainda não acabou
    prev1 = curr1.previous //último de list1
    next1 = curr1 //sentinela de list1
    // ligar list2 no fim de list1
    var last2 = list2.previous
    curr2.previous = prev1
    prev1.next = curr2
    next1.previous = last2
    last2.next = next1
}
// colocar list2 → vazia
list2.previous = list2
list2.next = list2.previous
return list1
}
```