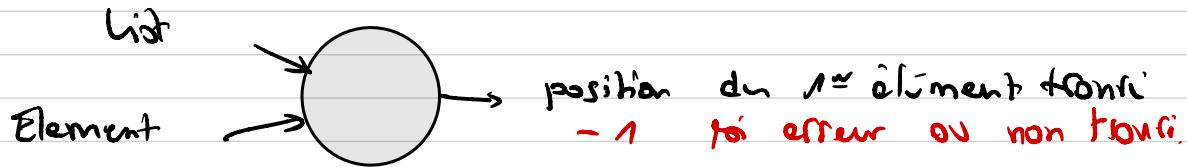
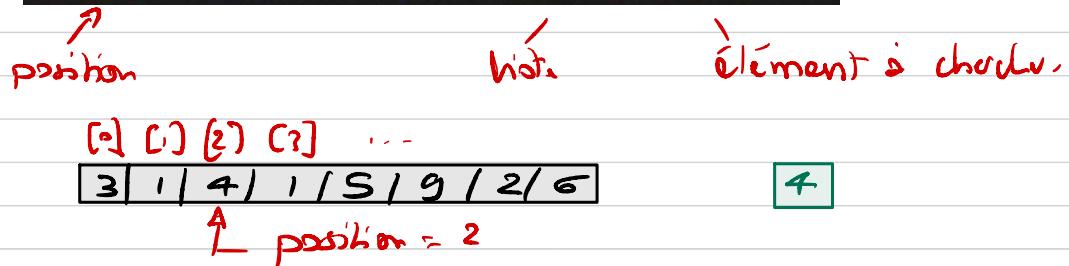


Recherche d'un élément par sa valeur.



```
int32_t searchElemByValue(pList l, Element e)
```



Parcourir la liste → [] ...
2 conditions à arrêt:

1) on a trouvé
2) atteint fin de liste

indicateur boolien : finished. TRUE → fin
FALSE → on continue

indicateur position : -1 : pas trouvé
≠ -1 : position de l'élément

k = 0; // index courant de recherche.

finished = isListEmpty(lis) // évite de chercher
si liste vide.

While (! finished) {

if (lis[k] == e) {

finished = true;
pos = k;

} on a trouvé.

k++;

if (k >= lis.numElem) { // fin de liste atteinte.

finished = false.

}

}

Réultat de la recherche par valeur

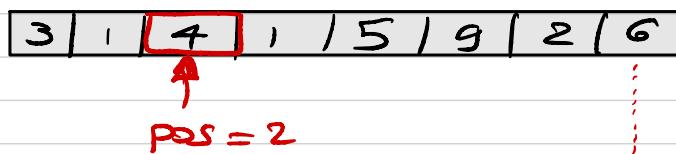
```
initList(&l);
insertElemAt(&l, END_OF_LIST, 3.0);
insertElemAt(&l, END_OF_LIST, 1.0);
insertElemAt(&l, END_OF_LIST, 4.0);
insertElemAt(&l, END_OF_LIST, 1.0);
insertElemAt(&l, END_OF_LIST, 5.0);
insertElemAt(&l, END_OF_LIST, 9.0);
insertElemAt(&l, END_OF_LIST, 2.0);
insertElemAt(&l, END_OF_LIST, 6.0);
displayList(&l);
pos = searchElemByValue(&l, 4);
printf("pos=%d\n", pos);
```

```
→ TD20210531 git:(main) ✘ ./app
 3.000 1.000 4.000 1.000 5.000 9.000 2.000 6.000
pos=2
```

main

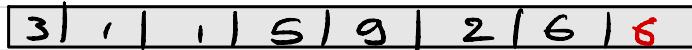
Suppression d'un élément dans une liste

AVANT



$n = 8$

APRÈS



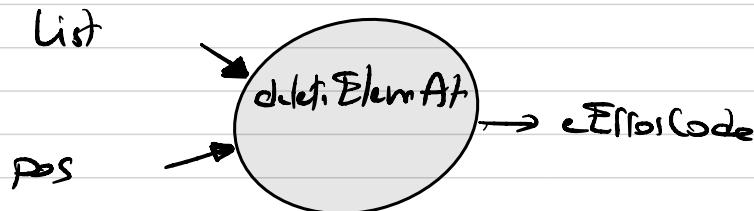
$n = 7$

Tous les éléments situés après $\rightarrow pos$ sont décalés d'une position à gauche \leftarrow

Boucle k de $[pos+1] \rightarrow [n]$
 $e[k] = e[k+1]$

M.A.J du nbre d'éléments de la liste

(1) Liste peut non être vide



15^b36

```

eErrorCode deleteElemAt(pList l, int32_t pos) {
    eErrorCode returnCode = E_NO_ERROR;
    int32_t k = 0;

    if (l != NULL) {
        if (!isEmpty(*l)) {
            for (k = pos+1; k < l->numElem; k++) {
                l->e[k-1] = l->e[k];
            }
            l->numElem--;
        }
        M.A.J nb elem.
    }
    else {
        returnCode = E_BAD_LIST;
    }
    return returnCode;
}

```

Annotations in yellow:

- test si liste valide
- test si liste non valide
- M.A.J nb elem.

(1) (2) (3)



15^b36

$l \rightarrow numElem = 4$
 $pos = 3$

$k = 4$
 $k < l \rightarrow numElem : \text{faux}$

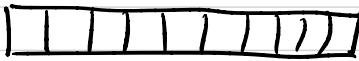
$l \rightarrow numElem < 3$

Liste Tableau Dynamique

Element $e[\text{LIST_CAPACITY}] \rightarrow \text{Element } *e;$

initList : allocation dynamique de "LIST_CAPACITY" éléments.

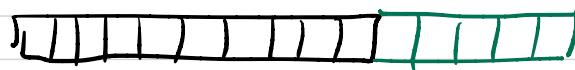
malloc



10 éléments alloués lors de l'init

insertion si liste pleine, réallocation de "LIST-RESIZE"

1^{re} extension



realloc

\leftarrow LIST-CAPACITY \rightarrow LIST-RESIZE

2nd



LIST-RESIZE

isListFull

numElem == LIST-CAPACITY

```
typedef struct {
    uint32_t numElem;
    Element e[LIST_CAPACITY];
} sList, *pList;
```

ajouter un champs pour la capacité de la liste
uint32_t maxNumElem;

31

① Dupliquer le TD2020S3a en TD2020S31b

② Modif sur sList

③ MAJ initList

④ Ajouter une fonction freeList

16'05

```

eErrorCode insertElemAt(pList l, int32_t pos, Element e) {
    eErrorCode returnCode = E_NO_ERROR;
    int32_t k = 0;
    int32_t n = 0;

    if (l != NULL)
    {
        if(!isListFull(l)) { } ① Test si liste  
pleine
            n = l->numElem;
            if(pos == -1)
                pos = n;
            for (k = n - 1; k >= pos; k--)
si oui → RESIZE  
si resize on
                { ↳ insertion
                    l->e[k + 1] = l->e[k];
                }
                l->e[pos] = e;
                l->numElem++;
            }
            else { ↳ LIST-FULL
                returnCode = E_LIST_FULL;
            }
        }
        else {
            returnCode = E_BAD_LIST;
        }
        return returnCode;
    }
}

```

if (isListFull(l)) {

[Augmentation de la taille de la liste
(+ LIST-RESIZE)

Si on Alloue insertion
sinon ERREUR E-LIST-FULL.

}

else {

insertion

}

|| 16⁴ 26

```

eErrorCode insertElemAt(pList l, int32_t pos, Element e) {
    eErrorCode returnCode = E_NO_ERROR;
    int32_t k = 0;
    int32_t n = 0;
    Element *newE = NULL;

    if (l != NULL)
    {
        if (isListFull(l)) {
            newE = (Element *)realloc(l->e, (l->maxNumElem + LIST_RESIZE) * sizeof(Element));
            if(newE) {
                l->e = newE;
                l->maxNumElem += LIST_RESIZE;
            }
            else {
                returnCode = E_LIST_FULL;
            }
        }
        else {
            if(!isListFull(l)) {
                n = l->numElem;
                if(pos == -1)
                    pos = n;
                for (k = n - 1; k >= pos; k--)
                {
                    l->e[k + 1] = l->e[k];
                }
                l->e[pos] = e;
                l->numElem++;
            }
            else {
                returnCode = E_LIST_FULL;
            }
        }
        else {
            returnCode = E_BAD_LIST;
        }
    }
    return returnCode;
}

```

test si list pleine

variable pour stocker la nouvelle @

test si realloc OK.

réattribuer tableau pour le null @

MAJ de la nouvelle capacité.

↳ si realloc n'a pas fonctionné => liste toujours pleine

inchange