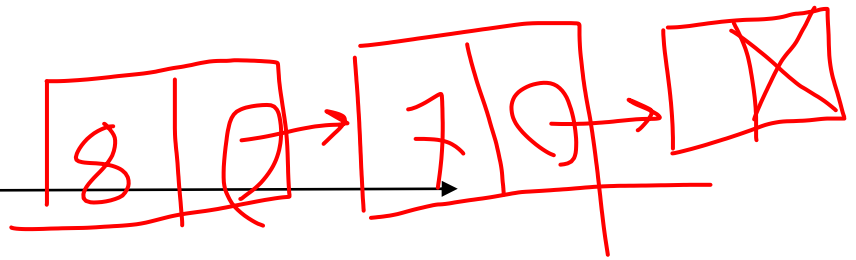
Course descriptions : semester 2, Data Science and Big Data..

Programmation & Algorithmique II

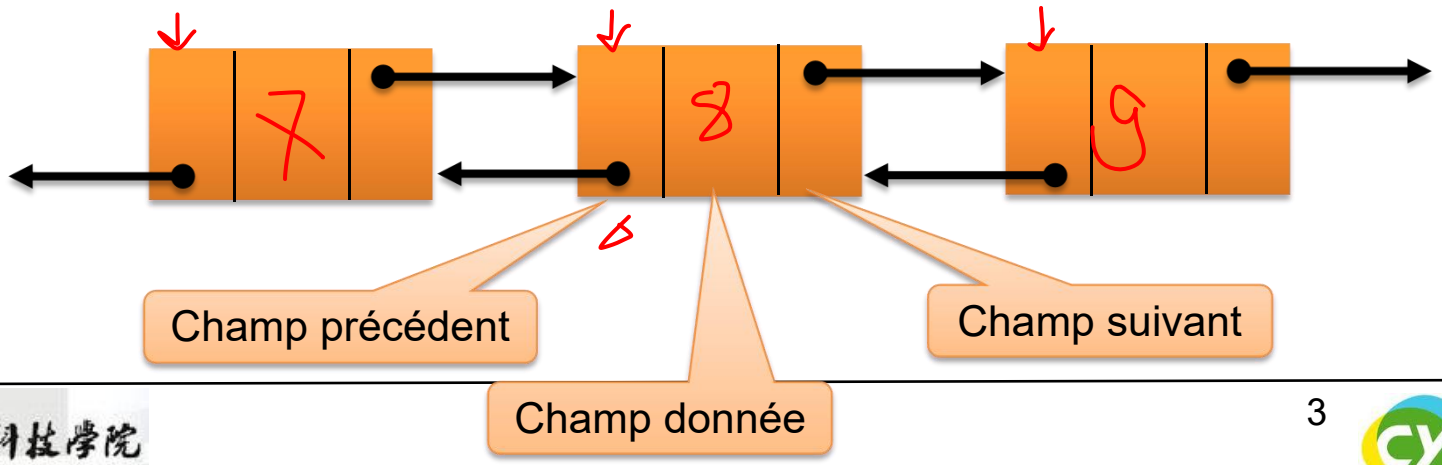
CM 6 : listes chaînées (3)

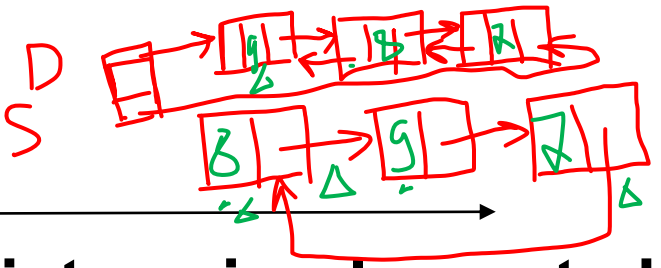
PLAN

- Les listes doublement chainées
  - La représentation graphique
  - Les services des listes doublement chaînées
    - Création
    - Ajout
    - Suppression
    - Affichage



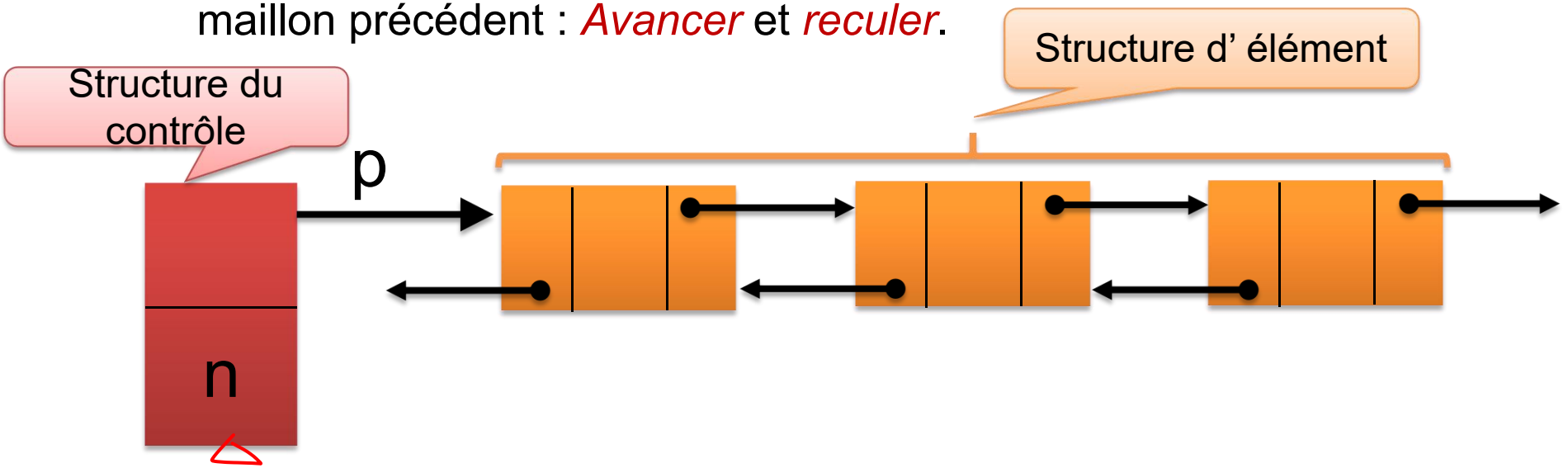
- **Définition**
  - Une liste chaînée, dans laquelle on peut accéder à l'élément prédécesseur.
    - Un élément est une structure qui comporte trois champs :
      - Le champ donnée : il contient des informations sur l'élément représenté par l' élément ;
      - Le champ suivant : il représente un pointeur qui contient l'adresse de l' élément suivante.
      - Le champ précédent : il représente un pointeur qui contient l'adresse de l' élément précédente.





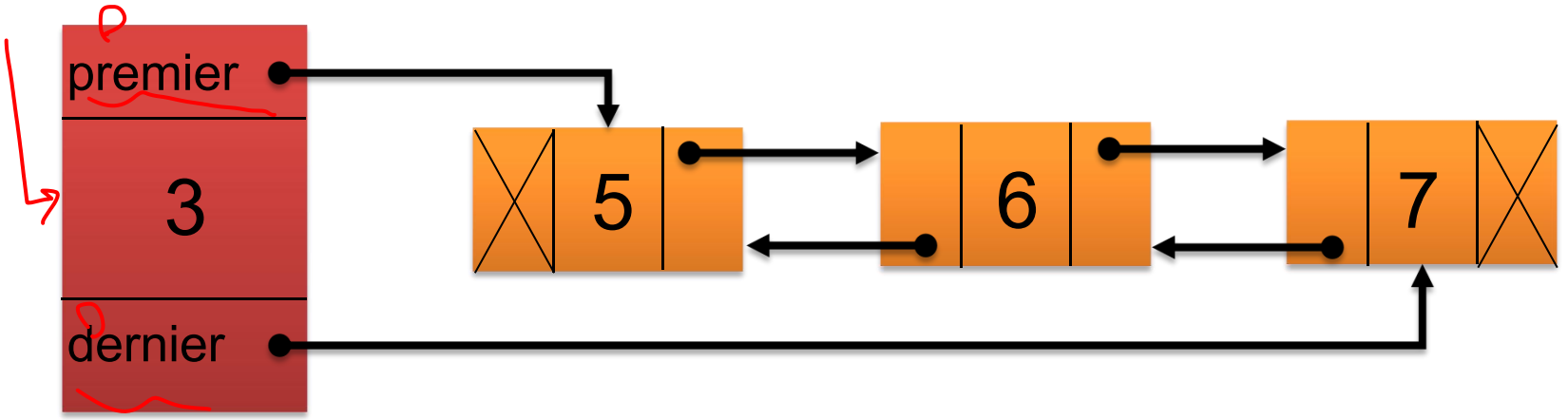
- **Points communs avec les listes simplement chainée**
  - Structures permettant de stocker une collection de données de même type.
  - L'espace mémoire utilisé n'est pas contigüe.
  - La taille est inconnue à priori.
  - Une liste doublement chainée est constituée de cellules qui sont liées entre elles par des pointeurs.
  - Pour accéder à un élément quelconque d'une liste, il faut parcourir la liste jusqu' à cet élément.
- **Différences**
  - **On peut accéder directement au premier et dernier élément**
  - **On peut parcourir la liste dans les 2 sens**
    - **on peut donc revenir en arrière.**

- **Manipulation:**
  - Une liste doublement chaînée présente l'avantage de donner accès au maillon précédent : *Avancer* et *reculer*.



Le p d'une liste doublement chaînée n'est pas forcément positionné sur la première cellule. (mais on le fait quand même.)

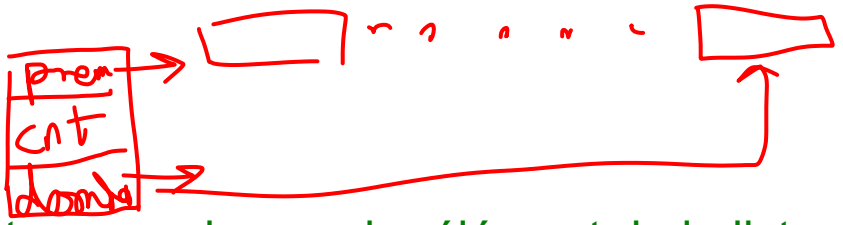
- **Implémentation avec deux pointeurs**
  - Pour déclarer une variable de type liste doublement chainée, il existe deux possibilités :
    - En utilisant une variable statique : Dliste L.
    - En utilisant une variable dynamique : Dliste * L.
  - La variable L est une structure contenant deux champs :
    - L.premier & L.dernier ←→ L->premier & L->dernier
- **Représentation graphique**



LISTES DOUBLEMENT CHAINÉES

- **Implémentation C d'une liste simplement chaînée en utilisant deux structures**

```
#include <stdio.h>
#include <stdlib.h>
//1) structure Element pour représenter un élément de la liste chaînée
typedef struct Element Element;
struct Element
{
        int nombre; // donnée
        Element *suivant; // pointeur vers l'élément suivant de la liste
        Element *precedent; // pointeur vers l'élément précédent de la liste
};
//2) La structure de contrôle
typedef struct Liste Liste;
struct Liste
{
        Element *premier; // pointeur vers le premier élément de la liste
        Element *dernier; // pointeur vers le dernier élément de la liste
        int cnt ;   // compteur des éléments de la liste
};
```

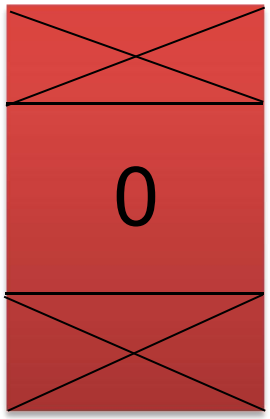


Création de la liste

On fait pointer la tête vers NULL.

On fait pointer la queue vers NULL.

On initialise le nombre d' éléments à 0.



LISTES DOUBLEMENT CHAINÉES

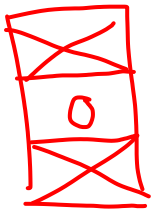
- **Implémentation C d'une liste simplement chaînée en utilisant deux structures**

```
Liste *initialisation()
{
   // Liste *liste = (Liste *)malloc(sizeof(Liste));
   Liste *liste = malloc(sizeof(*liste));

   if (liste == NULL)
   {
      exit(EXIT_FAILURE);
   }

   liste->premier = NULL;
   liste->dernier = NULL;
   liste->cnt = 0;

   return liste;
}
```



浙江科技學院 ZHEJIANG UNIVERSITY OF SCIENCE & TECHNOLOGY

CERGY PARIS UNIVERSITÉ

Ajout au début de la liste

Nous tentons de créer un nouveau élément (vérifiant que ca va réussi! Et faire le donnée=5)

S'il n'existe pas de dernier élément (donc la liste est vide) Alors

Nous faisons pointer le nouveau élément vers NULL (élément précédent) et NULL (élément suivant)

Nous faisons pointer la tête et la fin de liste vers notre nouvel élément

Sinon

Nous rattachons le premier élément de notre liste à notre nouvel élément

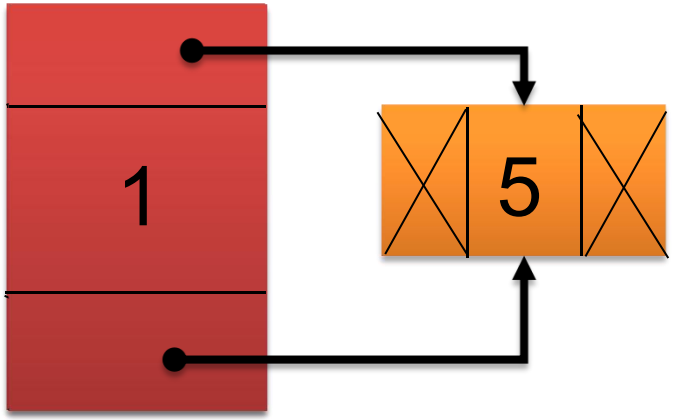
Nous faisons pointer le nouveau élément vers le premier élément de notre liste

Nous faisons pointer le nouveau élément vers NULL (élément précédent)

Nous faisons pointer la tête de liste vers notre nouvel élément

Fin si

Nous incrémentons le nombre des éléments



Ajout au début de la liste

Nous tentons de créer un nouveau élément (vérifiant que ca va réussi! Et faire le donnée=5)

S'il n'existe pas de dernier élément (donc la liste est vide) Alors

Nous faisons pointer le nouveau élément vers NULL (élément précédent) et NULL (élément suivant)

Nous faisons pointer la tête et la fin de liste vers notre nouvel élément

Sinon

Nous rattachons le premier élément de notre liste à notre nouvel élément

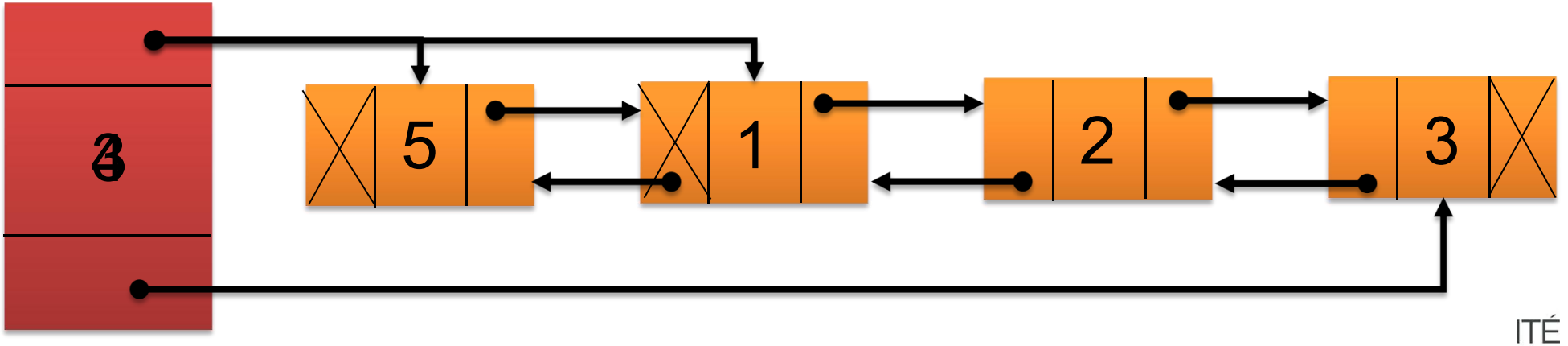
Nous faisons pointer le nouveau élément vers le premier élément de notre liste

Nous faisons pointer le nouveau élément vers NULL (élément précédent)

Nous faisons pointer la tête de liste vers notre nouvel élément

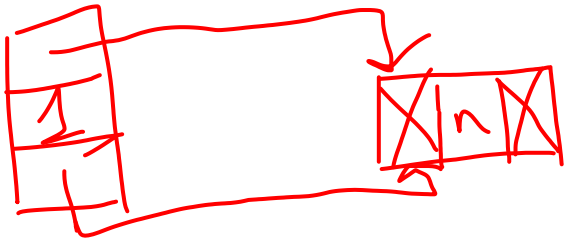
Fin si

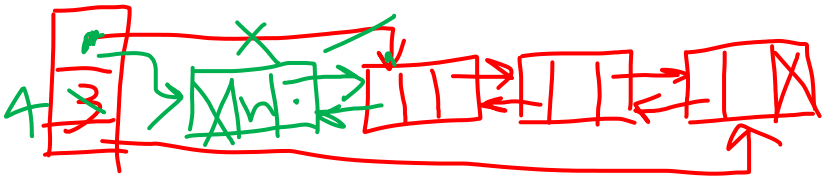
Nous incrémentons le nombre des éléments



- **Implémentation C d'une liste simplement chaînée en utilisant deux structures**

```
void inserTete(Liste *liste, int nvNombre)
{
  Element *p = malloc(sizeof(*p));
  p->nombre = nvNombre;
  if(liste->premier==NULL && liste->dernier == NULL)
  {
    p->suivant = NULL;
    p->precedent = NULL;
    liste->premier = p;
    liste->dernier = p;
  }
  else
  {
    p->suivant = liste->premier;
    p->precedent = NULL;
    liste->premier->precedent = p;
    liste->premier=p;
  }
  liste->cnt += 1;
}
```





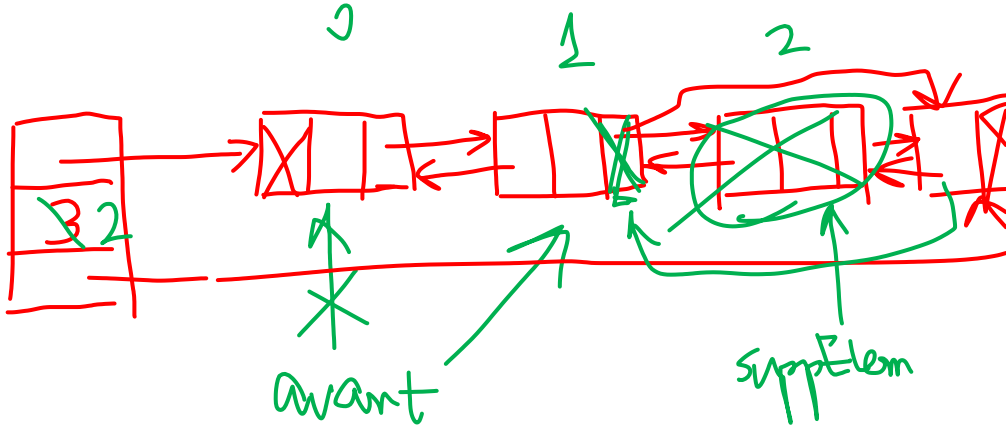
QUIZ

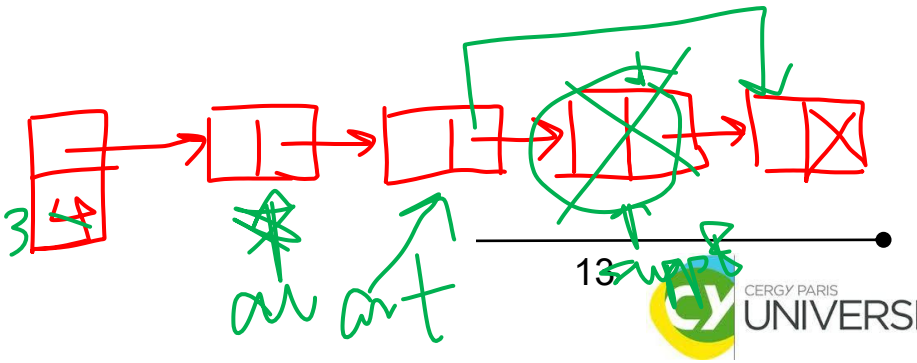
Question : Est-ce que la fonction en-dessous fonctionne correctement sans fuite de mémoire?
Choix:
A- Oui
B- Non

double ✗
simple ✓

5 minute

```
void suppElemPos(Liste *liste, int i){
    if (liste->premier == NULL && i < 1){
        printf("\n La liste est vide\n");
        exit(0);
    }
    if (liste->cnt == 1)
                        suppTete(liste);
    Element *avant = liste->premier;
    for(int x = 1; x < i; x++)
                avant = avant->suivant;
    Element *suppElem = avant->suivant;
    avant->suivant = suppElem->suivant;
    free(suppElem);
    liste->cnt--;
}
```





Ajout à la fin de la liste

Nous tentons de créer un nouveau élément (vérifiant que ca va réussi! Et faire le donnée=5)

S'il n'existe pas de dernier élément (donc la liste est vide) Alors

    Nous faisons pointer le nouveau élément vers NULL (élément précédent) et NULL (élément suivant)

    Nous faisons pointer la tête et la fin de liste vers notre nouvel élément

Sinon

    Nous rattachons le dernier élément de notre liste à notre nouvel élément

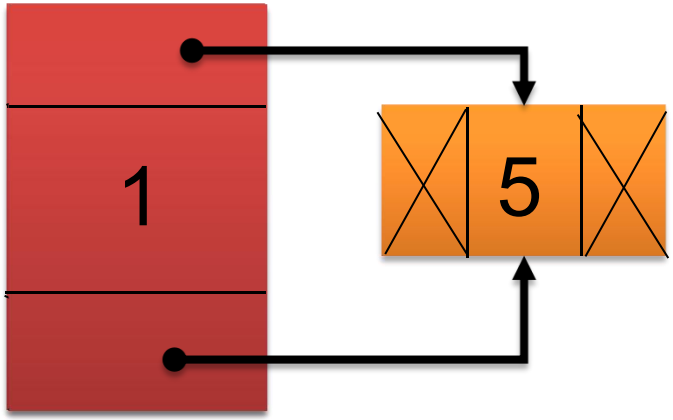
    Nous faisons pointer le nouveau élément vers le dernier élément de notre liste

    Nous faisons pointer le nouveau élément vers NULL (élément suivant)

    Nous faisons pointer le queue de liste vers notre nouvel élément

Fin si

Nous incrémentons le nombre des éléments



Ajout à la fin de la liste

Nous tentons de créer un nouveau élément (vérifiant que ca va réussi! Et faire le donnée=5)

S'il n'existe pas de dernier élément (donc la liste est vide) Alors

Nous faisons pointer le nouveau élément vers NULL (élément précédent) et NULL (élément suivant)

Nous faisons pointer la tête et la fin de liste vers notre nouvel élément

Sinon

Nous rattachons le dernier élément de notre liste à notre nouvel élément

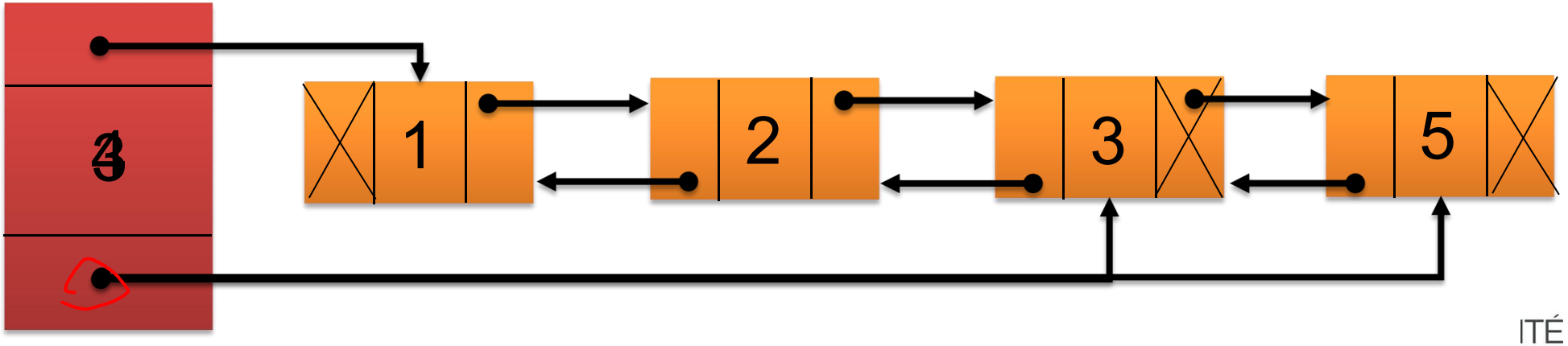
Nous faisons pointer le nouveau élément vers le dernier élément de notre liste

Nous faisons pointer le nouveau élément vers NULL (élément suivant)

Nous faisons pointer le queue de liste vers notre nouvel élément

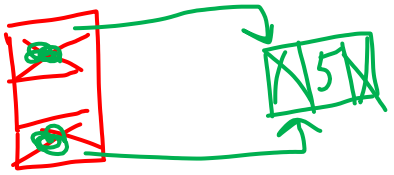
Fin si

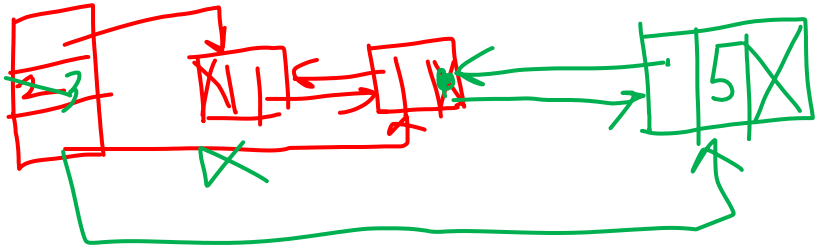
Nous incrémentons le nombre des éléments



- **Implémentation C d'une liste simplement chaînée en utilisant deux structures**

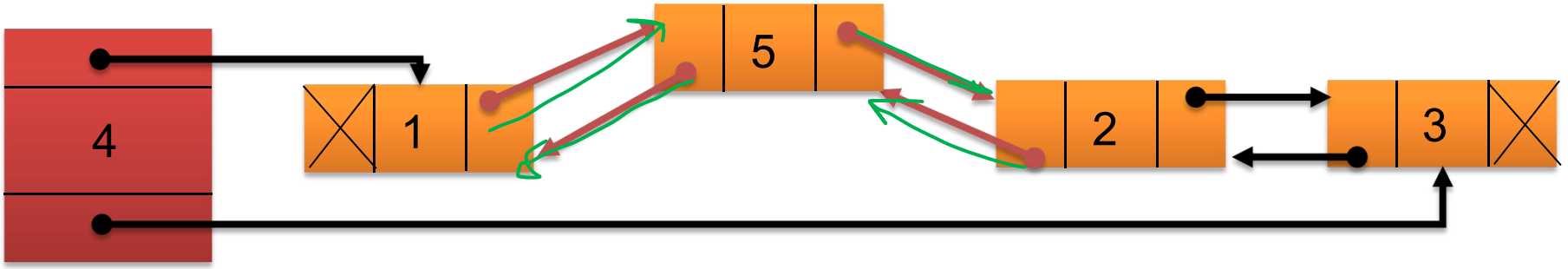
```
void inserQueue(Liste *liste, int nvNombre)
{
  Element *p = malloc(sizeof(*p));
  p->nombre = nvNombre;
  if(liste->premier==NULL && liste->dernier == NULL)
  {
    p->suivant = NULL;
    p->precedent = NULL;
    liste->premier = p;
    liste->dernier = p;
  }
  else
  {
    p->precedent = liste->dernier;
    p->suivant = NULL;
    liste->dernier->suivant = p;
    liste->dernier=p;
  }
  liste->cnt += 1;
}
```





Ajout à la ième position depuis la tête de la liste

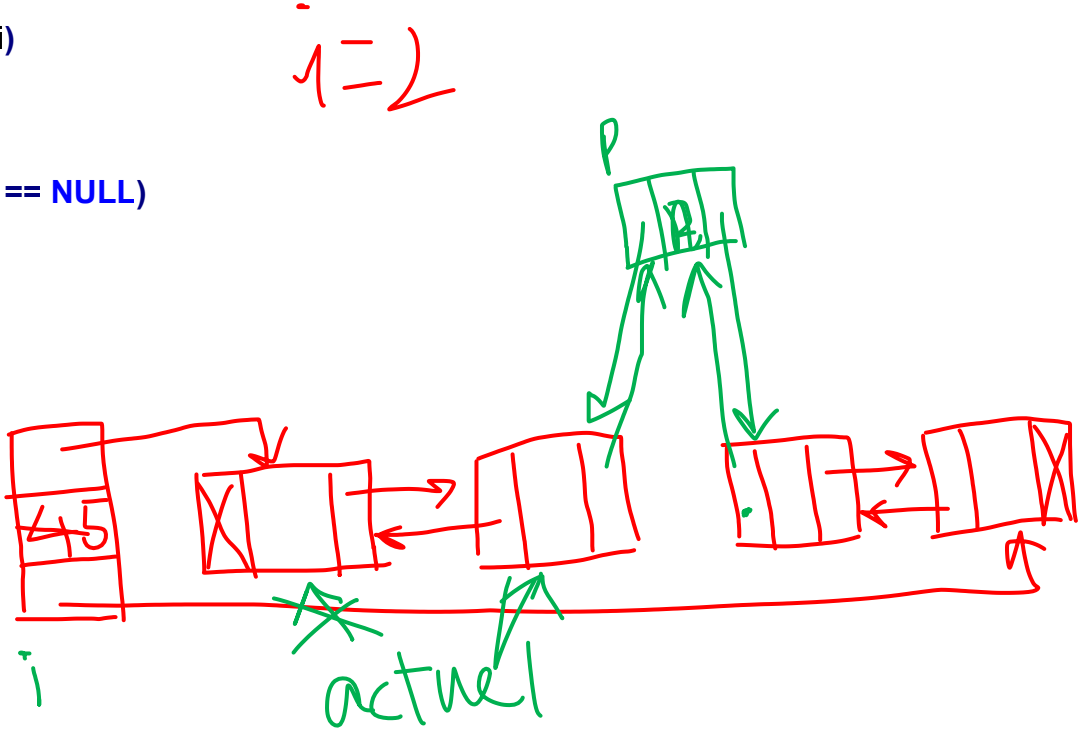
Nous tentons de créer un nouveau élément (vérifiant que ca va réussi! Et faire le donnée=5)
S'il n'existe pas de dernier élément (donc la liste est vide) Alors
    Nous faisons pointer le nouveau élément vers NULL (élément précédent) et NULL (élément suivant)
    Nous faisons pointer la tête et la fin de liste vers notre nouvel élément
Sinon SI i == 0 ALORS
    On ajoute au début de la liste
Sinon SI 0 < i < nombre d' éléments de la liste -1 ALORS
    Nous positionnons un pointeur sur le i-1ème élément.
    Nous faisons pointer le nouveau élément vers le ième élément suivant
    Nous faisons pointer vers le ième élément suivant vers le nouveau élément (précédent)
    Nous faisons pointer vers le i-1ème élément précédent vers le nouveau élément (suivant)
    Nous faisons pointer le nouveau élément vers le i-1ème élément précédent
Sinon
    On ajoute à la fin de la liste
Fin si
Nous incrémentons le nombre des éléments



LISTES DOUBLEMENT CHAINÉES

- **Implémentation C d'une liste simplement chaînée en utilisant deux structures**

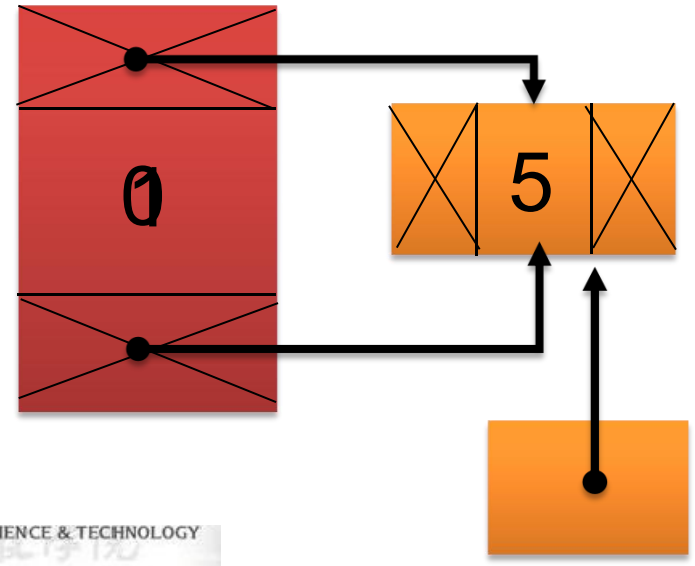
```
void inserPos(Liste *liste, int nvNombre, int i)
{
  Element *p = malloc(sizeof(*p));
  p->nombre = nvNombre;
  if(liste->premier==NULL && liste->dernier == NULL)
  {
    p->suivant = NULL;
    p->precedent = NULL;
    liste->premier = p;
    liste->dernier = p;
  }
  else if(i == 0)
    inserTete(liste, nvNombre);
  else if(i < liste->cnt)
  {
    Element *actuel = liste->premier;
    for (int x = 1; x<i; x++ ){
      actuel = actuel->suivant;
    }
    p->suivant = actuel->suivant;
    p->precedent = actuel;
    actuel->suivant->precedent = p;
    actuel->suivant = p;
  }
  else
  {
   inserQueue(liste, nvNombre);
   return;
  }
  liste->cnt += 1;
}
```



suppression

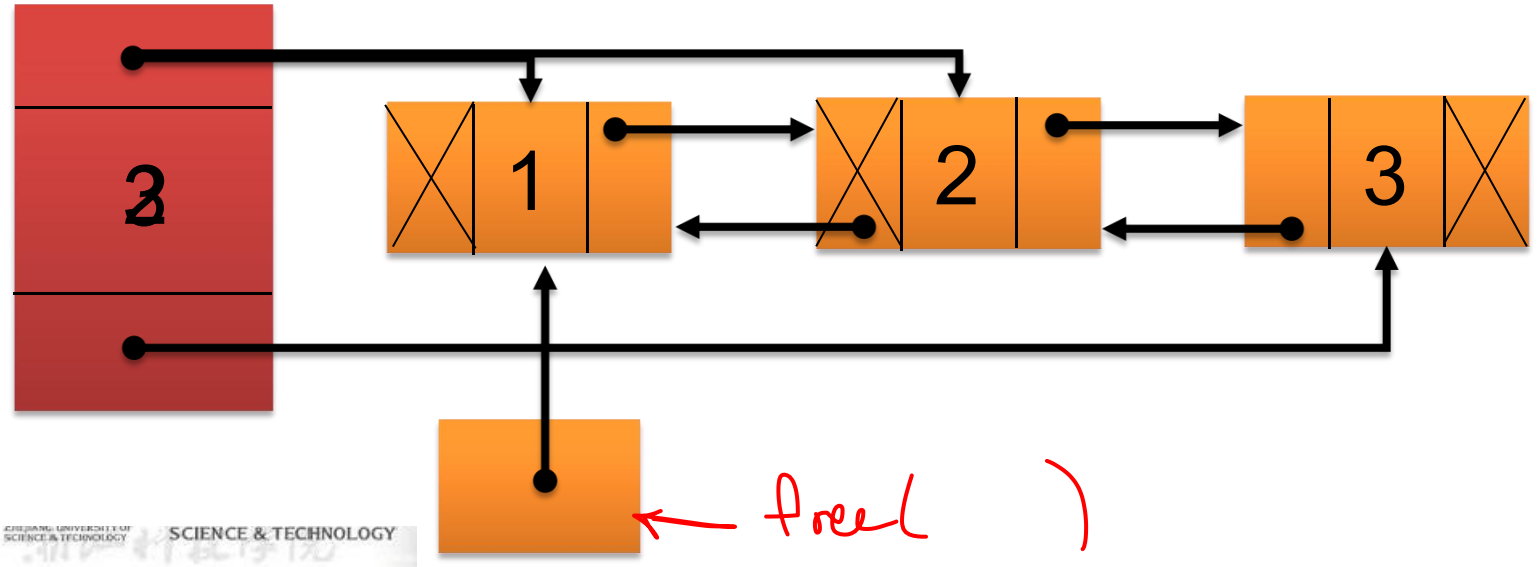
Retrait au début de la liste

SI la liste n'est pas vide ALORS
    On fait pointer un pointeur sur le premier élément de la liste.
    Si il y a qu' un seul élément ALORS
        On fait pointer la tête et la queue de liste sur NULL.
        On détruit l' élément pointé par le pointeur.
    Sinon
        On fait pointer la tête de liste sur le deuxième élément
        On fait pointer le nouveau élément en tête de la liste vers NULL (élément précédent)
        On détruit l' élément pointé par le pointeur.
    FinSI
    On décrémente le nombre d' éléments.
FINSI



Retrait au début de la liste

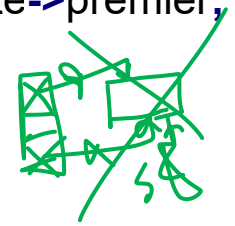
SI la liste n'est pas vide ALORS
    On fait pointer un pointeur sur le premier élément de la liste.
    Si il y a qu' un seul élément ALORS
        On fait pointer la tête et la queue de liste sur NULL.
        On détruit l' élément pointé par le pointeur.
    Sinon
        On fait pointer la tête de liste sur le deuxième élément
        On fait pointer le nouveau élément en tête de la liste vers NULL (élément précédent)
        On détruit l' élément pointé par le pointeur.
    FinSI
    On décrémente le nombre d' éléments.
FINSI

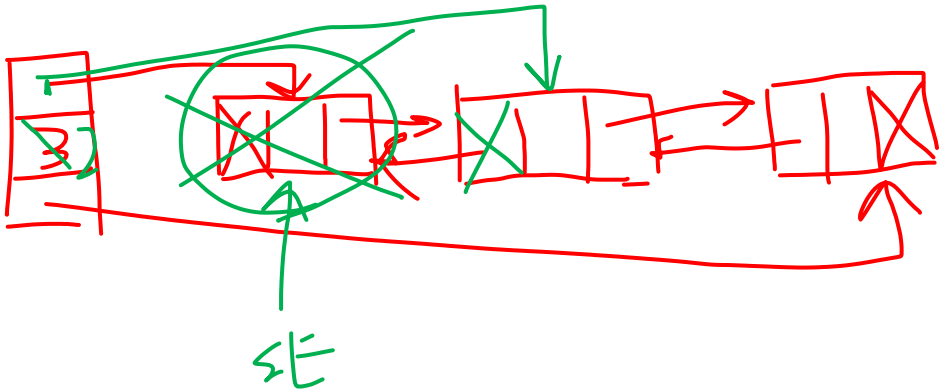


LISTES DOUBLEMENT CHAINÉES

- **Implémentation C d'une liste simplement chaînée en utilisant deux structures**

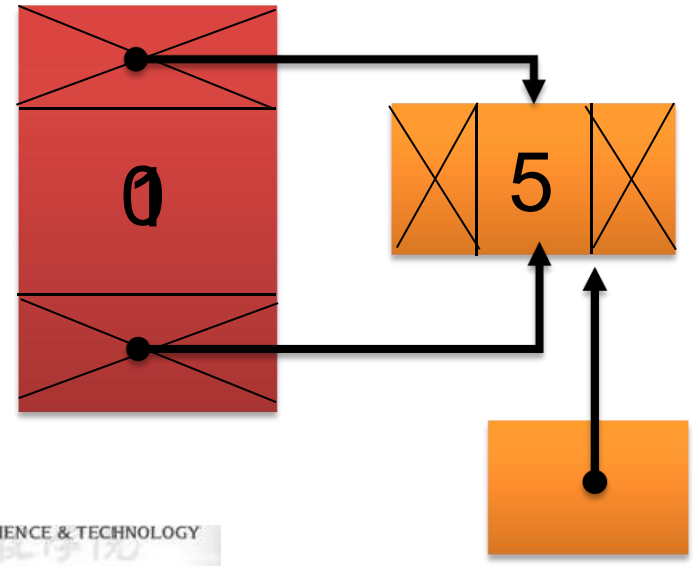
```
void suppTete(Liste *liste)
{
  if(liste)
  {
    Element *suppElem = liste->premier;
    if(liste->cnt == 1)
    {
      liste->premier = NULL;
      liste->dernier = NULL;
      free(suppElem);
    }
    else
    {
      liste->premier = liste->premier->suivant;
      liste->premier->precedent = NULL;
      free(suppElem);
    }
    liste->cnt--;
  }
}
```





Retrait à la fin de la liste

SI la liste n'est pas vide ALORS
    On fait pointer un pointeur sur le dernier élément de la liste.
    Si il y a qu' un seul élément ALORS
        On fait pointer la tête et la queue de liste sur NULL.
        On détruit l' élément pointé par le pointeur.
    Sinon
        On fait pointer la queue de liste sur l'avant dernier élément
        On fait pointer le nouveau élément en queue de la liste vers NULL (élément suivant)
        On détruit l' élément pointé par le pointeur.
    FinSI
    On décrémente le nombre d' éléments.
FINSI

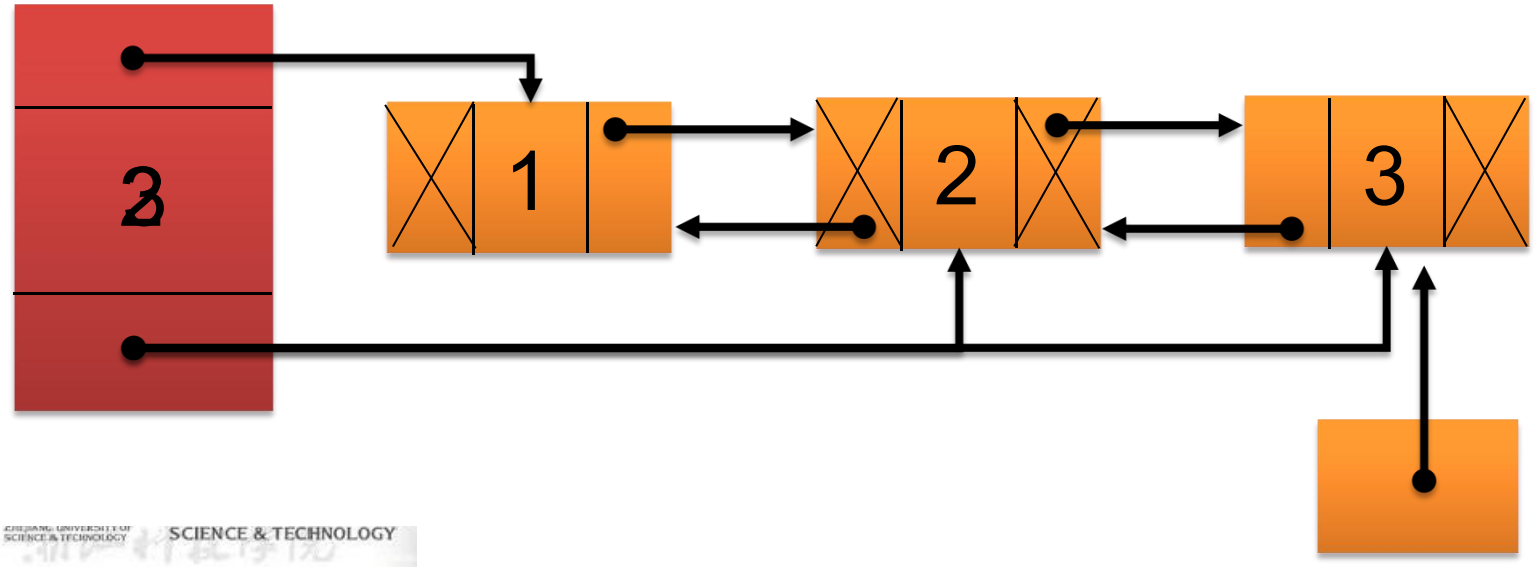


Retrait à la fin de la liste

SI la liste n'est pas vide ALORS
    On fait pointer un pointeur sur le dernier élément de la liste.
    Si il y a qu' un seul élément ALORS
        On fait pointer la tête et la queue de liste sur NULL.
        On détruit l' élément pointé par le pointeur.

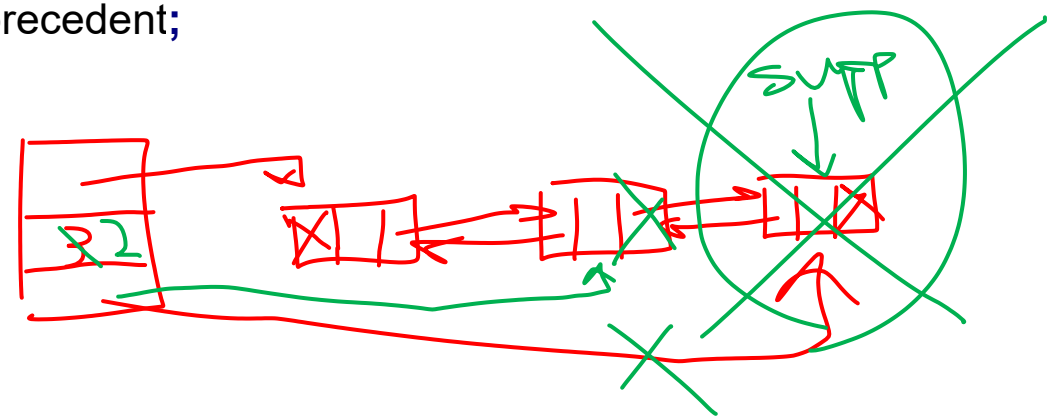
pas de parcour

    Sinon
        On fait pointer la queue de liste sur l'avant dernier élément
        On fait pointer le nouveau élément en queue de la liste vers NULL (élément suivant)
        On détruit l' élément pointé par le pointeur.
    FinSI
    On décrémente le nombre d' éléments.
FINSI



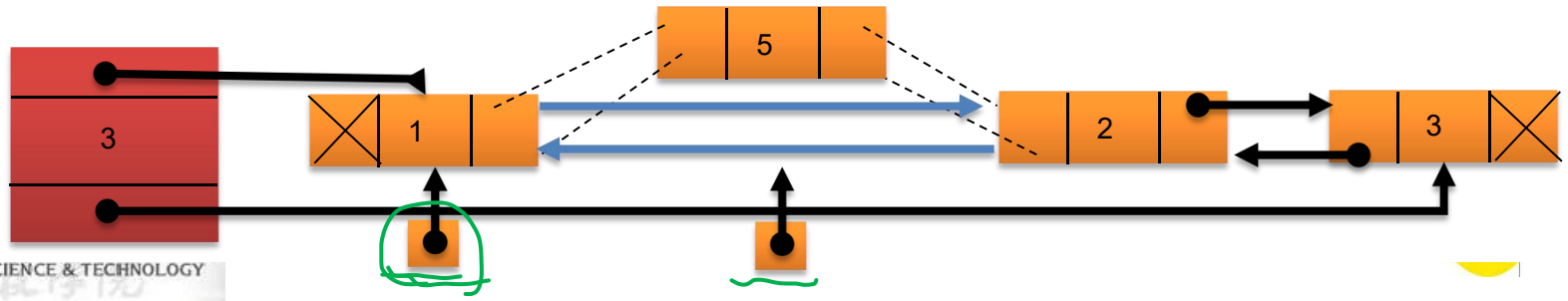
- **Implémentation C d'une liste simplement chaînée en utilisant deux structures**

```
void suppQueue(Liste *liste)
{
  if(liste)
  {
    Element *suppElem = liste->dernier;
    if(liste->cnt == 1)
    {
      liste->premier = NULL;
      liste->dernier = NULL;
      free(suppElem);
    }
    else
    {
      liste->dernier = liste->dernier->precedent;
      liste->dernier->suivant = NULL;
      free(suppElem);
    }
    liste->cnt--;
  }
}
```



Retrait à la ième position depuis la tête de la liste

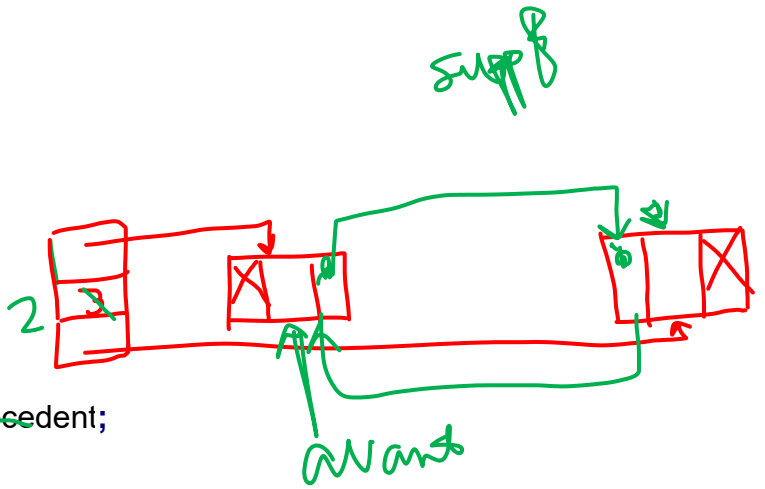
SI la liste n'est pas vide ALORS
  On fait pointer un pointeur sur le dernier élément de la liste.
  Si il y a qu' un seul élément ou i = 0 ALORS
    On fait pointer la tête et la queue de liste sur NULL.
    On détruit l' élément pointé par le pointeur.
  Sinon SI 0 < i < nombre d' éléments de la liste -1 ALORS
    Nous positionnons un pointeur sur le i-1ème élément.
    Nous positionnons un pointeur sur le ième élément.
    Nous faisons pointer le i-1ème élément sur le i+1ème élément.
    Nous faisons pointer le i+1ème élément sur le i-1ème élément.
    On détruit le ième élément.
  Sinon Si i = 0 ALORS
    On retrait au début de la liste
  Sinon
    On retrait à la fin de la liste
  Fin si
  Nous incrémentons le nombre des éléments
Fin SI



LISTES DOUBLEMENT CHAINÉES

- **Implémentation C d'une liste simplement chaînée en utilisant deux structures**

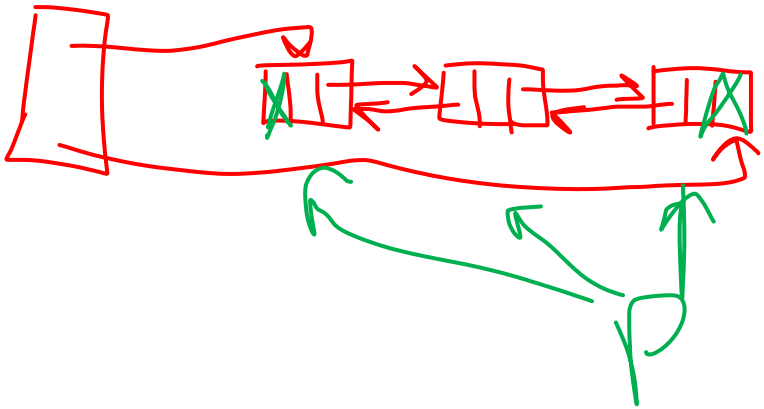
```
void suppElemPos(Liste *liste, int i)
{
 if(liste || i>=0)
 {
  if(liste->cnt == 1)
  {
   free(liste->premier);
   liste->premier = NULL;
   liste->dernier = NULL;
  }
  else if(i > 0 && i < liste->cnt-1)
  {
   Element *avant = liste->premier;
   for(int x = 1; x < i; x++)
    avant = avant->suivant;
   Element *suppElem = avant->suivant;
   suppElem->suivant->precedent = suppElem->precedent;
   avant->suivant = suppElem->suivant;
   free(suppElem);
  }
  else if(i==0)
  {
   suppTete(liste);
   return;
  }
  else
  {
   suppQueue(liste);
   return;
  }
  liste->cnt--;
 }
}
```



LISTES DOUBLEMENT CHAINÉES

- **Affichage d'une liste doublement chainée**

```
void afficherApartirDebut(Liste *liste)
{
  Element *p;
  p=liste->premier;
  while(p)
  {
    printf("%d ->",p->nombre);
    p=p->suivant;
  }
  printf("NULL\n");
}
```



```
void afficherApartirFin(Liste *liste)
{
  Element*p;
  p=liste->dernier;
  while(p)
  {
    printf("%d ->",p->nombre);
    p=p->precedent;
  }
  printf("NULL\n");
}
```

LISTES DOUBLEMENT CHAINÉES

- **Programme principale**