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

Programmation & Algorithmique II

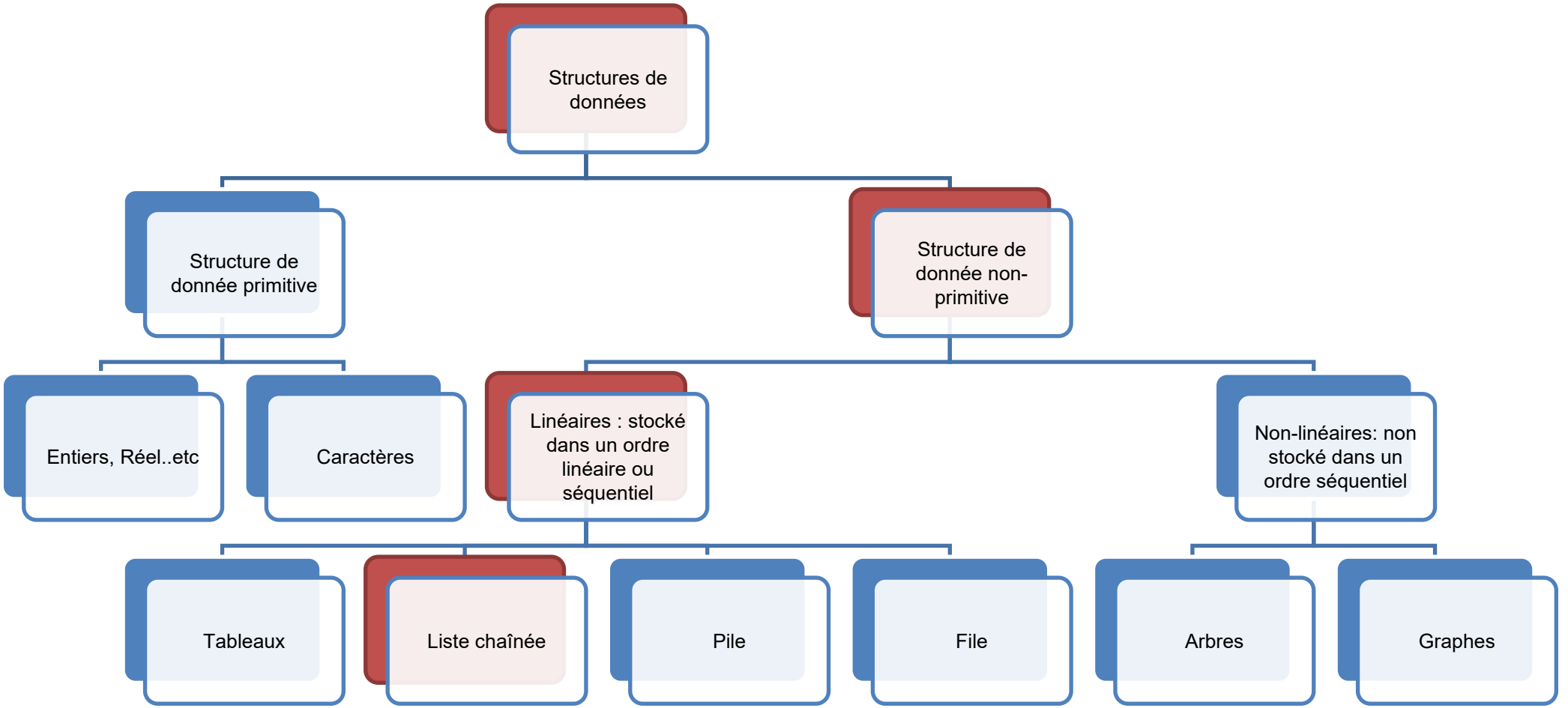
CM 4 : listes chaînées (1)

浙江科技學院 ZHEJIANG UNIVERSITY OF SCIENCE & TECHNOLOGY

CERGY PARIS UNIVERSITÉ

CLASSIFICATION DES STRUCTURES DE DONNÉES

- **Structures de données primitives et non primitives**

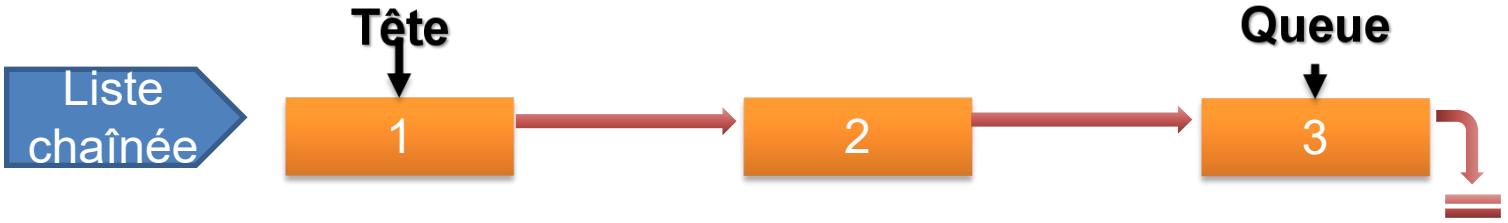


PLAN

- Les Listes simplement chaînées.
  - La représentations graphique
  - Les caractéristiques des listes chaînées
  - Les services des listes chaînées
    - Création
    - Ajout
    - Suppression
    - Consultation/recherche

- **Liste chaînée**
  - Une structure de données linéaire qui consiste en un groupe de nœuds dans une séquence.
    - Chaque nœud contient ses propres données et l'adresse du nœud suivant formant ainsi une structure semblable à une chaîne.
    - Les listes liées sont utilisées pour créer des arbres et des graphes.

- **Qu'est-ce qu'une liste chainée ?**
  - Une liste chaînée est une structure de données à l'intérieure de laquelle les objets sont ordonnés de façon linéaire.

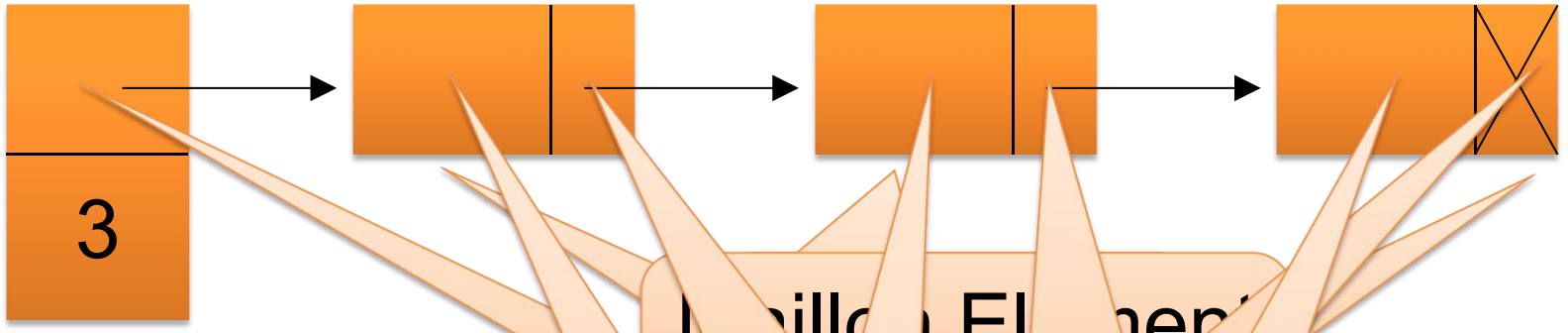


  - À la différence d'un tableau, où l'ordre linéaire est déterminé par les indices du tableau, l'ordre dans une liste chaînée est déterminé par un pointeur dans chacun des objets.

Contrairement aux tableaux, les éléments d'une liste chaînée ne sont pas placés côte à côte dans la mémoire. Chaque case pointe vers une autre case en mémoire qui n'est pas nécessairement stockée juste à côté.

LISTES CHAINÉES

- **Représentation simpliste d'une liste simplement chaînée**



Structure de contrôle

Données (toutes de même type)

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

```c
#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
};

//2) La structure de contrôle
typedef struct Liste Liste;
struct Liste
{
        Element *premier; // pointeur vers le premier élément de la liste
        int cnt = 0;   // compteur des éléments de la liste
};
```

- **Caractéristiques d'une liste simplement chaînée**
  - Pour consulter le ième élément, il faut consulter les i-1 premiers éléments.
  - L'ajout ou le retrait d'un nœud dans la liste ne nécessite le déplacement d'aucune donnée en mémoire.
  - Les données n'ont pas besoin d'être (et ne sont probablement pas) contigües en mémoire.

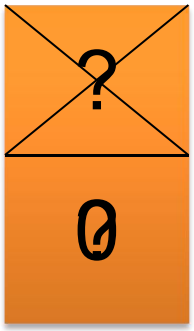
- **Les services à offrir dans une liste chaînée**
  - Une liste chaînée devrait minimalement offrir les fonctionnalités suivantes :
    - Création
    - Ajout (au début, à la fin et ailleurs)
    - Retrait (au début, à la fin et ailleurs)
    - Consultation (au début, à la fin et ailleurs)
    - Consultation de la longueur

- **Fonctionnement des différents services**
  - Les diapositives qui suivent illustrent le fonctionnement des différents services d'une liste simplement chaînée.
  - Il existe plusieurs façon d'implanter une liste. Celle qui a été choisie pose que le premier nœud possède l'indice **0** et que le dernier nœud possède l'indice $n - 1$ où n est le nombre de nœuds de la liste.

Création de la liste

On fait pointer la tête vers NULL.

On initialise le nombre de nœuds à 0.



- **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->cnt = 0;

   return liste;
}
```

Ajout au début de la liste

On tente de créer un nouveau nœud.

Si la création est un succès ALORS

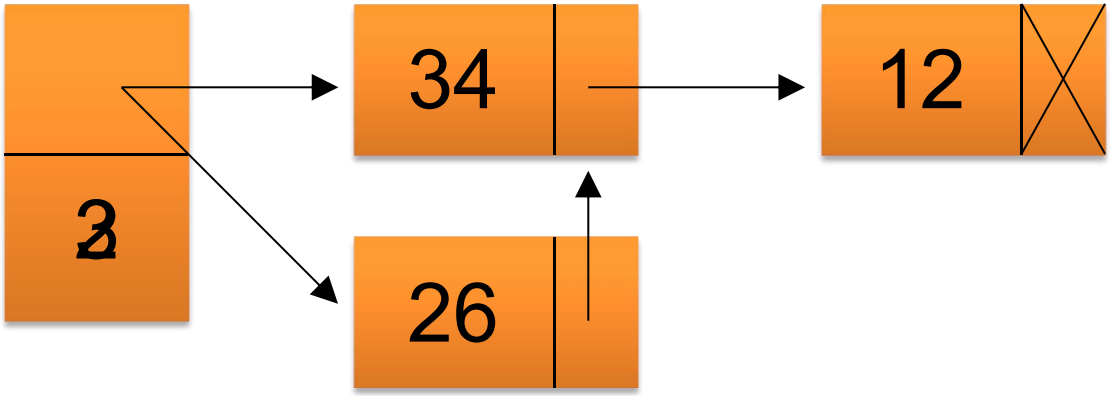
    On fait pointer le nouveau nœud vers le premier élément de la liste (NULL si celle-ci est vide).

    On initialise le nœud.

    On fait pointer la tête de liste sur le nouveau nœud.

    On incrémente le nombre de nœuds.

FINSI



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

```
void inserTete(Liste *liste, int nvNombre)
{
   /* Création du nouvel élément */
   // Element *nouveau = (Element *)malloc(sizeof(Element));
   Element *nouveau = malloc(sizeof(*nouveau));
   if (liste == NULL || nouveau == NULL)
   {
      exit(EXIT_FAILURE);
   }

   /* Insertion de l'élément au début de la liste */
   nouveau->suivant = liste->premier;
   nouveau->nombre = nvNombre;
   liste->premier = nouveau;
   liste->cnt +=1;
}
```

QUIZ

Question : Est-ce que le segment de code dessous est correct?

Choix:

A- Oui

B- Non

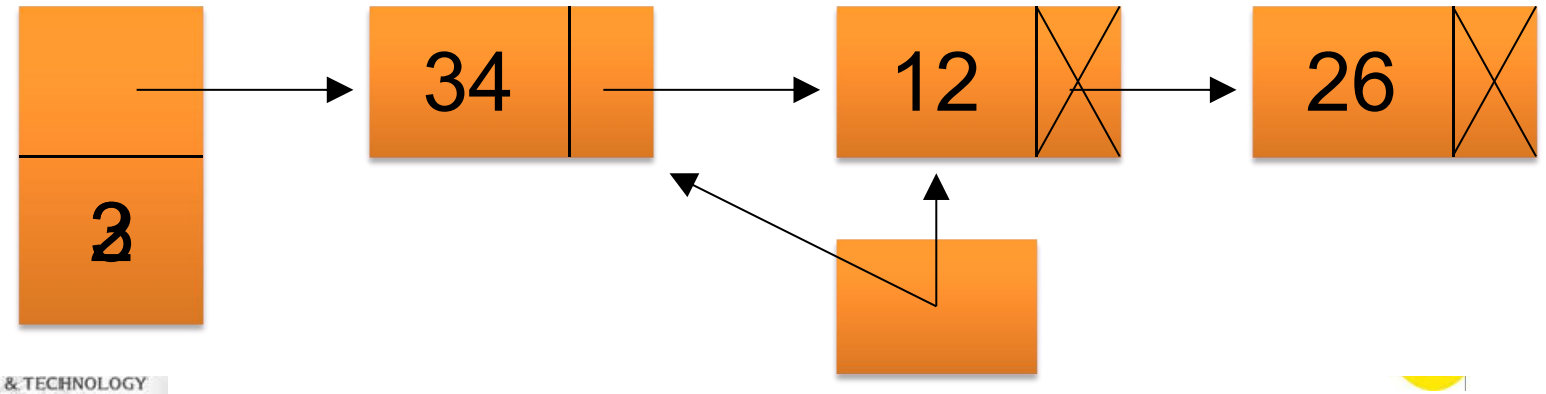
5 minute

```
void inserTete(Liste *liste, int nvNombre)
{
    /* Création du nouvel élément */
    // Element *nouveau = (Element *)malloc(sizeof(Element));
    Element *nouveau = malloc(sizeof(*nouveau));
    if (liste == NULL || nouveau == NULL)
    {
        exit(EXIT_FAILURE);
    }
    nouveau->nombre = nvNombre;

    /* Insertion de l'élément au début de la liste */
    liste->premier = nouveau;
    nouveau->suivant = liste->premier;
    liste->cnt +=1;
}
```

Ajout à la fin de la liste

SI la liste est vide ALORS
    On ajoute au début de la liste
SINON
    On tente de créer un nouveau nœud.
    SI la création est un succès ALORS
        On positionne un pointeur sur le dernier élément de la liste (voir consulter $i^{ème}$ élément).
        On initialise les champs du nouveau nœud.
        On fait pointer le dernier nœud sur le nouveau nœud.
        On incrémente le nombre de nœuds.
    FINSI
FINSI



LISTES CHAINÉES

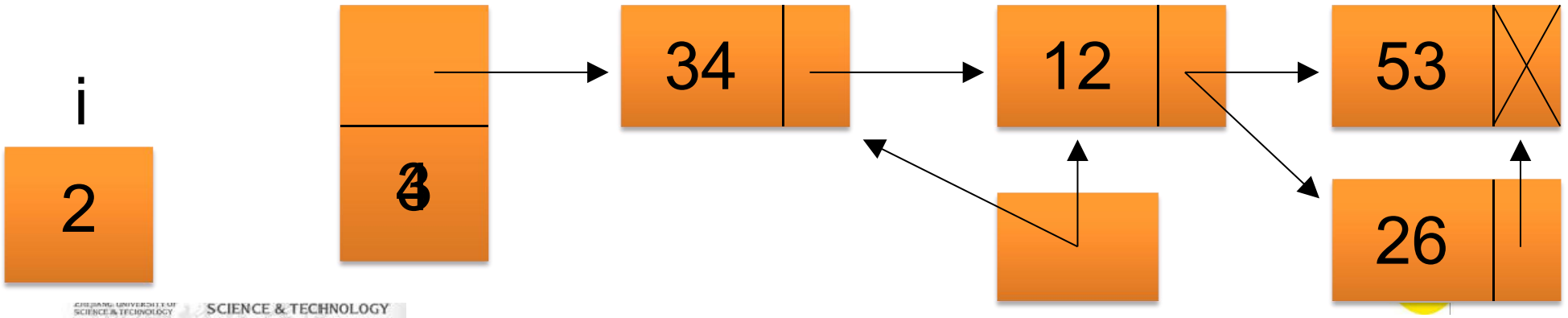
- **Implémentation C d'une liste simplement chaînée en utilisant deux structures**

```
void inserQueue(Liste *liste, int nvNombre){
    /* verification que la liste n'est pas vide*/
    if (liste->premier == NULL){
        printf("\nLa liste est vide\n");
        inserTete(liste, nvNombre);
    }
    /* création du nouvel élément*/
    Element *nouveau = (Element *)malloc(sizeof(Element));
    if (nouveau == NULL){
        printf("\nerreur d'allocation mémoire\n");
        exit(0);
    }
    nouveau->nombre = nvNombre;
    nouveau->suivant = NULL;

    /* rattachement de le nouveau élément à la fin de queue de la liste */
    Element *actuel = liste->premier;
    while(actuel->suivant != NULL){
        actuel = actuel->suivant;
    }
    actuel->suivant = nouveau;
    liste->cnt++;
}
```

Ajout à la ième position

SI i = 0 OU la liste est vide ALORS
    On ajoute au début de la liste
SINON SI i < nombre de nœuds de la liste ALORS
    On tente de créer un nouveau nœud.
    SI la création est un succès ALORS
        On positionne un pointeur sur le i-1ème nœud.
        On initialise les données du nouveau nœud.
        On fait pointer le nouveau nœud sur le ième nœud (NULL si on ajoute à la fin).
        On fait pointer le i-1ème nœud sur le nouveau.
        On incrémente le nombre de nœuds.
    FINSI
FINSI



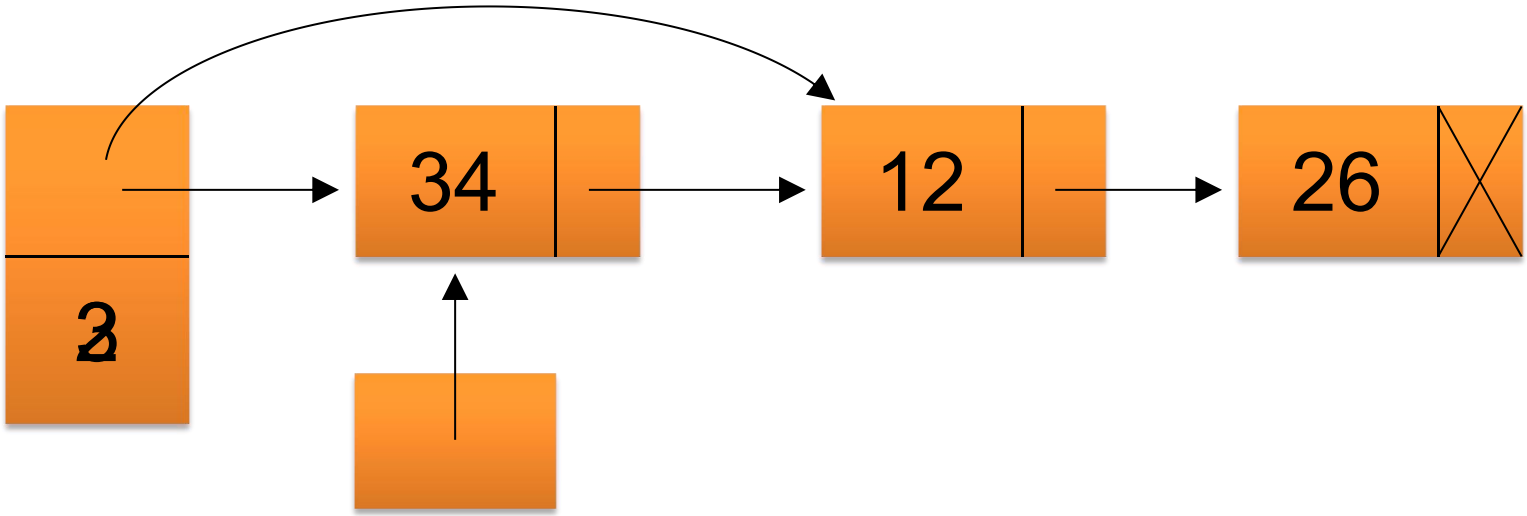
LISTES CHAINÉES

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

```
void inserPos(Liste *liste, int v, int i){
    if (liste->premier == NULL || i == 0){
        inserTete(liste, v);
    }
    else{
            if (i < liste->cnt){
                    Element *nouveau = (Element *)malloc(sizeof(Element));
                    if (nouveau == NULL){
                            printf("\nerreur d'allocation mémoire\n");
                            exit(0);
                    }
                    Element *actuel = liste->premier;
                    for (int x = 1; x<i; x++ ){
                            actuel = actuel->suivant;
                     }
                     nouveau->nombre = v;
                     nouveau->suivant = actuel->suivant;
                     actuel->suivant = nouveau;
                     liste ->cnt +=1;
            }
            else{
                    printf("la posisition est plus grand que le nombre des éléments --> inserQueue");
                    inserQueue(liste, v);
            }
    }
}
```

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.
    On fait pointer la tête de liste sur le deuxième nœud (NULL s'il n'y avait qu'un seul nœud).
    On détruit le nœud pointé par le pointeur.
    On décrémente le nombre de nœuds.
FINSI



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

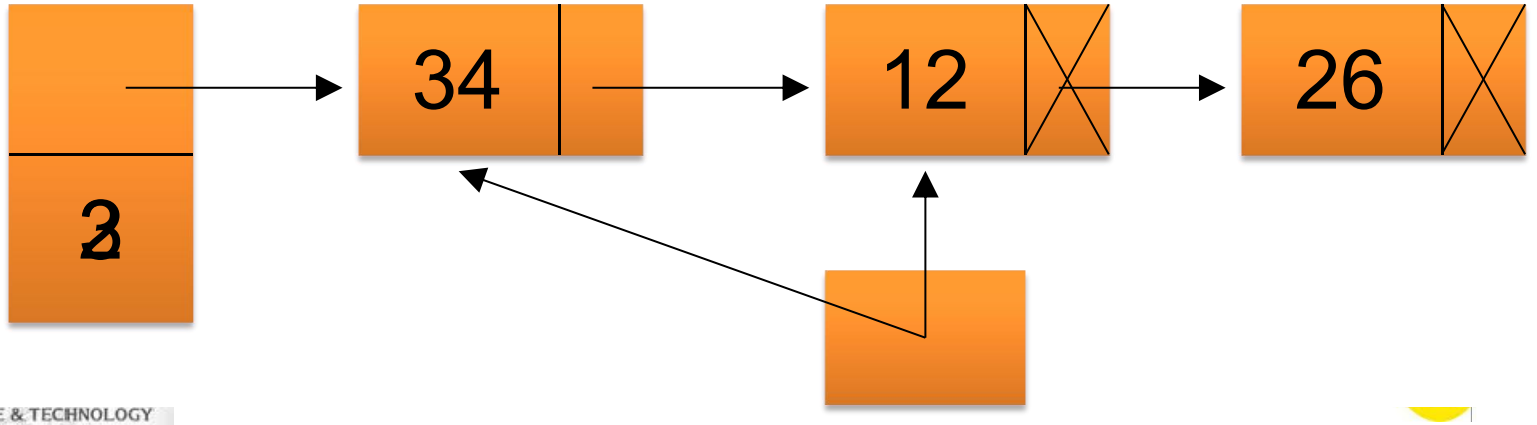
```
void suppTete(Liste *liste){
    if (liste->premier == NULL){
        printf("\nLa liste est vide \n");
        exit(0);
    }

    Element *suppE = liste->premier;
    liste->premier = liste->premier->suivant;
    free(suppE);
    liste->cnt--;

}
```

Retrait à la fin de la liste

SI la liste n'est pas vide ALORS
    SI la liste ne contient qu'un seul élément ALORS
        Retirer au début de la liste.
    SINON
        On fait pointer un pointeur sur l'élément à l'indice nombre de nœuds – 1.
        On détruit le dernier nœud.
        On fait pointer l'avant dernier vers NULL.
        On décrémente le nombre de nœuds.
    FINSI
FINSI



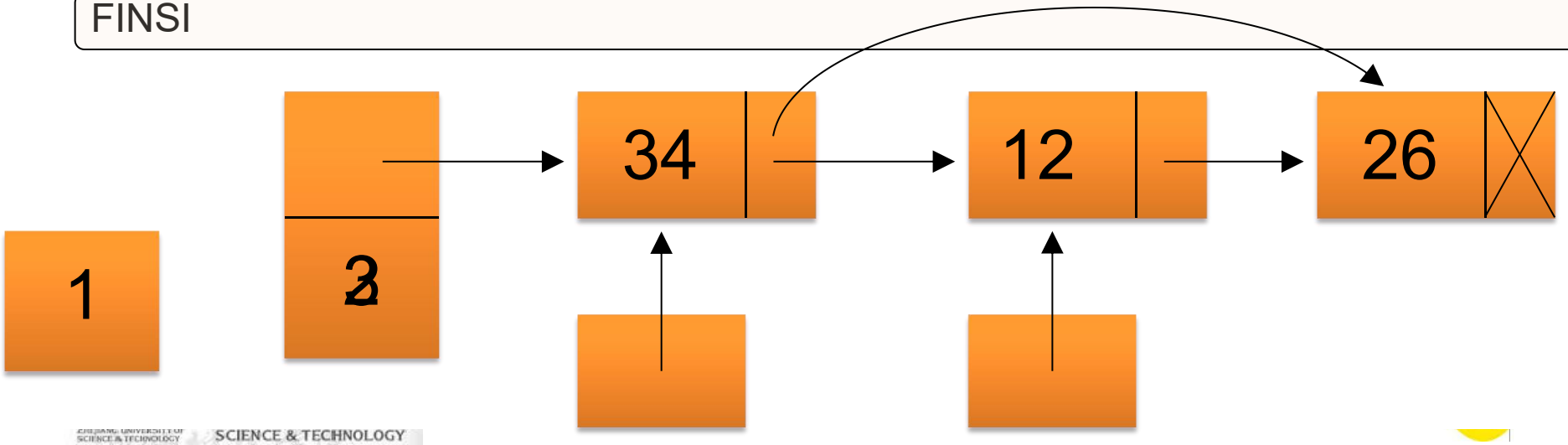
- **Implémentation C d'une liste simplement chaînée en utilisant deux structures**

```
void suppQueue(Liste *liste){
   /* verification que la liste n'est pas vide*/
   if (liste->premier == NULL){
      printf("\nLa liste est vide\n");
      exit(0);
   }
   /* suppression de le dernier élément de la liste */
   if (liste->cnt == 1)
                        suppTete(liste);
   else{


                        Element *actuel = liste->premier;
                        while(actuel->suivant->suivant != NULL){
                                 actuel = actuel->suivant;
                        }
                        free(achuel->suivant);
                        actuel->suivant = NULL;
                        liste->cnt--;
            }
}
```

Retrait à la ième position

SI i ≥ 0 et i < nombre de nœuds de la liste ALORS
    SI la liste ne contient qu'un seul élément ALORS
        Retirer au début de la liste.
    SINON
        On fait pointer un pointeur sur l'élément à l'indice i – 1.
        On fait pointer un pointeur sur l'élément à l'indice i.
        On fait pointer le i-1ème nœud sur le i+1ème nœud (NULL si on détruit le dernier).
        On détruit le ième nœud.
        On décrémente le nombre de nœuds.
    FINSI
FINSI

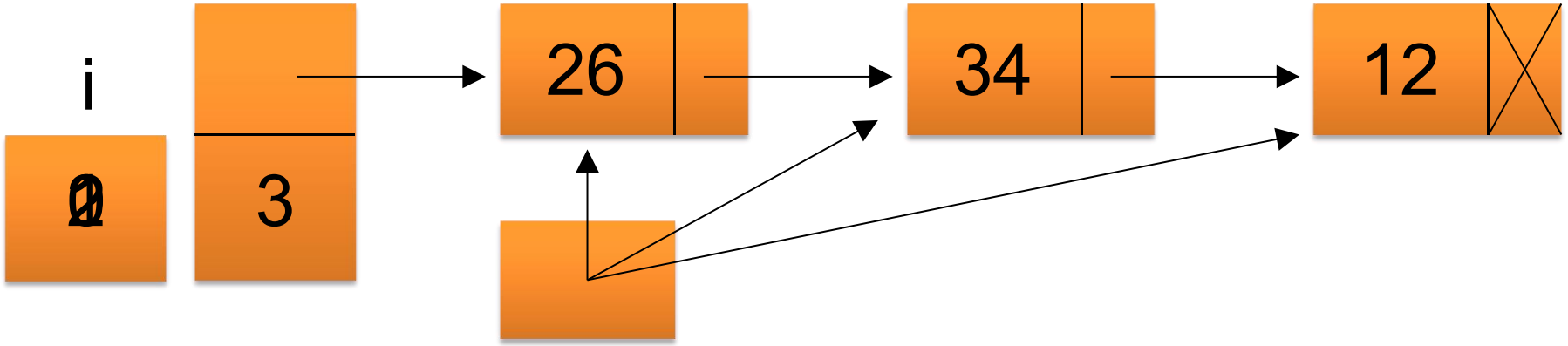


- **Implémentation C d'une liste simplement chaînée en utilisant deux structures**

```
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--;
}
```

Consultation du ième élément

```
SI i ≥ 0 et i < nombre de nœuds de la liste ALORS
    On fait pointer un pointeur sur le premier nœud.
    TANT QUE i > 0 BOUCLE
        Faire pointer le pointeur vers le prochain nœud.
        i ← i - 1
    FIN TANT QUE
FINSI
```



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

```
int consultElem(Liste *liste, int i){

        if (i >= 0 && i < liste->cnt)
        {
                Element *actuel = liste->premier;
                while(i>0)
                {
                        actuel = actuel->suivant;
                        i--;
                }
        return actuel->nombre;
        }
        else exit(0);
}
```