Chapitre 3 : Les listes chaînées

Hassen NAKBI ::: hassen.nakbi@gmail.com

1^{ere} année Licence

4 mars 2024

Introduction

Les structures de données

- Pour mémoriser une collection des données, nous avons utilisé des tableaux, chaînes et enregistrements.
- Les tableaux sont adaptés pour les cas où les éléments sont du même type et où les accès aux éléments sont fréquents.
- Les enregistrements sont utiles pour stocker des objets avec plusieurs attributs.
- Les tableaux et enregistrements possèdent des limites et inconvénients tels que l'impossibilité d'ajouter un nouvel élément au début, au milieu ou à la fin.

2 Liste chaînée

- 2.1 Définition
- 2.2 Caractéristiques
- 2.3 Types d'une liste chaînée
- 2.4 Définition algorithmique et programmation

Notion de liste chaînée

2.1 Définition d'une liste chaînée

Une **liste chaînée** est une structure de données correspondant à une suite d'éléments, qui ne sont pas indexés dans la liste, mais pour chaque élément (sauf le dernier) on sait où il se trouve l'élément suivant.

Par conséquent, on ne peut pas accéder à un élément particulier sans passer par le premier élément de la liste et en parcourant tous les éléments jusqu'à ce qu'on atteigne l'élément recherché.

Caractéristiques

2.2 Caractéristiques

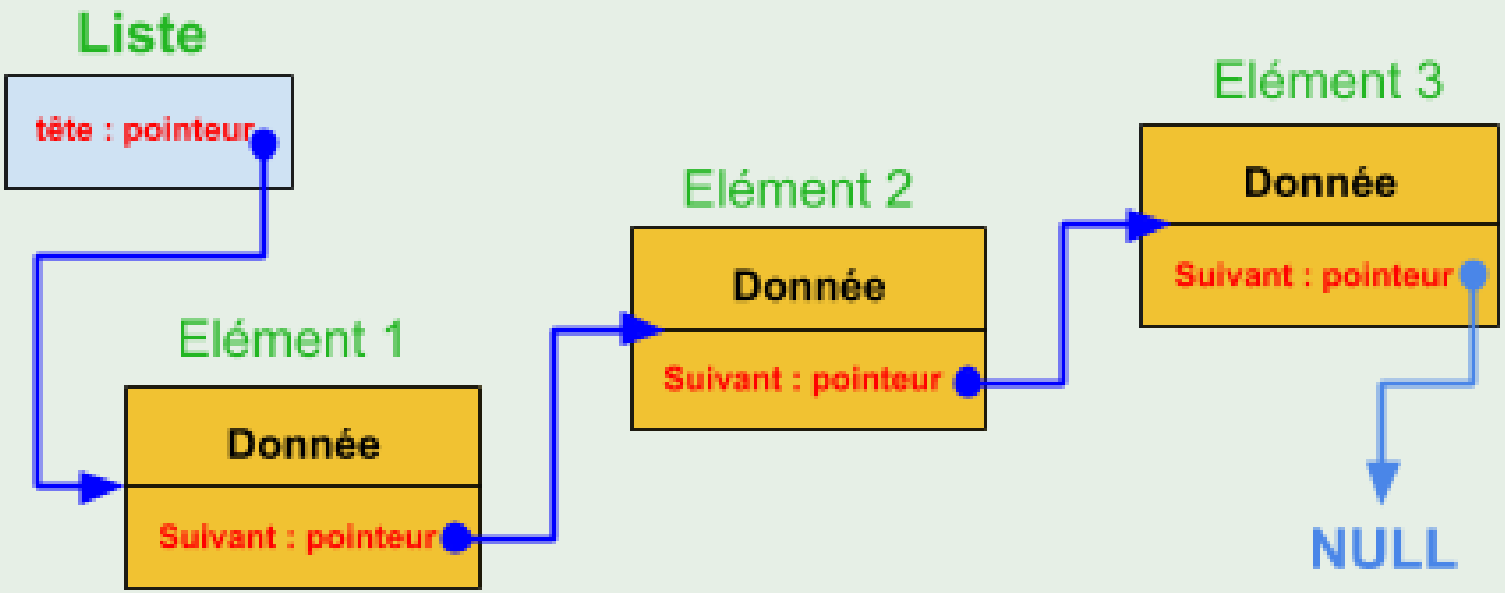
- Une **liste simplement chaînée** est modélisée par un pointeur de début appelée tête de même type que l'élément, afin de mémoriser l'adresse de premier élément.
- Si la liste chaînée est vide, le pointeur tête contient l'adresse NIL.
- Chaque élément de liste est un enregistrement composé d'une donnée de type simple ou bien structuré et un pointeur suivant pour mémoriser l'adresse de l'élément suivant.

Caractéristiques

2.2 Caractéristiques

- Le dernier élément de la liste n'a pas successeur, son champ pointeur suivant contient **l'adresse NIL**.
- Les éléments et les pointeurs de la liste sont de **même types**.

Exemple d'une liste chaînée

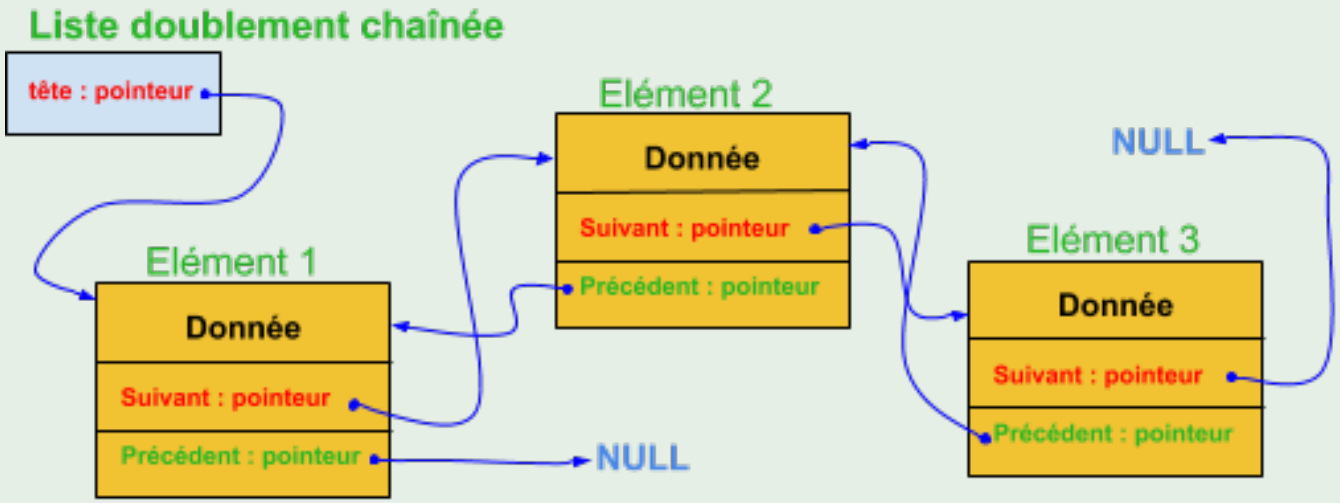


Types d'une liste chaînée

2.3 Types

1. Liste chaînée simple : chaque élément contient un pointeur vers l'élément suivant.
2. Liste chaînée double : chaque élément contient un pointeur vers l'élément suivant et un pointeur vers l'élément précédent. Cela permet de parcourir la liste dans les deux sens et facilite certaines opérations, comme l'insertion d'un élément avant un autre élément.
3. Liste chaînée circulaire : dans ce type de liste chaînée, le dernier élément est lié au premier élément, créant ainsi une boucle.

Exemple d'une liste doublement chaînée



Définition algorithmique

Syntaxe en algorithmique

TYPE Liste = structure
champ1 : type1
champ2 : type2
suivant : ^**Liste**
Fin structure

VAR tete :^Liste
DÉBUT
tete ⟵ NIL
FIN

Exemple : liste chaînée des éléments chiffre

TYPE chiffre= structure

val : entier

suivant : ^**chiffre**

Fin structure

VAR tete : ^chiffre

DÉBUT

tete ⟵ NIL

FIN

Liste chaînée en Langage C

Déclaration en Langage C

```
typedef struct list {
        type1 champ1;
        type2 champ2;
struct list * suivant;
} Liste;

Liste * tete;
```

```
typedef struct list Liste;
struct list {
        type1 champ1;
        type2 champ2;
Liste * suivant;
} Liste;

Liste * tete;
```

Liste chaînée en Langage C

Exemple

```
typedef struct entier {
        short val;
struct entier * suivant;
} chiffre;

chiffre * tete = NULL;
```

Opérations sur une liste chaînée

1. Ajouter d'un élément au début, au milieu et à la fin.
2. Supprimer un élément au début, au milieu et à la fin.
3. Concaténer deux listes.

Ajout au début de la liste

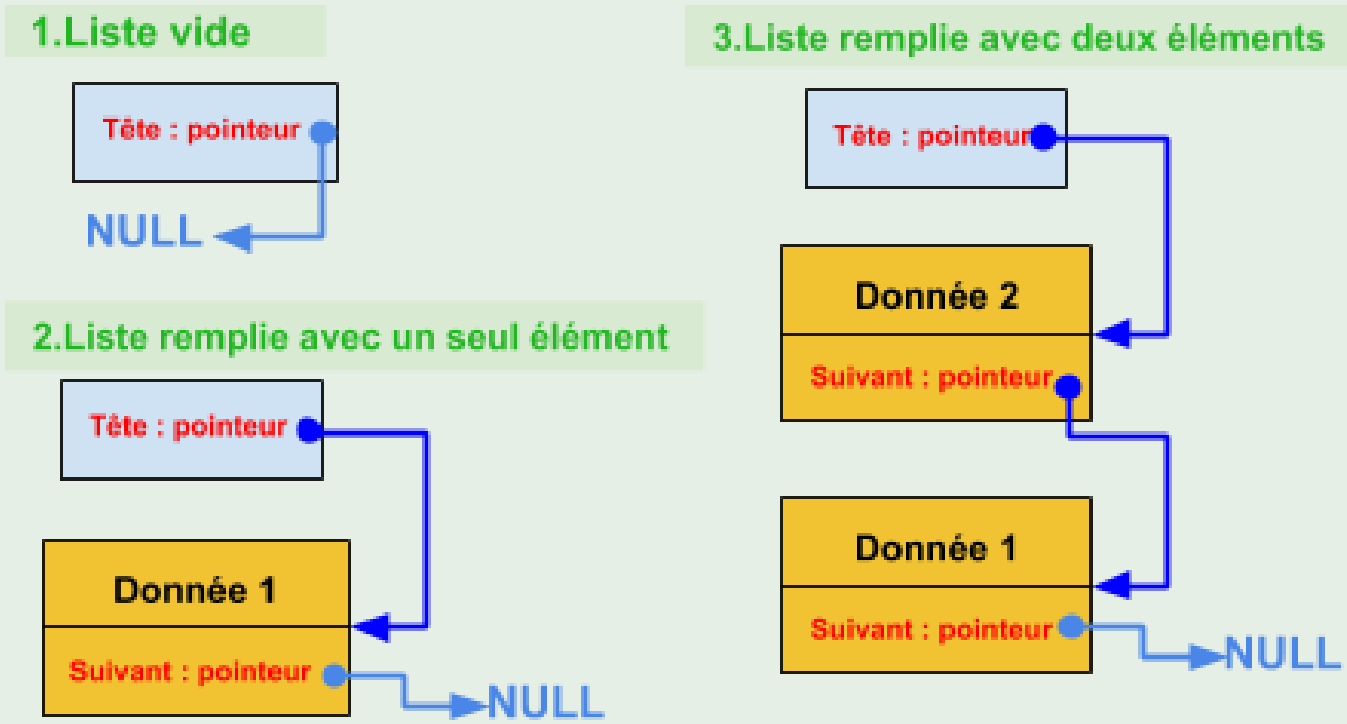
a. Ajout au début de la liste

Pour pouvoir ajouter un nouveau élément au début de la liste chaînée, il fallait suivre les étapes suivantes :

1. Création d'un nouveau élément d'une façon dynamique.
2. Mise à jour de **pointeur suivant** de nouveau élément vers l'adresse contenue dans le **pointeur tête**.
3. Mise à jour de **pointeur tête** de la liste vers l'adresse de nouveau élément.

Ajout au début de la liste

a. Ajout au début de la liste



Solution Algorithmique

Module de création d'un nouveau élément

Fonction nouveau(e : entier) : ^chiffre
VAR element : ^chiffre
DÉBUT
element $\longleftarrow$ allouer(chiffre)
(element$\longrightarrow$val) $\longleftarrow$ e
Renvoyer element
FIN

Solution Algorithmique

Module d'ajout au début de liste

Procédure ajoutdebut(tete :^^chiffre, N :^chiffre)
DÉBUT
(N⟶suivant) ⟵ tete^
tete^ ⟵ N
FIN

Implémentation en Langage C

Module de création d'un nouveau élément

```
chiffre * nouveau(short e){
chiffre * element = malloc(sizeof(chiffre)) ;
(element⟶val)= e ;
return element ;
}
```

Implémentation en Langage C

Module d'ajout au début de liste

```
void ajoutdebut(chiffre **tete,chiffre *N) {
(N⟶suivant) = *tete;
*tete = N;
}
```

Ajout au milieu de la liste

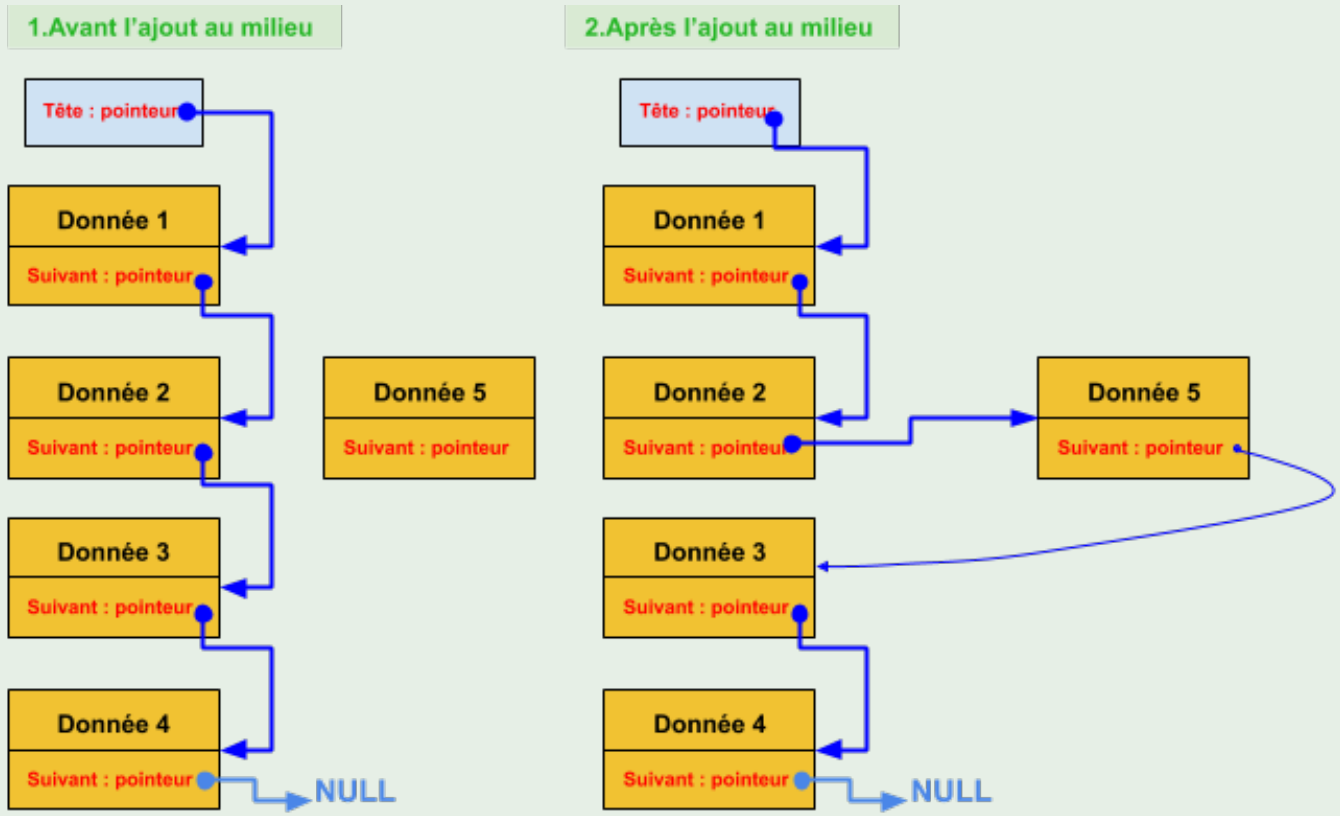
b. Ajout au milieu de la liste

Pour pouvoir ajouter un nouveau élément au milieu de la liste chaînée, il fallait suivre les étapes suivantes :

1. Création d'un nouveau élément d'une façon dynamique.
2. Parcourir la liste en utilisant le pointeur tête, pour atteindre l'adresse de l'élément milieu.
3. Mise à jour de **pointeur suivant** de nouveau élément vers l'adresse **pointeur suivant** de l'élément avant le milieu.
4. Mise à jour de **pointeur suivant** de l'élément avant le milieu vers l'adresse de nouveau élément.

Ajout au milieu de la liste

b. Ajout au milieu de la liste



Solution Algorithmique

Module d'ajout au milieu de liste

Procédure ajoutmilieu(tete :^^chiffre, N :^chiffre)
VAR p :^chiffre, nb : entier
DÉBUT
nb $\longleftarrow$ 0
p $\longleftarrow$ tete^
Tant que (nb $<$ 2) **Faire** (On suppose l'élément milieu existe après deux éléments.)
p $\longleftarrow$ (p$\longrightarrow$suivant)
nb $\longleftarrow$ nb + 1
Fin Tant que
(N$\longrightarrow$suivant) $\longleftarrow$ (p$\longrightarrow$suivant)
(p$\longrightarrow$suivant) $\longleftarrow$ N
FIN

Implémentation en Langage C

Module d'ajout au milieu de liste

```
void ajoutmilieu(chiffre **tete,chiffre *N){
chiffre * p = *tete;
short nb = 0;
while (nb < 2){
p = (p⟶suivant);
nb++;
}
N⟶suivant = (p⟶suivant);
p⟶suivant = N;
}
```

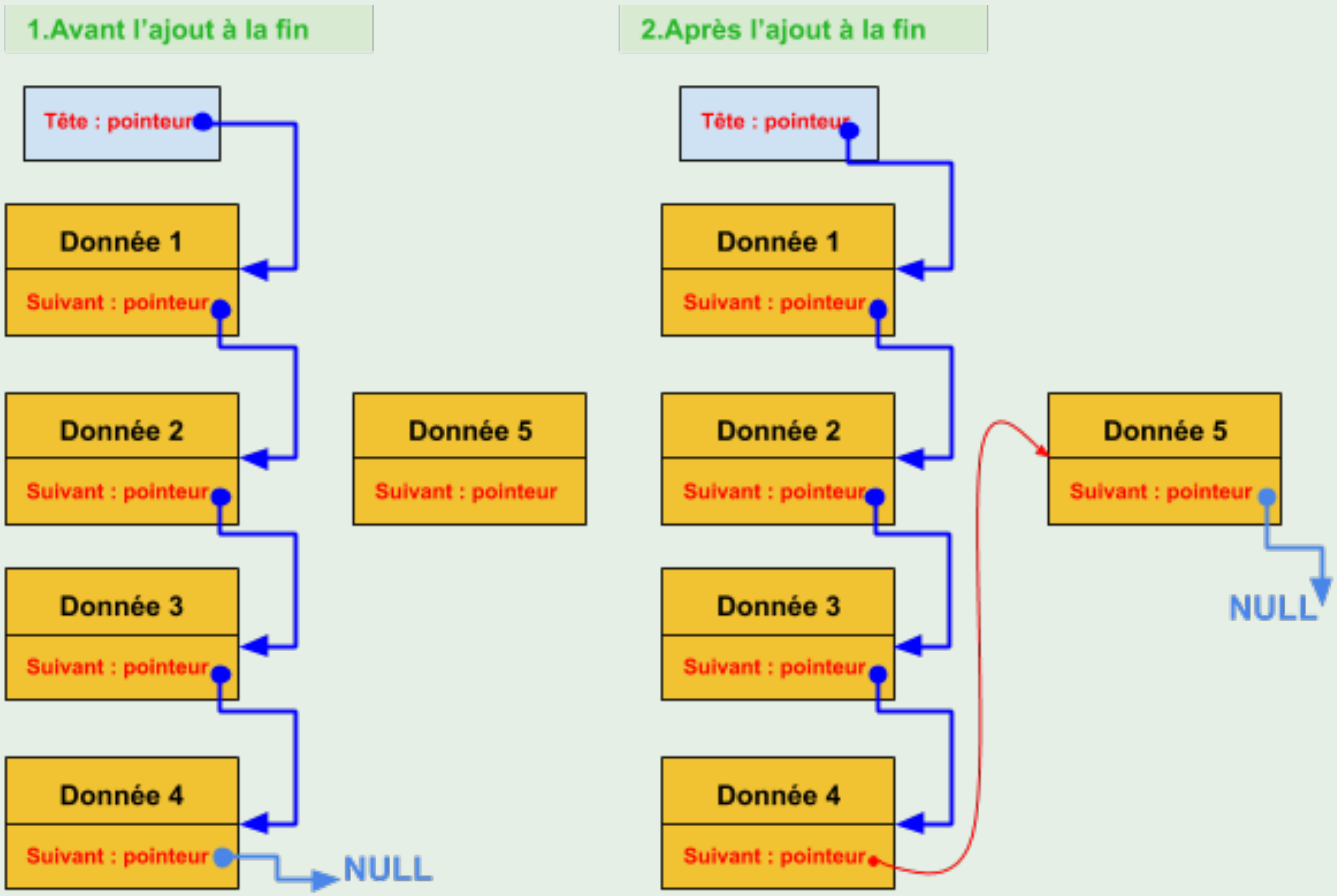
Ajout à la fin de la liste

c. Ajout à la fin de la liste

Pour pouvoir ajouter un nouveau élément à la fin de la liste chaînée, il fallait suivre les étapes suivantes :

1. Création d'un nouveau élément d'une façon dynamique.
2. Parcourir la liste en utilisant le pointeur tête, pour atteindre l'adresse de dernier élément, son **pointeur suivant est nul**.
3. Mise à jour de **pointeur suivant** de dernier élément vers l'adresse de nouveau élément.
4. Mise à jour de **pointeur suivant** de nouveau élément vers l'adresse nulle.

c. Ajout à la fin de liste



Solution Algorithmique

Module d'ajout à la fin de liste

Procédure ajoutfin(tete :^^chiffre, N :^chiffre)
VAR p :^chiffre
DÉBUT
p $\longleftarrow$ tete^
Tant que ((p$\longrightarrow$suivant) $\neq$ NIL) **Faire**
p $\longleftarrow$ (p$\longrightarrow$suivant)
Fin Tant que
(p$\longrightarrow$suivant) $\longleftarrow$ N
(N$\longrightarrow$suivant) $\longleftarrow$ NIL
FIN

Implémentation en Langage C

Module d'ajout à la fin de liste

```
void ajoutfin(chiffre **tete,chiffre *N){
chiffre * p = *tete;
while ((p⟶suivant) != NULL){
p = (p⟶suivant);
}
(p⟶suivant) = N;
(N⟶suivant) = NULL;
}
```

Suppression au début de la liste

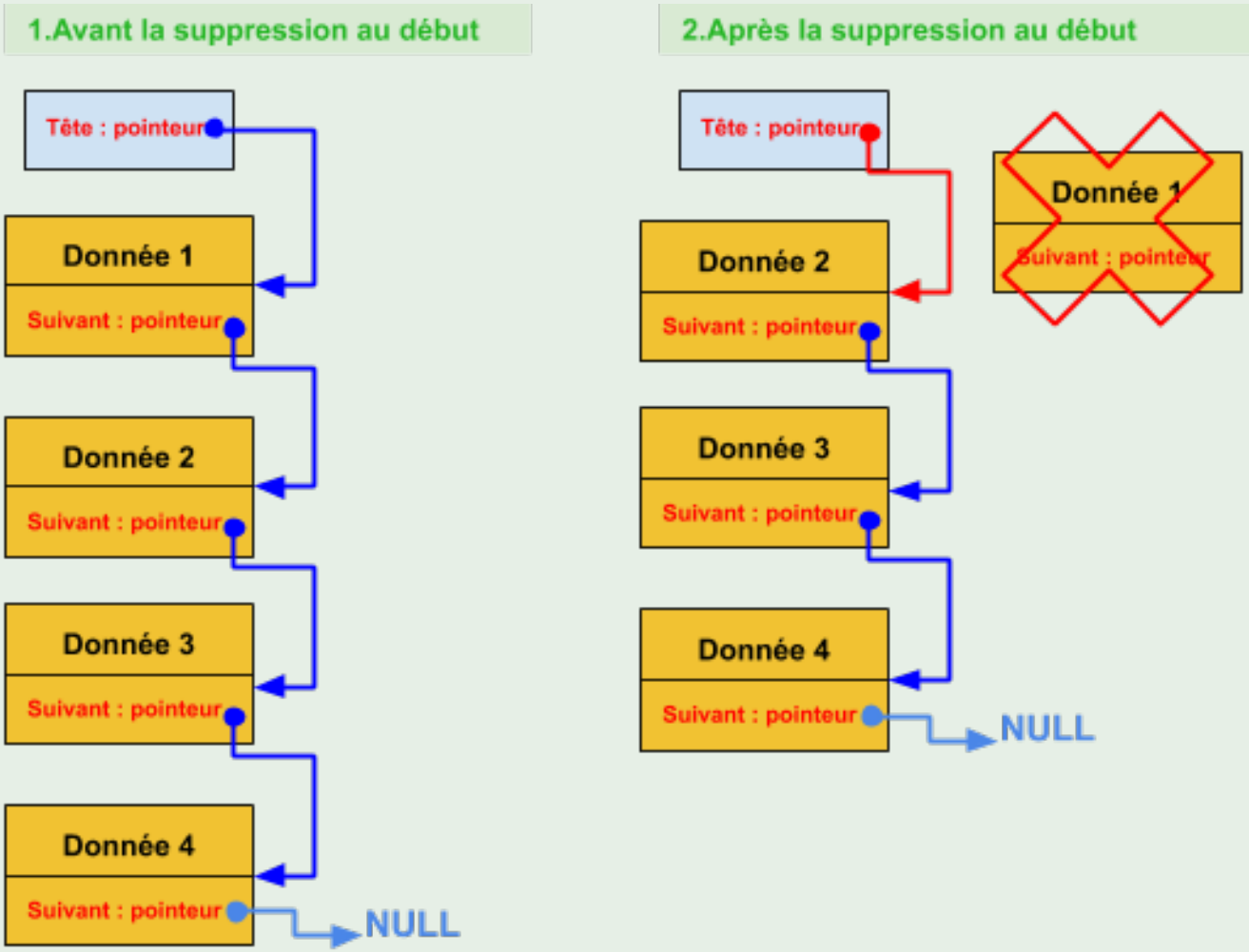
a. Suppression au début

Pour pouvoir supprimer un élément au début de la liste, il fallait suivre les étapes suivantes :

1. Sauvegarde de l'adresse de premier élément contenue le **pointeur tete**.
2. Mise à jour de **pointeur tête** par l'adresse contenue dans le **pointeur suivant** de premier élément.
3. **Libération** de la mémoire pour l'adresse de **premier élément**.

Suppression au début de la liste

a. Suppression au début



Solution Algorithmique

Module de suppression au début

Procédure supprimerdebut(tete :^^chiffre)
VAR sup :^chiffre
DÉBUT
sup $\longleftarrow$ tete^
tete^ $\longleftarrow$ (tete^ $\longrightarrow$suivant)
Libérer(sup)
FIN

Implémentation en Langage C

Module de suppression au début

```
void supprimerdebut(chiffre **tete){
chiffre * sup = *tete;
*tete = (*tete⟶suivant);
free(sup);
}
```

Suppression au milieu de la liste

b. Suppression au milieu

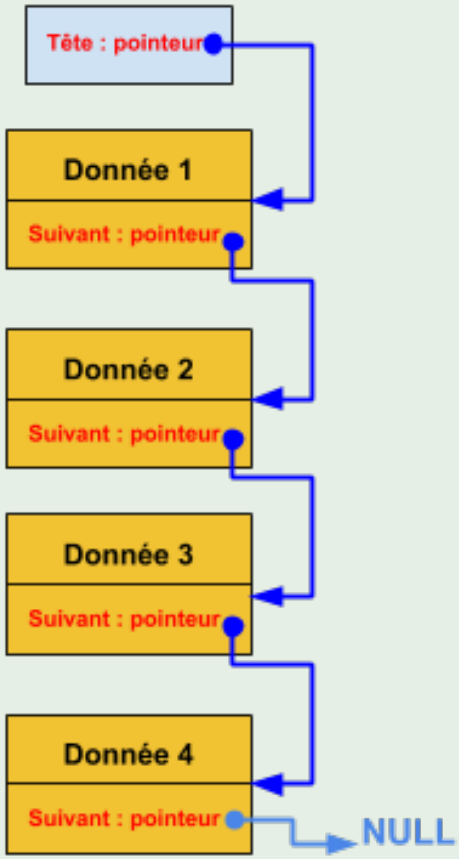
Pour pouvoir supprimer un élément au milieu de la liste, il fallait suivre les étapes suivantes :

1. Parcourir la liste pour chercher l'adresse de **l'élément avant milieu**.
2. Mise à jour de **pointeur suivant** de l'**élément avant milieu** par l'adresse de pointeur suivant après le milieu.
3. **Libération** de la mémoire par l'**adresse de l'élément milieu**.

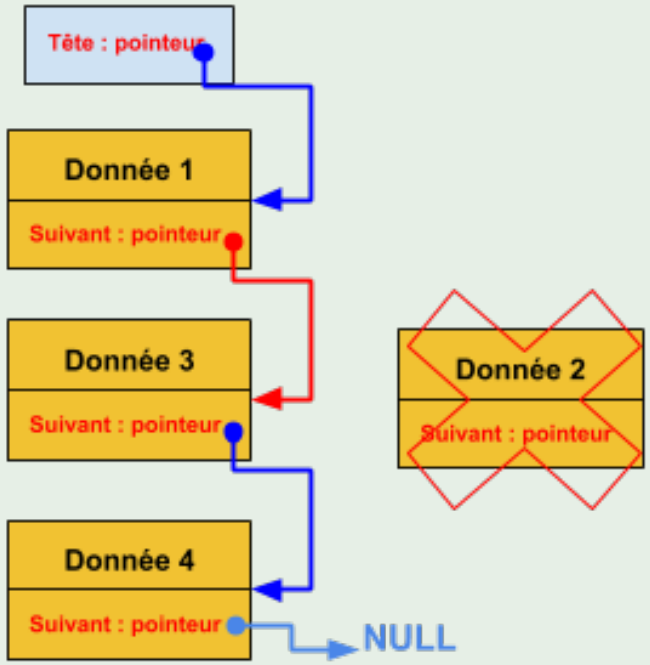
Suppression au milieu de la liste

b. Suppression au milieu

1.Avant la suppression au milieu



2.Après la suppression au milieu



Solution Algorithmique

Module de suppression au milieu

Procédure supprimermilieu(tete :^^chiffre)
VAR sup,p :^chiffre
DÉBUT
sup $\longleftarrow$ (tete^) $\longrightarrow$suivant
p $\longleftarrow$ tete^
(p$\longrightarrow$suivant)$\longleftarrow$(sup$\longrightarrow$suivant)
Libérer(sup)
FIN

Implémentation en Langage C

Module de suppression au milieu

```
void supprimermilieu(chiffre **tete){
chiffre * sup = (*tete)⟶suivant ;
chiffre * p = *tete ;
p⟶suivant = sup⟶suivant ;
free(sup) ;
}
```

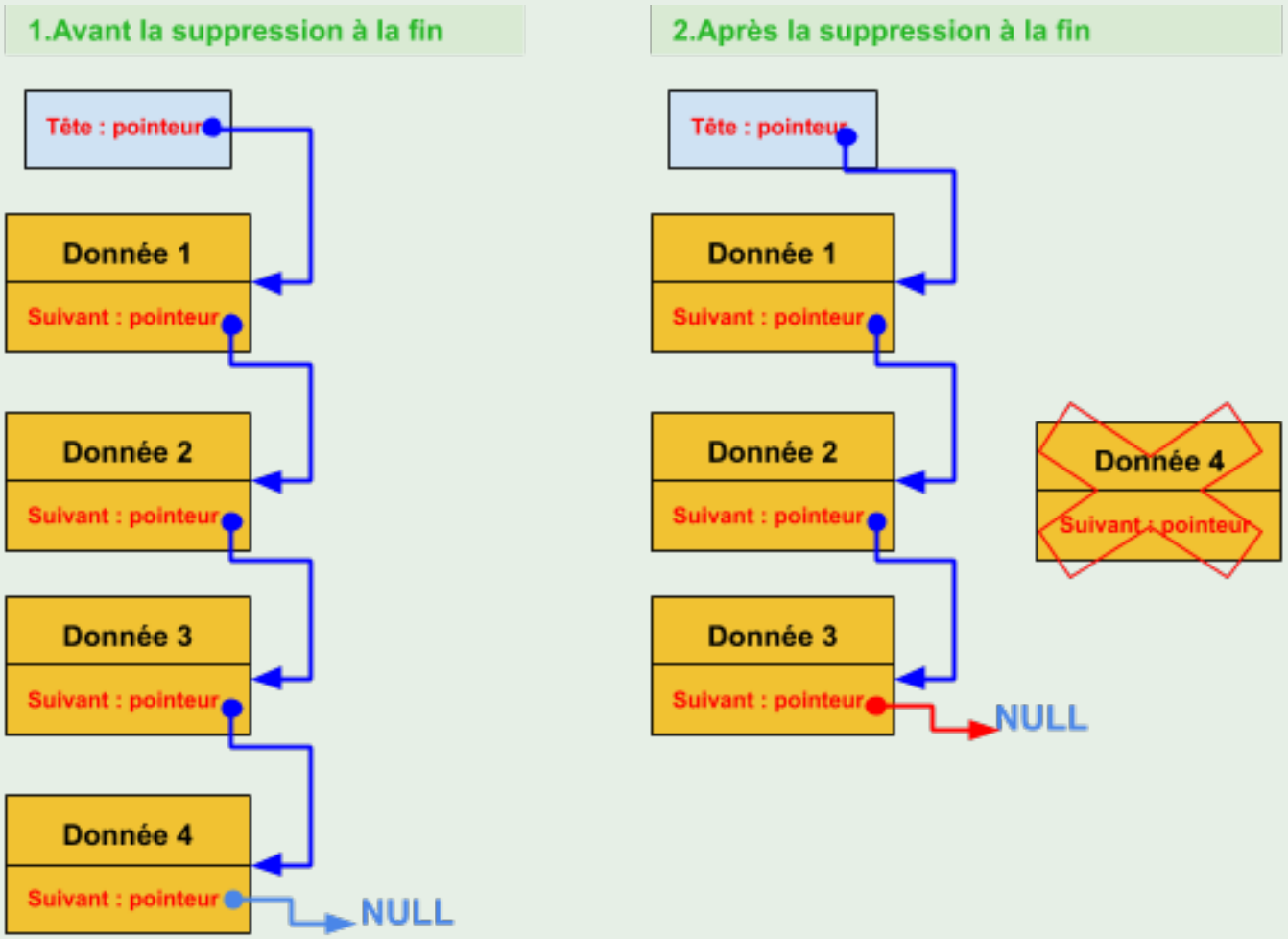
Suppression a la fin de la liste

c. Suppression à la fin

Pour pouvoir supprimer un élément a la fin de la liste, il fallait suivre les étapes suivantes :

1. Parcourir la liste pour chercher **l'avant dernier élément**, le dernier élément contient l'adresse NIL située au niveau pointeur suivant.
2. Mise à jour de pointeur suivant de l'**avant dernier élément** par l'adresse NIL.
3. Libération de la mémoire par l'**adresse de dernier élément**.

c. Suppression à la fin de liste



Solution Algorithmique

Module de suppression à la fin

Procédure supprimerfin(tete :^^chiffre)
VAR sup :^chiffre
DÉBUT
sup ⟵ tete^
Tant que ((sup⟶suivant)⟶suivant≠ NIL) **Faire**
sup ⟵ (sup⟶suivant)
Fin tant que
Libérer(sup⟶suivant)
sup⟶suivant ⟵ NIL ;
FIN

Implémentation en Langage C

Module de suppression à la fin

```
void supprimerfin(chiffre **tete){
chiffre * sup = *tete ;
while ((sup⟶suivant)⟶suivant ! = NULL)
sup = (sup⟶suivant) ;
free(sup⟶suivant) ;
sup⟶suivant = NULL ;
}
```

Concaténation des listes

Concaténation des listes

Pour pouvoir concaténer deux listes, il fallait suivre les étapes suivantes :

1. Recherche de dernier élément de la première liste.
2. Mise à jour de **pointeur suivant** de dernier élément de la première liste par l'adresse contenue dans le **pointeur tete** de seconde liste.