Algorithmique Avancée et Programmation en C
Listes chainées

Rym Guibadj

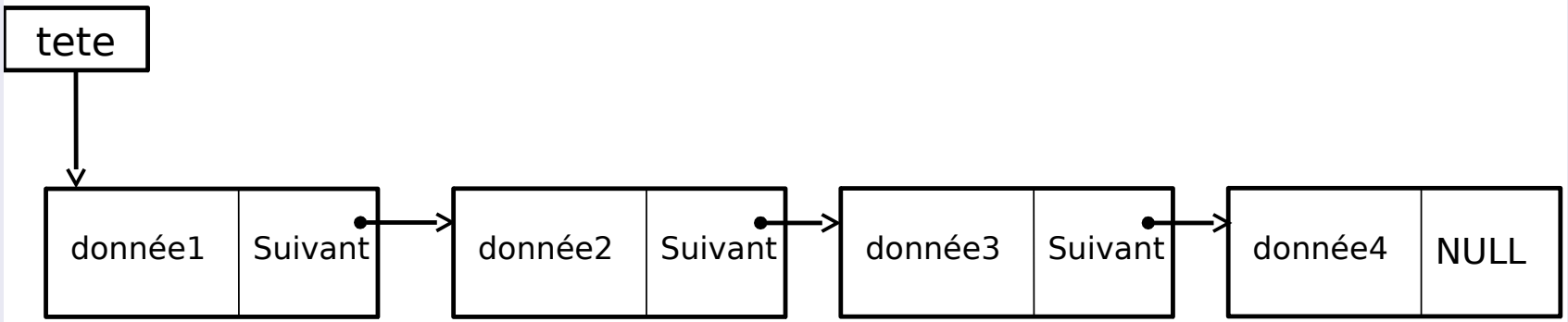
LISIC, EILCO

Listes chaînées

définition

Une liste est un ensemble d'objets de même type qui sont chaînés entre eux.

Représentation graphique



Principe

Chaque élément est une structure contenant les données de l'objet et un pointeur vers une structure de même type (l'élément suivant de la liste).

Propriétés

- Une liste chaînée est une structure linéaire qui n'a pas de dimension fixée à sa création.
- Ses éléments de même type sont éparpillés dans la mémoire et reliés entre eux par des pointeurs.
- Sa dimension peut être modifiée selon la place disponible en mémoire.
- La liste est accessible uniquement par sa tête de liste c'est-à-dire son premier élément.
- La fin de la liste est caractérisée par un pointeur NULL. Le dernier élément existant voit donc son champ ?suivant ? vaut NULL.

Liste ou Tableau

Structure	Dimension	Position d'une information	Accès à une information
Tableau	Fixe	Par son indice	Directement par l'indice
Liste chaînée	Evolue selon les actions	Par son adresse	Séquentiellement par le pointeur de chaque élément

Quand est qu'il faut utiliser une liste chaînée ?

Lorsque l'on doit traiter des objets représentés par des suites sur lesquelles on doit effectuer de nombreuses suppressions et de nombreux ajouts. Les manipulations sont alors plus rapides qu'avec des tableaux.

les types de listes chaînées

- **Les listes simplement chaînées** : chaque éléments dispose d'un pointeur vers l'élément suivant de la liste.
- **Les listes doublement chaînées** : chaque élément dispose de deux pointeurs pointant respectivement vers l'élément précédent et l'élément suivant.
- **Les listes circulaires** : le dernier élément pointe sur le premier élément de la liste.

Remarque

Ces différents types peuvent être mixés selon les besoins.

Opérations sur les listes chaînées

Primitives nécessaires

- Insertion
- Suppression
- Recherche
- Modification
- Affichage

Liste linéaire simple

Déclaration en C

```
typedef struct element
{
    Type1 champ1 ;
    Type2 champ2 ;
    Type3 champ3 ;
    .......
    struct element * suivant ;
}Element ;


Element * liste = NULL ;
```

Liste linéaire simple

Exemple de déclaration d'une liste d'entiers en C

```
typedef struct element
{
    int valeur ;
    struct element * suivant ;
}Element ;


Element * liste = NULL ;
```

Liste linéaire simple

Algorithm InsertionEnTete (liste, pElement)

arguments
liste : pointeur vers la tête de la liste
pElement : pointeur vers l'élément à insérer
begin
if (liste == NULL) **then** */* liste vide*/*
liste = pElement
else */*liste non vide*/*
pElement->suivant = liste */* p pointe vers le début de la liste*/*
liste = pElement */*p devient le nouveau pointeur du début de liste*/*
end if
return liste
end

Liste linéaire simple

Algorithm InsertionListeTriee (liste, pElement)

```
arguments
  liste : pointeur vers la tête de la liste
  pElement : pointeur vers l'élément à insérer
  begin
    if (liste == NULL) then /* liste vide*/
      liste = pElement
    else /* liste non vide*/
      if (pElement->valeur < liste->valeur) then /* insertion en début de liste*/
        liste = InsertionEnTete(liste, pElement)
      else
        p = liste
        pp = NULL
        /* parcourir la liste pour trouver la position d'insertion*/
        while ((p != NULL) and (pElement->valeur > p->valeur)) do
          pp=p
          p = p->suivant
        end while
        /* effectuer le chainage */
       pp->suivant = pElement
       pElement->suivant = p
      end if
    end if
    return liste
  end
```

Liste linéaire simple

Algorithm Affichage (liste)

arguments
liste : pointeur vers la tête de la liste
begin
p = liste */* p est un pointeur sui sert à parcourir la liste*/*
while (p != NULL) **do**
afficher(p->valeur)
pElement = pElement->suivant
end while
end

Liste linéaire simple

Algorithm Suppression (liste, val)

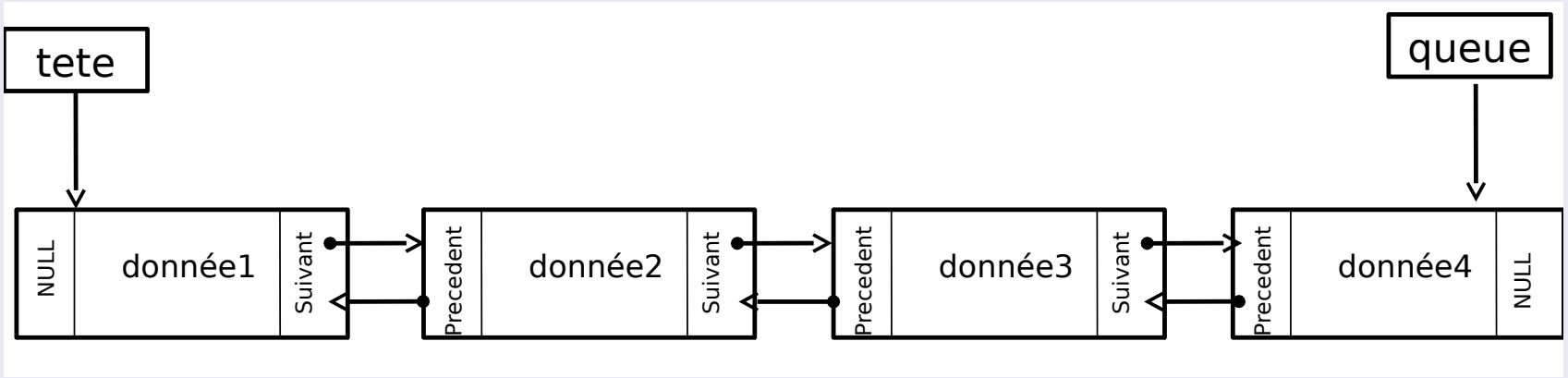
```
arguments
  liste : pointeur vers la tête de la liste
  val : la valeur de l'élément à supprimer
  begin
     if (liste == NULL) return liste /* liste vide*/
     if (val == liste->valeur) then /* suppression du premier élément*/
          p = liste
          liste = liste->suivant
          desallouer(p)
      else
          p = liste
          pp = NULL
          /* parcourir la liste pour trouver la valeur à supprimer*/
          while ((p != NULL) and (val != p->valeur)) do
            pp=p
            p = p->suivant
          end while
          /* supprimer l'élément pointé par p*/
          if (p != NULL) then
            pp->suivant = p->suivant
            desallouer(p)
     end if
   return liste
end
```

Liste linéaire doublement chainée

définition

Chaque élément dispose de deux pointeurs pointant respectivement vers l'élément précédent et l'élément suivant.

Représentation graphique



Liste linéaire doublement chainée

Déclaration en C

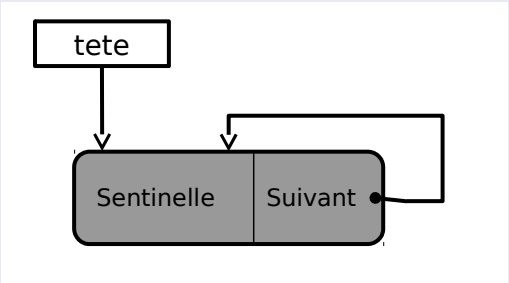
```
typedef struct element
{
    Type1 champ1 ;
    Type2 champ2 ;
    Type3 champ3 ;
    .......
    struct element * suivant ;
    struct element * precedent ;
}Element ;


typedef struct listeDouble
{
    Element * tete ;
    Element * queue ;
}ListeDouble ;
```

Liste circulaire avec sentinelle

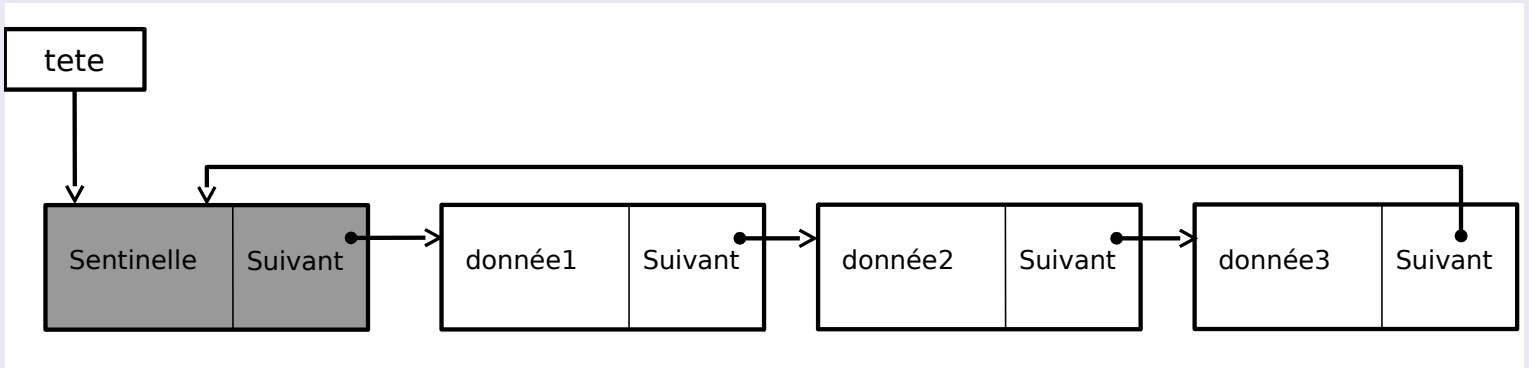
définition

Le dernier élément pointe sur le premier élément de la liste.

Liste circulaire vide



Liste circulaire avec 3 éléments



Liste circulaire avec sentinelle

Idée

- Ajout d'une cellule sentinelle en tête de liste
- liste->suivant : pointe sur la première cellule de la liste,
- le champ donnée du premier élement n'est pas utilisé

Avantages

- Simplifie la gestion des cas limites : cas où la liste est vide (la liste contient toujours au moins une cellule)
- Cela évite les problèmes de tests avec la valeur NULL