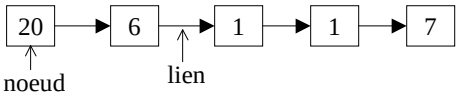
Les listes chaînées

Les listes chaînées

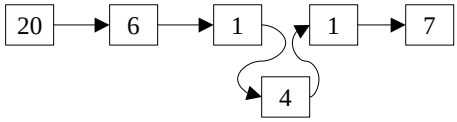
Les tableaux :
- ont une taille fixe ;
- occupent un espace contiguë.

	20	6	1	...	1	7
Indices :	0	1	2	...	n-1	n

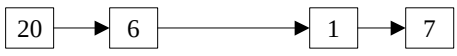
Une liste chaînée est un ensemble d'éléments organisés séquentiellement



Ajouter un élément :



Supprimer un élément :

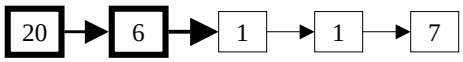


Inconvénient des listes chaînées

Avec un tableau : accès direct à un élément en connaissant son indice.

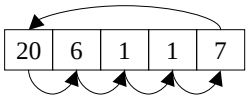
indice : 2

Avec une liste chaînée : parcourir la liste pour accéder à un élément.

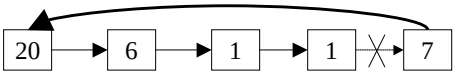


Avantages des listes chaînées

Avec un tableau : déplacer un élément ⇒ décalage.

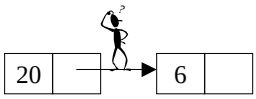


Avec une liste chaînée : déplacer un élément ⇒ modifier ses liens.



- Ajout d'un élément ;
- Suppression d'un élément.

Comment représenter les liens ?



Stocker un lien dans une case mémoire particulière : un pointeur

Un pointeur est une variable qui contient l'adresse (l'endroit où est rangé en mémoire) d'une autre variable :

20	suivant = 100

6	

100 : adresse

Algorithme
Début
    Variable suivant: **pointeur de** ...
Fin

Comment représenter les liens et les nœuds ?

- Pour former une liste chaînée, tous nœuds doit avoir un lien !

20	100

6	

100

- Regrouper nœud et lien dans une structure d'enregistrement :

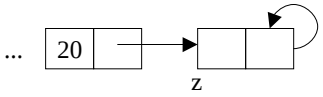
nœud.h

```
#ifned __NŒUD_H
#define __NŒUD_H

    typedef struct n{
        int info ;
        struct n* suivant ;
    } NŒUD ;

#endif    /* __NŒUD_H */
```

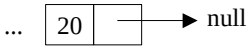
N'importe quel type de variables

Comment représenter les liens et les nœuds ?

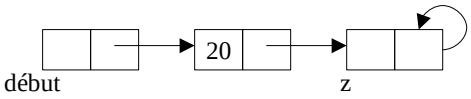
- Le dernier nœud doit avoir un lien !
  - Utiliser un nœud factice qui pointe sur lui même :



  - Utiliser un pointeur nul :

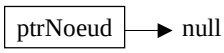


- Pour mémoriser le début de la liste, on utilise parfois un nœud factice :



Initialiser une liste chaînée

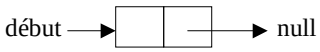
Initialiser une liste chaînée




```
NŒUD* initialiser(){
      return NULL ;
}

void main(){
      NŒUD* ptrNoeud ;
      ptrNoeud = initialiser() ;
}
```

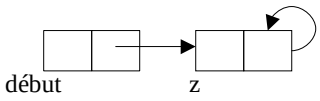

Initialiser une liste chaînée avec un nœud factice au début

```
NŒUD* initialiser(){
      NŒUD* ptrNoeud ;
      ptrNoeud = (NŒUD*)malloc( sizeof( NŒUD ) ) ;
      if( ptrNoeud != NULL ){
            ptrNoeud->suivant = NULL ;
      }
      return ptrNoeud ;
}

void main(){
      NŒUD* debut ;
      debut = initialiser() ;
}
```



Initialiser une liste chaînée avec un nœud factice au début et à la fin

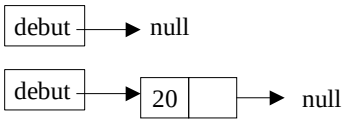
```
NŒUD* initialiser(){
    NŒUD* z, *debut ;
    z = (NŒUD*)malloc( sizeof( NŒUD ) ) ;
    if( z != NULL ){
        z->suivant = z ;
        debut = (NŒUD*)malloc( sizeof( NŒUD ) ) ;
        if( debut != NULL ){
            debut->suivant = z ;
        }
    }
    return debut ;
}

void main(){
    NŒUD* debut ;
    debut = initialiser() ;
}
```



Insérer dans une liste chaînée

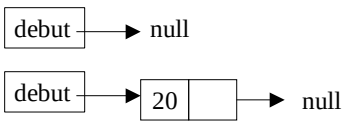
Insérer dans une liste chaînée avec un argument de type pointeur

```
NŒUD* insererEnTete( NŒUD* debut, int i ){
    NŒUD* nouveau ;
    nouveau = (NŒUD*)malloc( sizeof( NŒUD ) ) ;
    if( nouveau != NULL ){
        nouveau->suivant = debut ;
        nouveau->info = i ;
    }
    return nouveau ;
}

void main(){
    NŒUD* debut ;
    debut = initialiser() ;
    debut = insererEnTete( debut, 20 ) ;
}
```



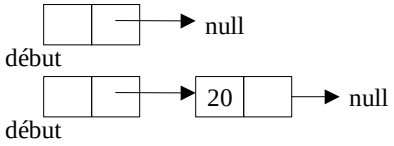
Insérer dans une liste chaînée avec un argument de type pointeur de pointeur

```
void main(){
    NŒUD* debut ;
    int res ;
    debut = initialiser() ;
    res = insererEnTete( &debut, 20 ) ;
}

    int insererEnTete( NŒUD** debut, int i ){
        NŒUD* nouveau ;
        nouveau = (NŒUD*)malloc( sizeof( NŒUD ) ) ;
        if( nouveau != NULL ){
            nouveau->suivant = *debut ;
            nouveau->info = i ;
            *debut = nouveau ;
            return 1 ;
        } else {
            return 0 ;
        }
    }
```



Insérer dans une liste chaînée avec un nœud factice au début

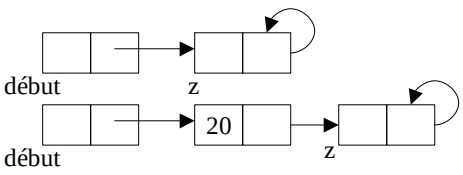
```
int insererEnTete( NŒUD* debut, int i ){
      NŒUD* nouveau ;
      nouveau = (NŒUD*)malloc( sizeof( NŒUD ) ) ;
      if( nouveau != NULL ){
            nouveau->suivant = debut->suivant ;
            nouveau->info = i ;
            debut->suivant = nouveau ;
            return 1 ;
      } else {
            return 0 ;
      }
}
void main(){
      NŒUD* debut ;
      int res ;
      debut = initialiser() ;
      res = insererEnTete( debut, 20 ) ;
}
```



Insérer dans une liste chaînée avec un nœud factice au début et à la fin

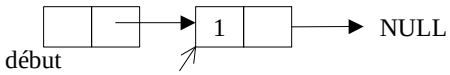
• La fonction précédente est utilisée puisqu'elle ne dépend que du premier nœud :

```
int insererEnTete( NŒUD* debut, int i ){
      NŒUD* nouveau ;
      nouveau = (NŒUD*)malloc( sizeof( NŒUD ) ) ;
      if( nouveau != NULL ){
            nouveau->suivant = debut->suivant ;
            nouveau->info = i ;
            debut->suivant = nouveau ;
            return 1 ;
      } else {
            return 0 ;
      }
}
```



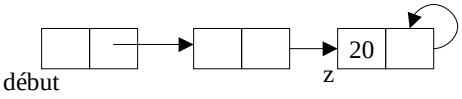
Rechercher dans une liste chaînée

Rechercher un élément




```
Nœud* recherchePrecedent( Nœud* debut, int i ){
    while( debut!=NULL && debut->info!=i ){        /*tant que info du suivant ≠ i*/
        debut = debut ->suivant ;                  /*continuer la recherche*/
    }
    return debut ;
}
```

Rechercher un élément dans un liste ayant un nœud factice à la fin



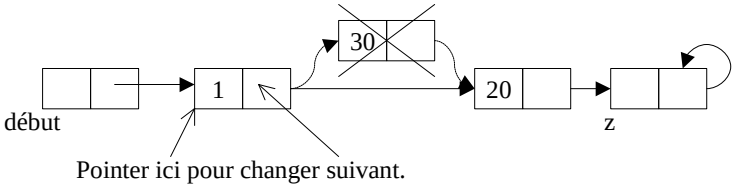

```
Nœud* rechercheePrecedent( Nœud* debut, int i ){
    while( debut->info != i ){                    /*tant que info du suivant ≠ i*/
        debut = debut ->suivant ;                 /*continuer la recherche*/
    }
    return debut ;
}
```

- Ce n'est plus nécessaire de tester le fin de la liste.

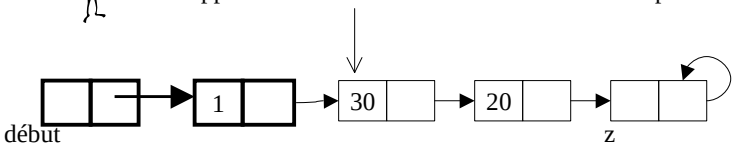
Supprimer dans une liste chaînée

Supprimer dans une liste chaînée

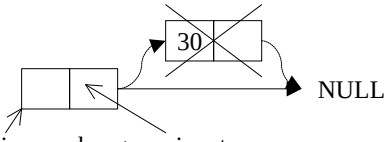
- Supprimer un élément (30 par exemple) :



Pour supprimer un élément ⇒ il faut s'arrêter sur le précédent :

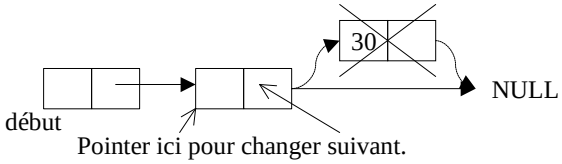


Rechercher l'élément précédant celui à supprimer sans nœud factice




```
Nœud* rechercheAvant( Nœud* debut, int i ){
    if( debut !=NULL ){                 /* liste non vide */
        if( debut->info == i ){         /* si le premier est celui recherché */
            return debut ;
        }
        while(debut->suivant!=NULL && debut ->suivant->info != i ){
                debut = debut ->suivant ;
        }
    }
    return debut ;
}
```

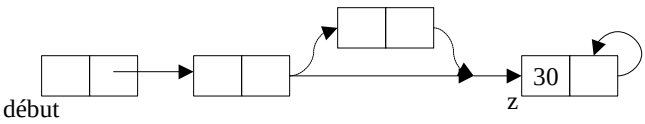
Rechercher l'élément précédant celui à supprimer avec un nœud factice au début




```
Nœud* recherchePrecedent( Nœud* debut, int i ){
    while( debut->suivant!=NULL && debut->suivant->info != i ){
        debut = debut->suivant ;
    }
    return debut ;
}
```

• Ce n'est plus nécessaire de tester le début de la liste.

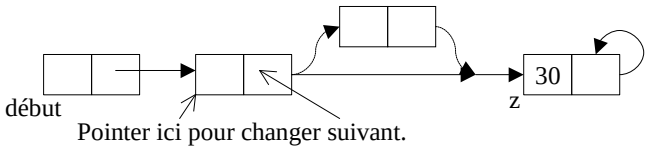
Rechercher l'élément précédant celui à supprimer avec un nœud factice au début et à la fin




```
Nœud* recherchePrecedent( Nœud* debut, int i ){
    while( debut->suivant->info != i ){
        debut = debut->suivant ;
    }
    return debut ;
}
```

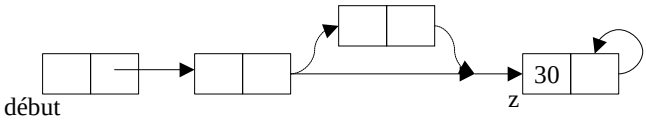
• Ce n'est plus nécessaire de tester ni le début, ni la fin de la liste.

Supprimer dans une liste chaînée




```
void supprimerSuivant( NŒUD* n ){
    Nœud* tmp ;
    tmp = n->suivant ;                      /* mémorise le suivant … */
    n->suivant = n->suivant->suivant ;      /* pour le changer ici … */
    free( tmp ) ;                           /* et le détruire là */
}
```


Rechercher et supprimer dans une liste chaîné ayant un nœud factice au début et à la fin




```
void supprimer( Nœud* debut, int i ){
    Nœud* ptrPreced ;
    ptrPreced = recherchePrécédent( debut, i ) ;
    if( ptrPreced!=NULL && ptrPreced->suivant!=ptrPreced->suivant->suivant ){
        supprimerSuivant( ptrPreced ) ;
    }
}
```