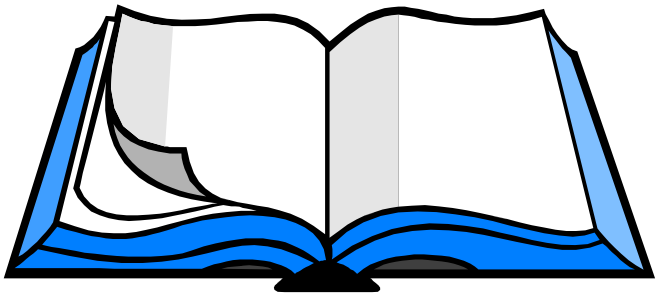
IUT de Lannion **Ludovic Liétard**
Département Informatique
Semestre 3

R3.02 Développement efficace

Support de cours
(Les Listes Linéaires)



1. La structure de Liste Linéaire

1.1 Définition

1.2 Représentation dynamique d'une liste linéaire

1.1 Définition

Une liste est une structure chaînée.

C'est la structure la plus générale. Il n'y a aucune politique particulière pour les insertions ou les retraits.

Chaque élément est référencé par sa valeur ou sa place dans la liste.

Il est possible :

- de créer une liste
- d'insérer un élément en tête
- d'insérer un élément après le $k^{ème}$
- d'accéder au $k^{ème}$ élément de la liste
- de déterminer si la liste est vide
- de déterminer l'existence d'un élément
- de retirer un élément
- de retirer le $k^{ème}$ élément

Cette liste d'opérations n'est pas exhaustive.

On peut donc définir les opérations suivantes (à implémenter par des procédures ou fonctions) :

Créer :
Crée une liste vide

Vide :
Retourne vrai si la liste est vide

Taille :
Délivre le nombre d'éléments dans la liste

Insérer_en_Tete :
Insère une valeur en début de chaînage

Retirer :
Retire la tête de liste

Insérer_Après :
Insère une valeur en position (k+1)

Acceder :
Délivre l'élément en position k.

Retirer_pos :
Retire de la liste l'élément en position k. Cet élement est également retourné.

Existe :
Retourne vrai si un élément est dans la liste.

1.2 Représentation dynamique d'une liste

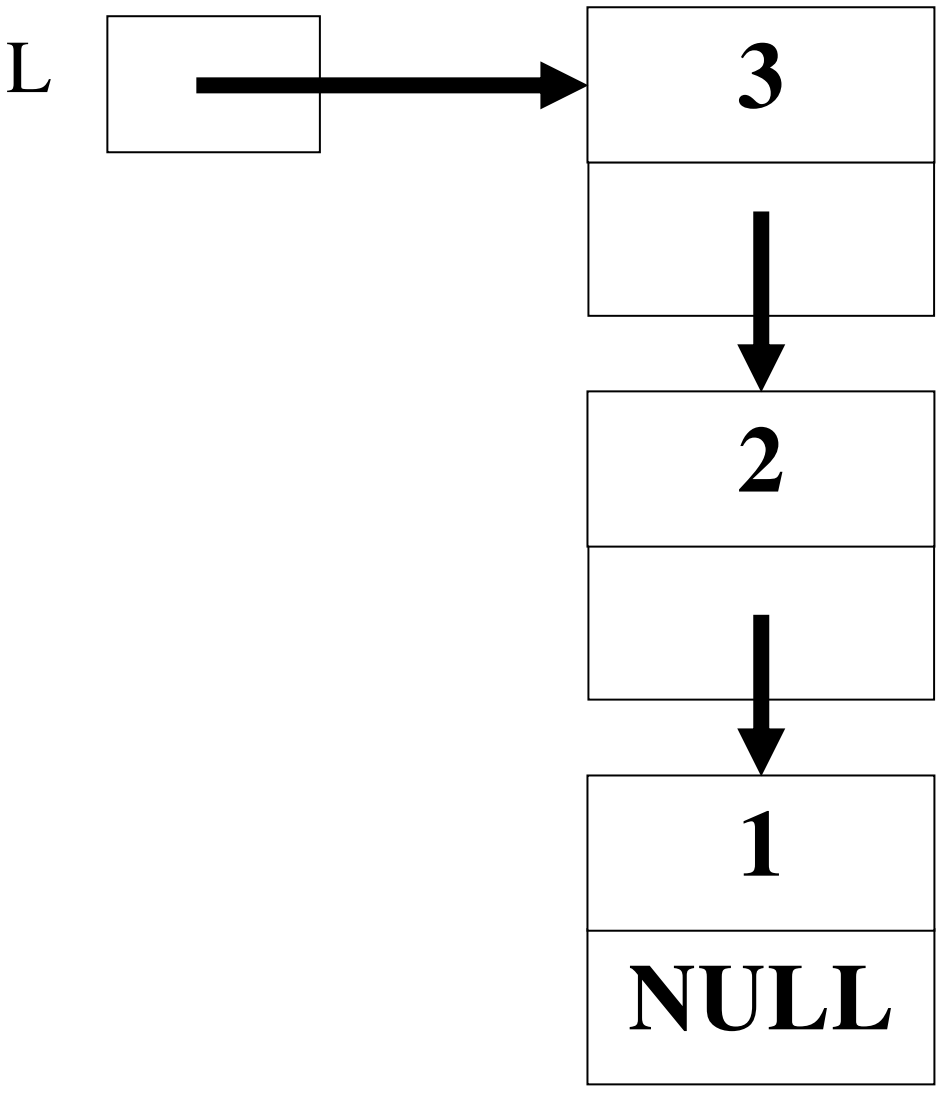
La déclaration d'une liste d'entiers est :

```
typedef struct Elem{
    int val ;
    struct Elem * svt;
} element;

typedef element* Liste;
```

La liste pointe sur la tête.

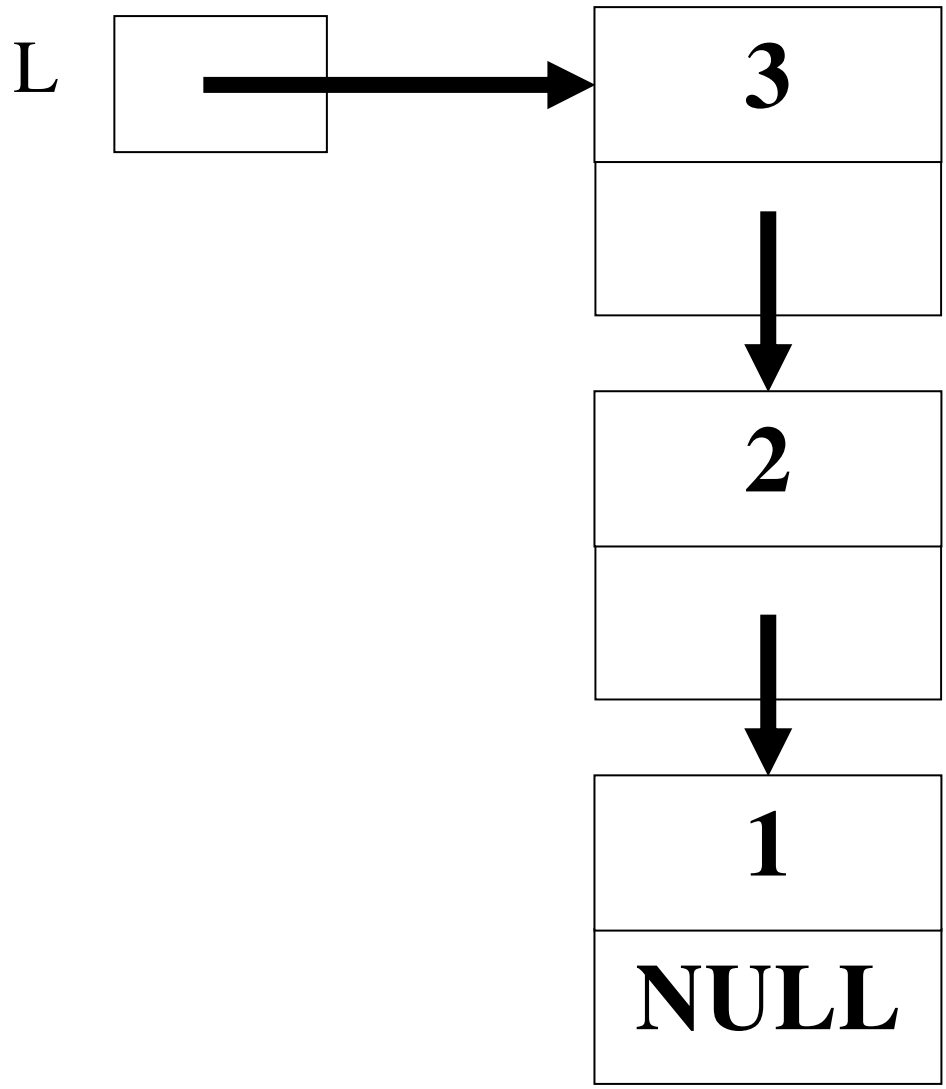
Liste vide

Liste non vide (avec 3 éléments):



Un principe essentiel :

Il ne faut pas perdre la tête de liste !!



Que se passe-t-il si L est modifié ?

Tout parcours de la liste implique un pointeur auxiliaire.

```
void initialisation(Liste *L){
  *L=NULL;
}

int vide(Liste L){
  if (L==NULL){
    return 1;
  }else{
    return 0;
  }
}
```

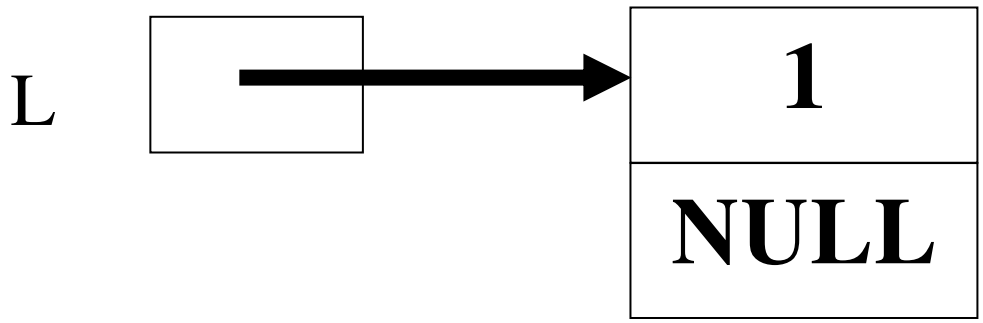
```
int taille(Liste L){
  element *P;
  int cpt;

  cpt=0;
  P=L;
  while (P!=NULL){
    cpt++;
    P=P->svt;
  }
  return cpt;
}
```

Pour *insérer en tête*, deux cas sont à distinguer :

Cas de la liste vide :

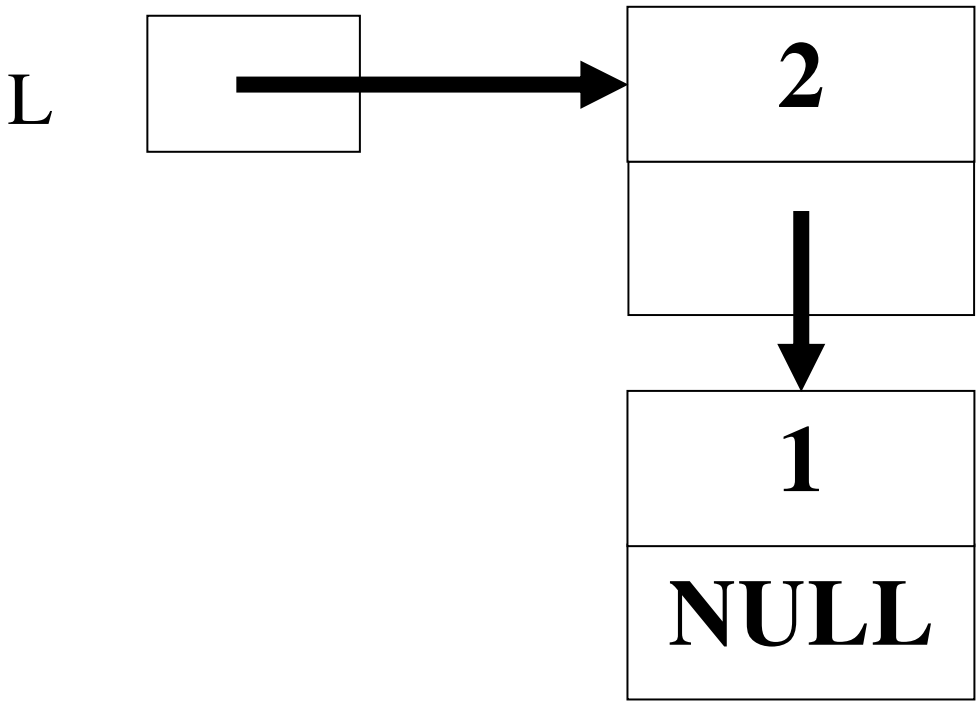
Avant insertion en tête :

Après insertion en tête :

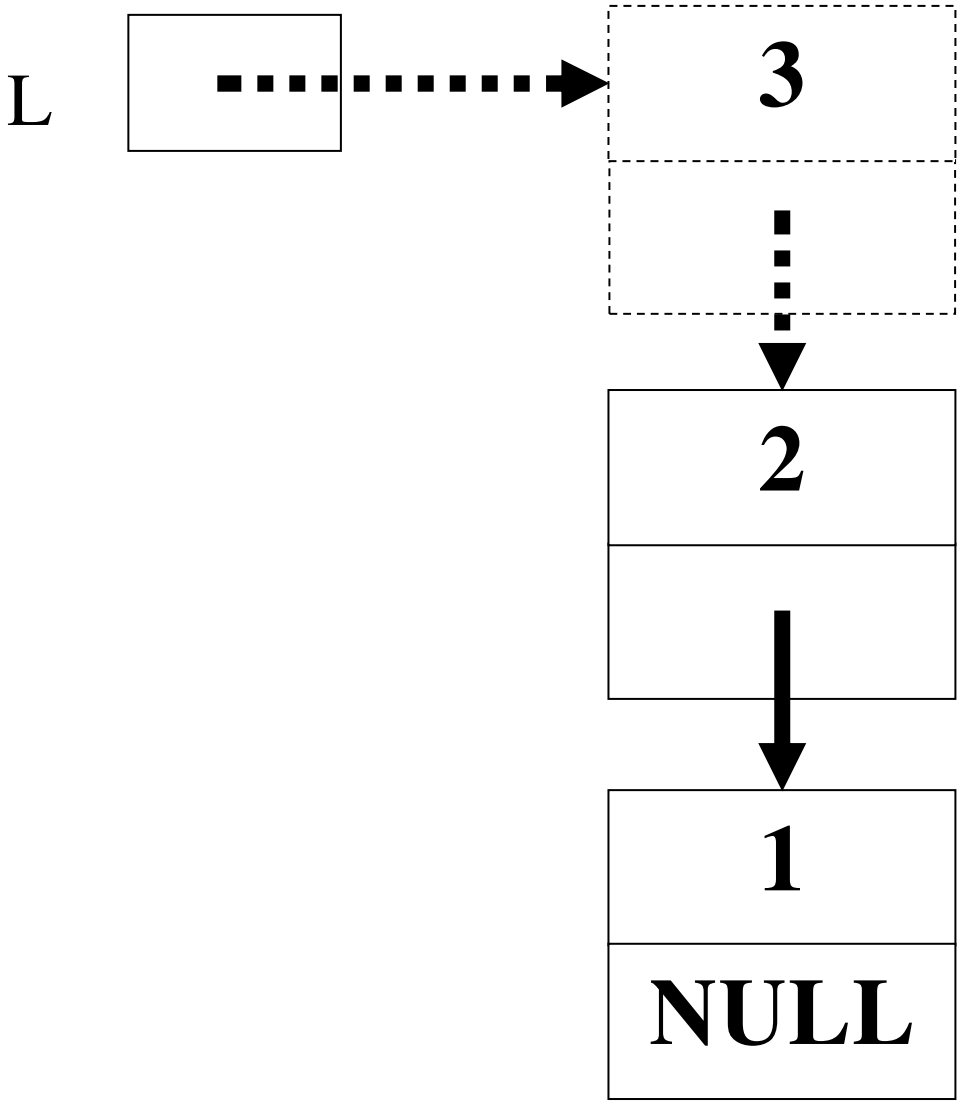


Cas de la liste non vide :

Avant insertion en tête:



Après insertion en tête :

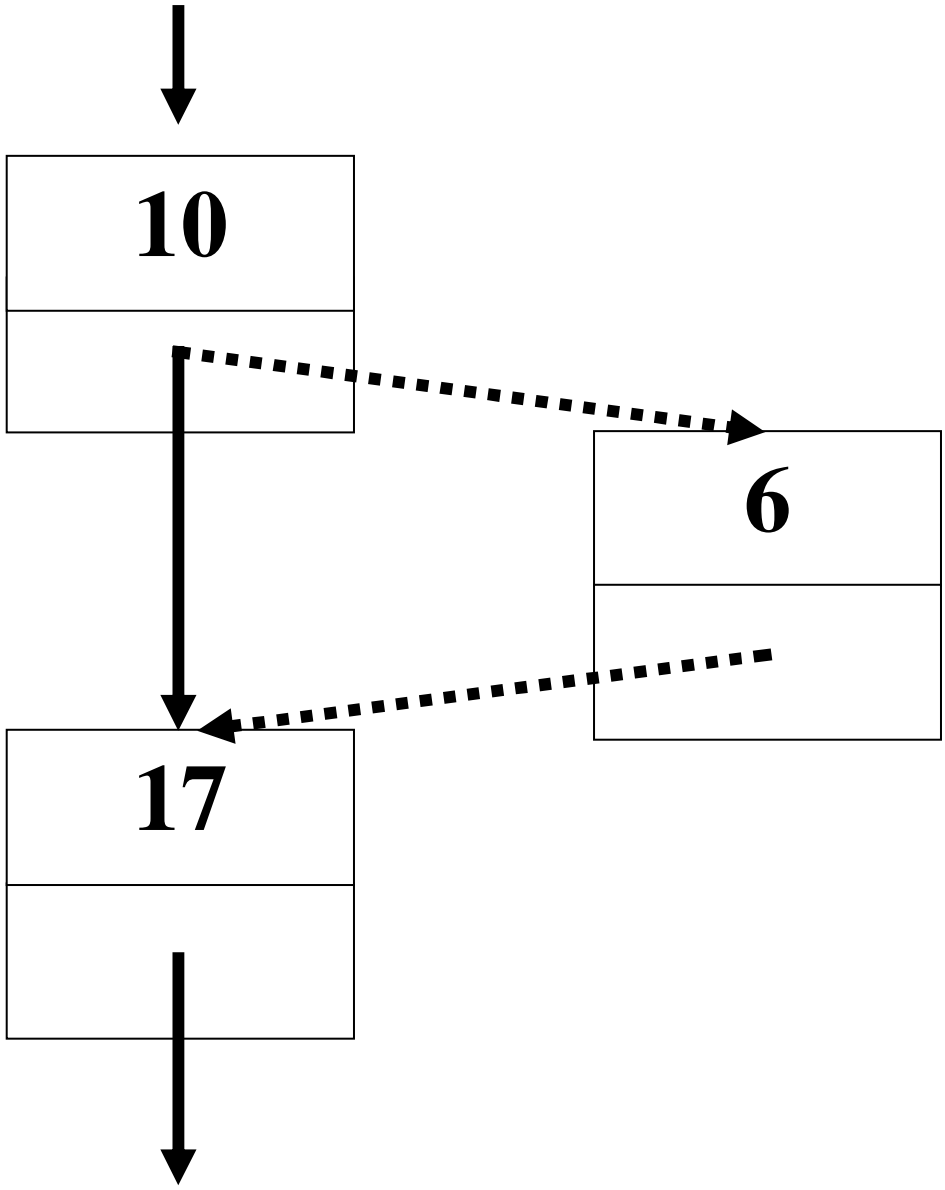


```
void insereEnTete(Liste *L,int valeur){
  element *p;

   p=(element*)
          malloc(sizeof(element));
  p->val=valeur;
  p->svt=NULL;

  if (*L==NULL){
    *L=p;
  }else{
    p->svt=(*L);
    *L=p;
  }
}
```

```
int obtenirTete(Liste L){
  return L->val;
}
```

Pour *insérer après*, on doit parcourir la liste pour trouver le $k^{ème}$ élément.

1) s'il est présent on insère après :



```
void Inserer_Apres
                    (Liste *L, int  k, int N){
   int I;
   element *courant;
   element *P;

    I=1;
   courant=*L;
   while((I!=k) && (courant!=NULL)){
              courant=courant->svt;
              I++;
   }
  if (courant==NULL) {
       printf("pas de place %d\n",k);
  }else{
P=(element*)  malloc(sizeof(element));
       P->val=N;
       P->svt=courant->svt;
      courant->svt=P;
  }
}
```

Accéder à l'élément en position k:

```
void acceder(Liste L,int k, int * n){
   int I;
   element *courant;

    I=1;
   courant=L;
   while((I!=k) && (courant!=NULL)){
              courant=courant->svt;
              I++;

   }
  if (courant==NULL) {
        printf("pas de place %d\n",k);
  }else{
    *n=courant->val;
  }
}
```

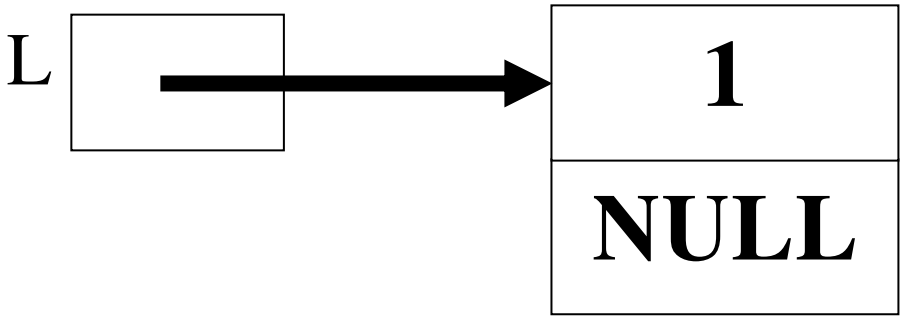
Accéder à l'élément en position k:

```
element * acceder(Liste L,int k){
   int I;
   element *courant;

    I=1;
   courant=L;
   while((I!=k) && (courant!=NULL)){
              courant=courant->svt;
              I++;

   }
   return courant;
}
```

Retirer la tête de liste :

Cas de la liste avec un seul élément :

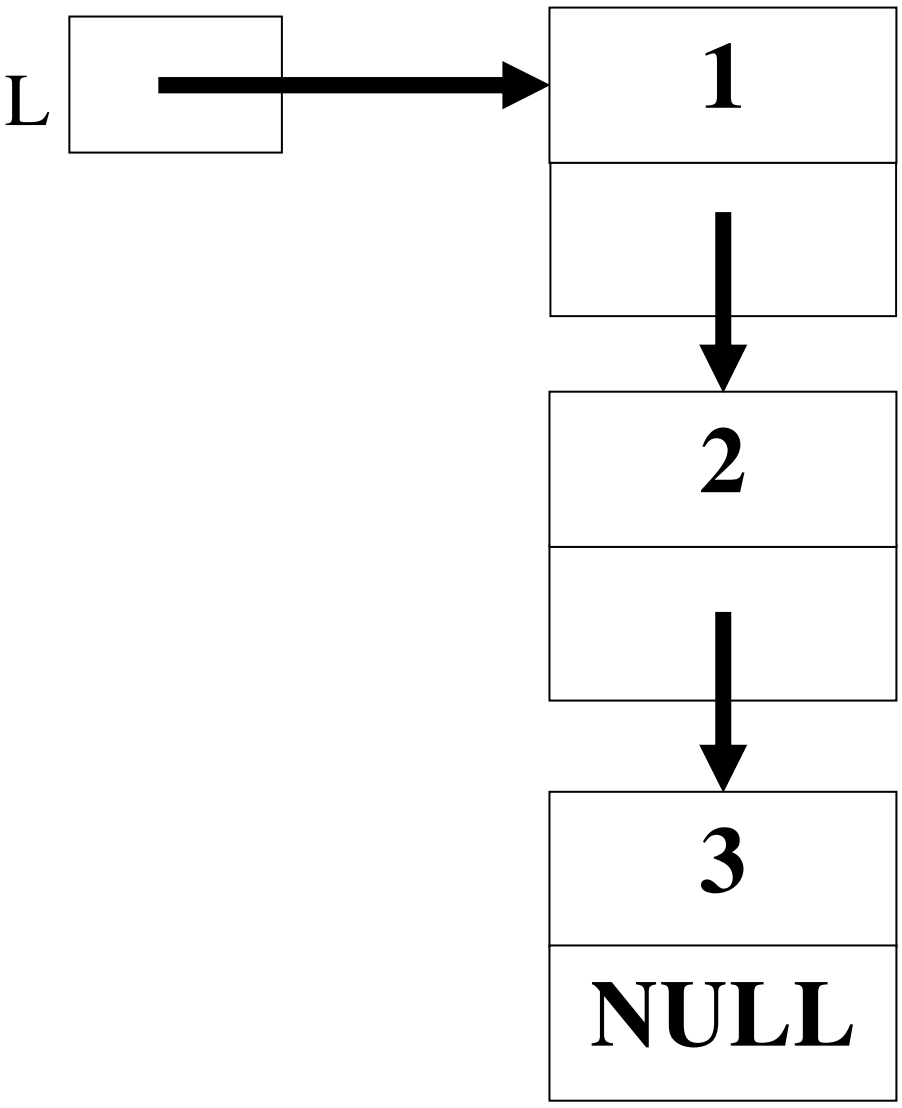
Avant suppression :



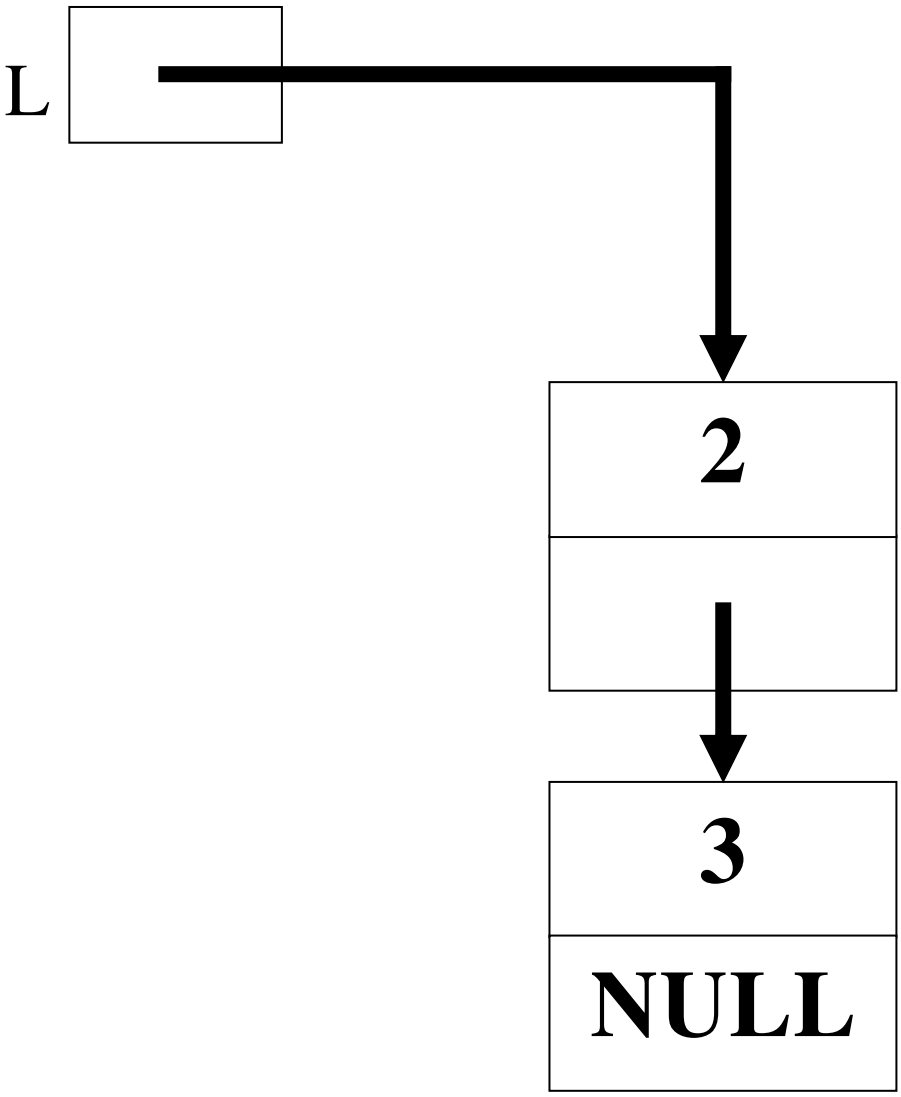
Après suppression :

Cas de la liste avec plusieurs éléments :

Avant suppression :



Après suppression :

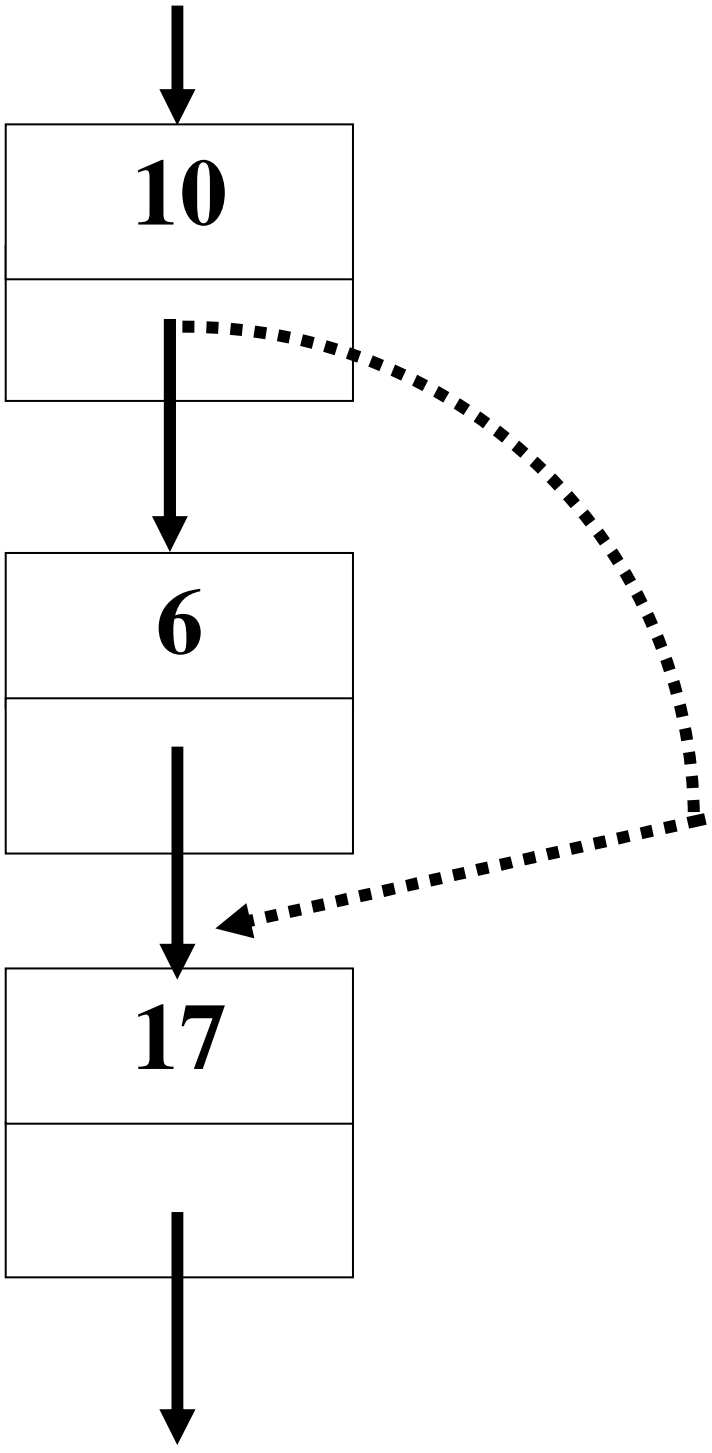


```
void retirer(Liste *L){
    if ((*L) != NULL) {
        (*L)=(*L)->svt;
    }
}
```

```
void retirer(Liste *L,int *V){
  if (*L!=NULL){
    *V=(*L)->val;
    (*L)=(*L)->svt;
  }
}
```

Pour *Retirer_pos*, on doit parcourir la liste pour trouver le $k^{ème}$ élément.

2) s'il est présent on le supprime (6 dans l'exemple) :



```
void retirerPos(Liste *L,int k){
  element * courant;
  element *prec;
   int I;

  courant=*L;
   prec=NULL;
   I=1;
  while ((courant!=NULL) && (I!=k)){
       prec=courant;
       courant=courant->svt;
       I++;
   }
  if (courant!=NULL){
       if (prec==NULL) {
            *L=(*L)->svt;
       }else{
         prec->svt=courant->svt;
       }
   }
}
```

```
int existe(Liste L,int k){
  element * courant;
  int res;

  courant=L;
   res=0;
   while ((courant!=NULL) &&
(res!=1)) {
       if ((courant->val)==k){
   res=1;
       }
       courant=courant->svt;
   }
   return res;
```