Partie 3 : Structures de données Linéaires

Structures de données linéaires

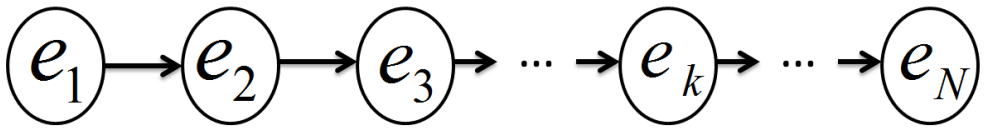
Listes

Terminologie

- Une liste est une séquence finie formée de 0 ou plusieurs éléments de même type.
- Exemples
  - (3, 7, 2, 8, 10, 4) est une liste d'entiers.
  - (lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche) est la liste des jours de la semaine.
- Chaque élément dans une liste est caractérisé par
  - son rang (sa place) dans cette liste
  - son contenu
- La longueur d'une liste est son nombre d'éléments.
- Une liste de longueur N dispose alors de N rangs (de 1 à N, par exemple)

Terminologie

- Parties d'une liste (non vide) $L = (e_1, e_2, ..., e_N)$:
  - L'élément e_1 est appelé la tête de la liste : c'est son premier élément.
  - $(e_2, ..., e_N)$ est appelée la queue de la liste (ou son reste). Il s'agit d'une liste.
  - L'élément $e_i + 1$ est le successeur de l'élément e_i (l'élément e_N n'a pas de successeur).
  - L'élément $e_i - 1$ est le prédécesseur de l'élément e_i (l'élément e_1 n'a pas de prédécesseur).



Opérations sur une Liste (1)

- liste_vide : $\rightarrow$ Liste
  - Opération d'initialisation ; la liste créée est vide
- longueur : Liste $\rightarrow$ Entier
  - Retourne le nombre d'éléments dans la liste
- insérer : Liste $\times$ Entier $\times$ Élément : $\rightarrow$ Liste
  - insérer(L,j,e) : liste obtenue à partir de L en remplaçant la place j par une place contenant e, sans modifier places précédentes et en décalant places suivantes
- supprimer : Liste $\times$ Entier : $\rightarrow$ Liste
  - supprimer(L,j) : liste obtenue à partir de L en supprimant la place j et son contenu, sans modifier places précédentes et en décalant places suivantes

Opérations sur une Liste (2)

- kème : Liste $\times$ Entier $\rightarrow$ Élément
  - Fournit l'élément de rang donné dans une liste
- accès : Liste $\times$ Entier $\rightarrow$ Place
  - Connaître la place de rang donné : accès(L,k) est la place de rang k dans la liste L
- contenu : Liste $\times$ Place $\rightarrow$ Élément
  - Connaître l'élément d'une place donnée. contenu(L,p) = e : dans la liste L, la place p contient l'élément e
- succ : Liste $\times$ Place $\rightarrow$ Place
  - Passer de place en place. succ(L,p) = p' : dans la liste L, la place qui succède à la place p est la place p'. Opération indéfinie si place en entrée est la dernière place de la liste

Type Abstrait Liste(1)

Type Liste
Utilise Élément, Booléen, Place
Opérations
liste_vide : $\rightarrow$ Liste
longueur : Liste $\rightarrow$ Entier
insérer : Liste $\times$ Entier $\times$ Élément $\rightarrow$ Liste
supprimer : Liste $\times$ Entier $\rightarrow$ Liste
kème : Liste $\times$ Entier $\rightarrow$ Élément
accès : Liste $\times$ Entier $\rightarrow$ Place
contenu : Liste $\times$ Place $\rightarrow$ Élément
succ : Liste $\times$ Place $\rightarrow$ Place
Préconditions
insérer(l,k,e) est-défini-ssi $1 \leq k \leq$ longueur(l)+1
supprimer(l,k) est-défini-ssi $1 \leq k \leq$ longueur(l)
kème(l,k) est-défini-ssi $1 \leq k \leq$ longueur(l)
accès(l,k) est-défini-ssi $1 \leq k \leq$ longueur(l)
succ(l,p) est-défini-ssi $p \neq$ accès(l,longueur(l))

Type Abstrait Liste (2)

Axiomes

Soit, e : Élément, l, l' : Liste, k, j : Entier

si l = liste_vide alors longueur(l) = 0

sinon si l = insérer(l',k,e) alors longueur(l)=longueur(l')+1

sinon soit l = supprimer(l',k)alors longueur(l)=longueur(l')-1

si $1 \leqslant j < k$ alors kème(insérer(l,k,e),j) = kème(l,j)

sinon si j = k alors kème(insérer(l,k,e),j) = e

sinon kème(insérer(l,k,e),j) = kème(l,j-1)

si $1 \leqslant j < k$ alors kème(supprimer(l,k),j) = kème(l,j)

sinon kème(supprimer(l,k),j) = kème(l,j+1)

succ(l,accès(l,k)) = accès(l,k+1)

contenu(l,accès(l,k)) = kème(l,k)

Type Abstrait Liste (3)

Axiomes

Soit, e : Élément, l, l' : Liste, k, j : Entier

si $1 \leqslant k < j <$ longueur(l) alors

contenu(l,accès(supprimer(l,j),k)) = contenu(l,accès(l,k))

si $1 \leqslant k = j <$ longueur(l) alors

contenu(l,accès(supprimer(l,j),k))= contenu(l,accès(l,k+1))

si $1 \leqslant j < k <$ 1+longueur(l) alors

contenu(l,accès(insérer(l,k,e),j)) = contenu(l,accès(l,j))

si $1 \leqslant k = j <$ 1+longueur(l) alors

contenu(l,accès(insérer(l,k,e),j))= e

si $1 \leqslant k < j <$ 1+longueur(l) alors

contenu(l,accès(insérer(l,k,e),j)) = contenu(l,accès(l,j-1))

Opérations Auxiliaires sur une Liste

- concaténer : liste $\times$ liste $\rightarrow$ Liste
  - Accroche deuxième liste en entrée à la fin de la première liste
- est_présent : Liste $\times$ Élément $\rightarrow$ Booléen
  - Teste si un élément figure dans une liste

Extension Type Abstrait Liste

Extension type Liste

Opérations

concaténer : liste $\times$ liste $\rightarrow$ Liste

est_présent : Liste $\times$ Élément $\rightarrow$ Booléen

Préconditions

$\emptyset$

Axiomes

Soit, e : Élément, l, l' : Liste, k, j : Entier

longueur(concaténer(l,l')) = longueur(l) + longueur(l')

si k $\leqslant$ longueur(l)

alors kème(concaténer(l,l'),k)= kème(l,k)

sinon kème(concaténer(l,l'),k)= kème(l',k-longueur(l))

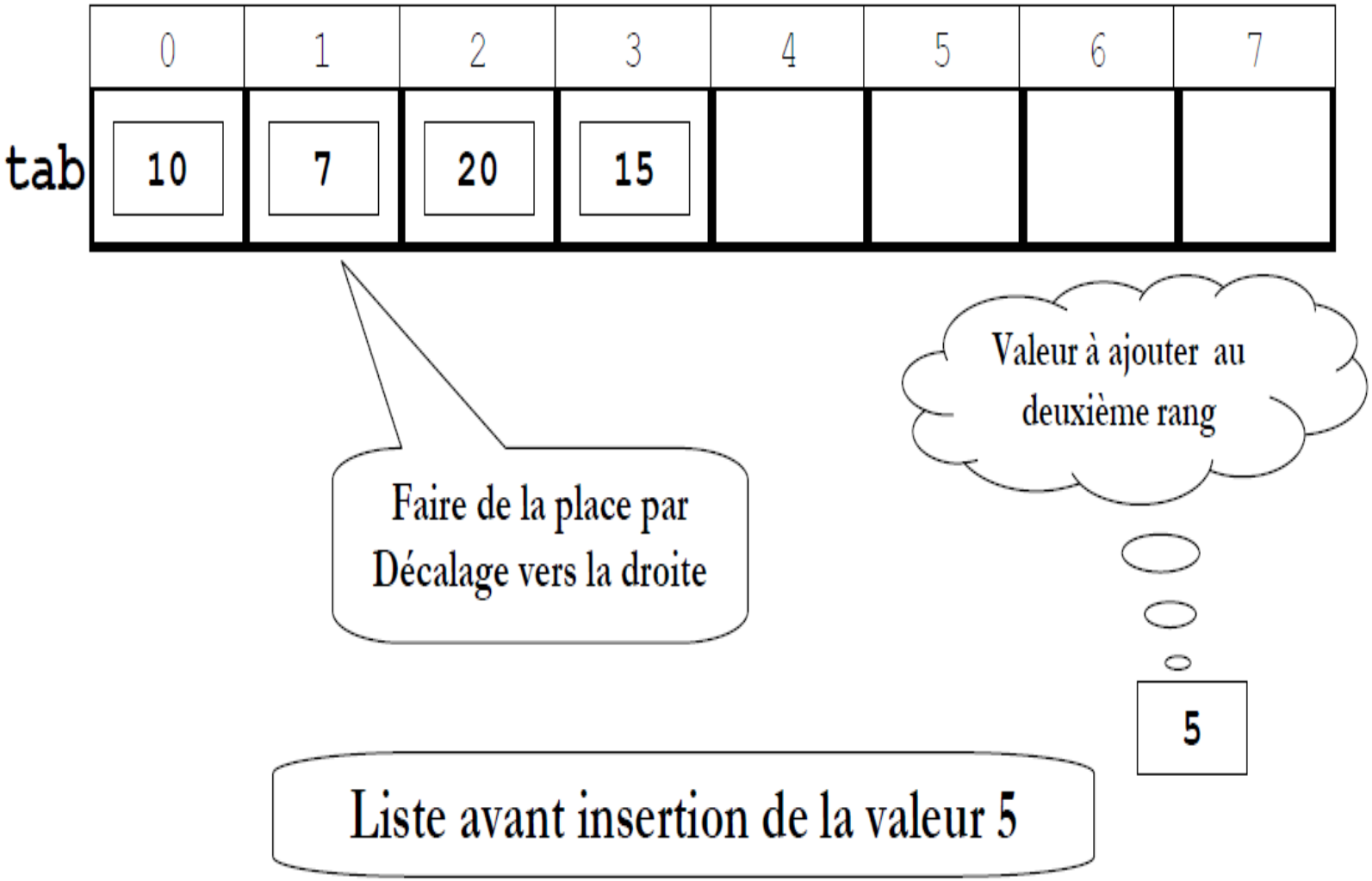
si longueur(l)=0 alors est_présent(l,e)=faux

sinon si kème(l,1)=e alors est_présent(l,e)=vrai

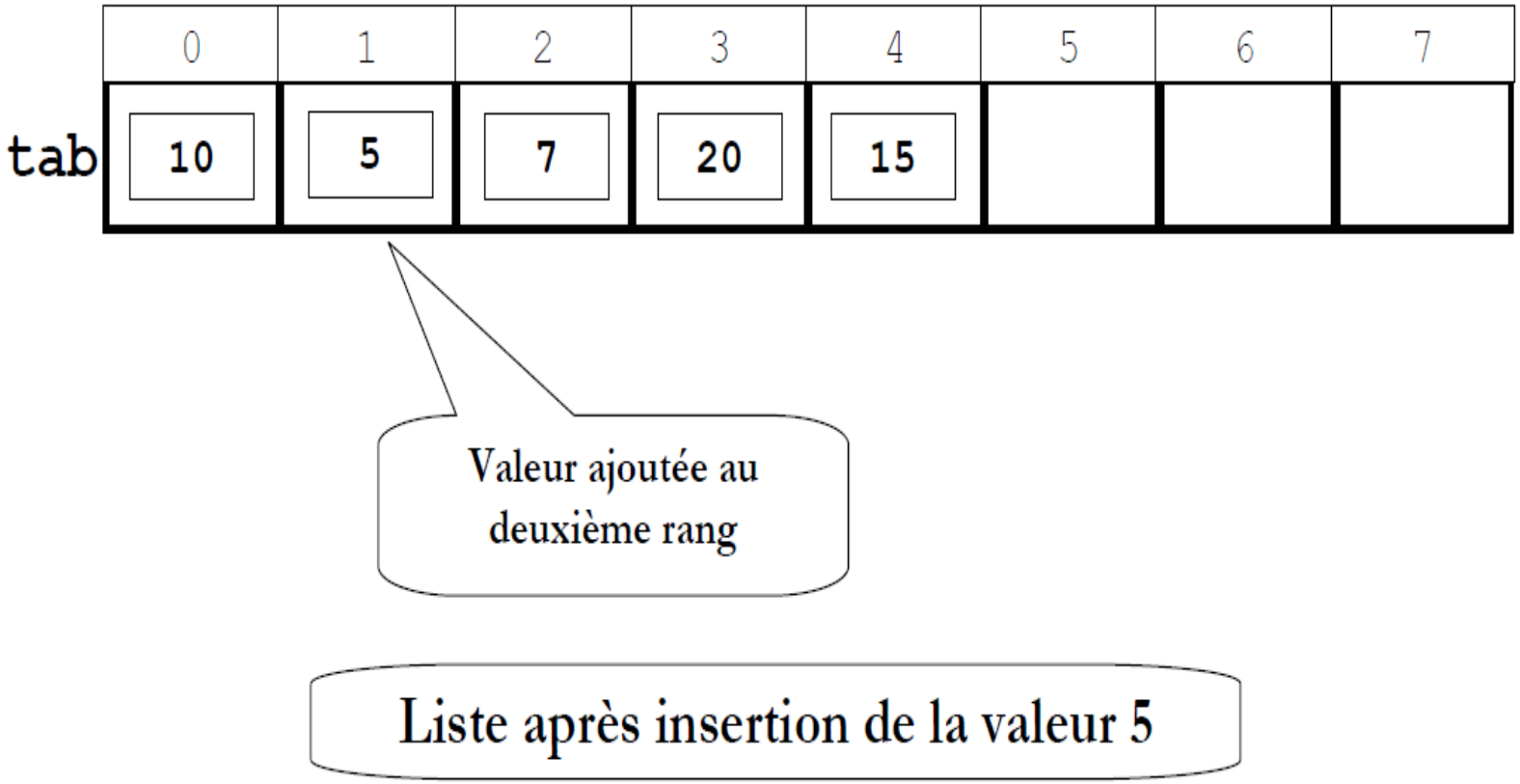
Représentation Contiguë d'une Liste

- Les éléments sont rangés les uns à côté des autres dans un tableau
  - La ième case du tableau contient le ième élément de la liste
  - Le rang est donc égal à la place ; ce sont des entiers
- La liste est représentée par une structure en langage C
  - Un tableau représente les éléments
  - Un entier représente le nombre d'éléments dans la liste
  - La longueur maximale, MAX_LISTE, de la liste doit être connue

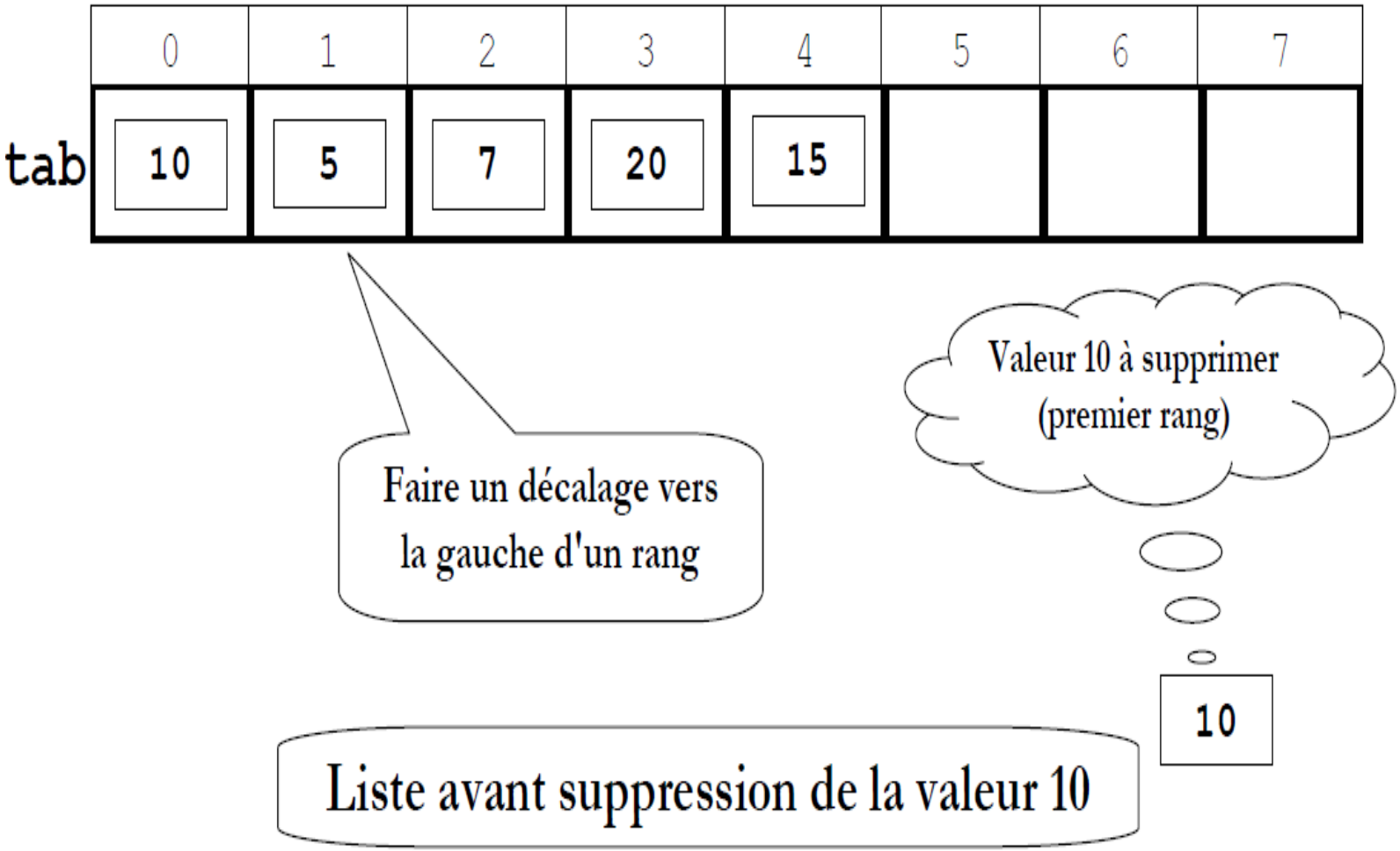
Ajout dans une liste contiguë



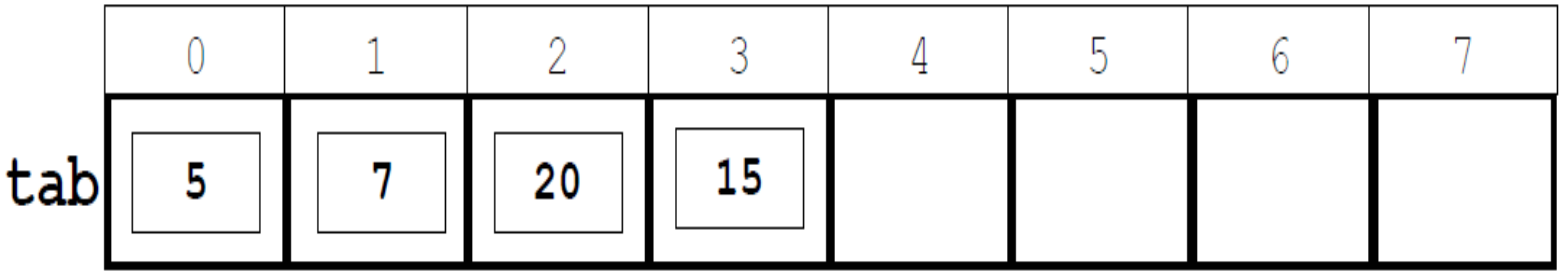
Ajout dans une liste contiguë



Suppression dans une liste contiguë



Suppression dans une liste contiguë



Liste après suppression de la valeur 10

Liste Contiguë (Contiguous List)

Liste Contiguë en C

```c
// taille maximale liste
#define MAX_LISTE 10

// type des éléments
typedef int Element;


// type Liste
typedef struct {
 Element tab[MAX_LISTE];
 int taille;
} Liste;
```

Spécification d'une Liste Contiguë

```
/* fichier "TListe.h" */
#ifndef _LISTE_TABLEAU
#define _LISTE_TABLEAU
#define MAX_LISTE 100 /* taille maximale de la liste */
typedef int element; /* les éléments sont des int */
typedef int Place; /* la place = le rang (un int) */
typedef struct {
 element tab[MAX_LISTE]; // les éléments de la liste
 int taille; // nombre d'éléments dans la liste
} Liste;
Liste liste_vide (void); // Déclaration des fonctions
int longueur (Liste l);
Liste inserer (Liste l, int i, element e);
Liste supprimer (Liste l, int i);
element keme (Liste l, int k);
#endif
```

Réalisation d'une Liste Contiguë (1)

```
Liste liste_vide(void) {
 Liste l;
 l.taille = 0;
 return l;
}

int longueur(Liste l) {
 return l.taille;
}

Liste supprimer(Liste l, int r) {
// précondition : 1<= r <= longueur(l)
int i;
if (r<1|| r>longueur(l)) { printf("Erreur : rang non valide !\n");
    exit(1);
 }
 for (i= r; i<longueur(l); i++) l.tab[i-1] = l.tab[i];
 (l.taille)--;
  return l;
}
```

Réalisation d'une Liste Contiguë (2)

```
Liste inserer(Liste l, int r, element e) {
// précondition : longueur(l) = MAX_LISTE
// et 1 <= r <= longueur(l)+1
 int i;
 if (longueur(l) == MAX_LISTE) {
   printf("Erreur : liste saturée !\n");
   exit(1);
 }
 if (r<1 || r>longueur(l)+1) {
   printf("Erreur : rang non valide !\n");
   exit(1);
 }
 for (i = longueur(l); i>=r; i--)
   l.tab[i] = l.tab[i-1];
 l.tab[r-1] = e;
 (l.taille)++;
 return l;
}
```

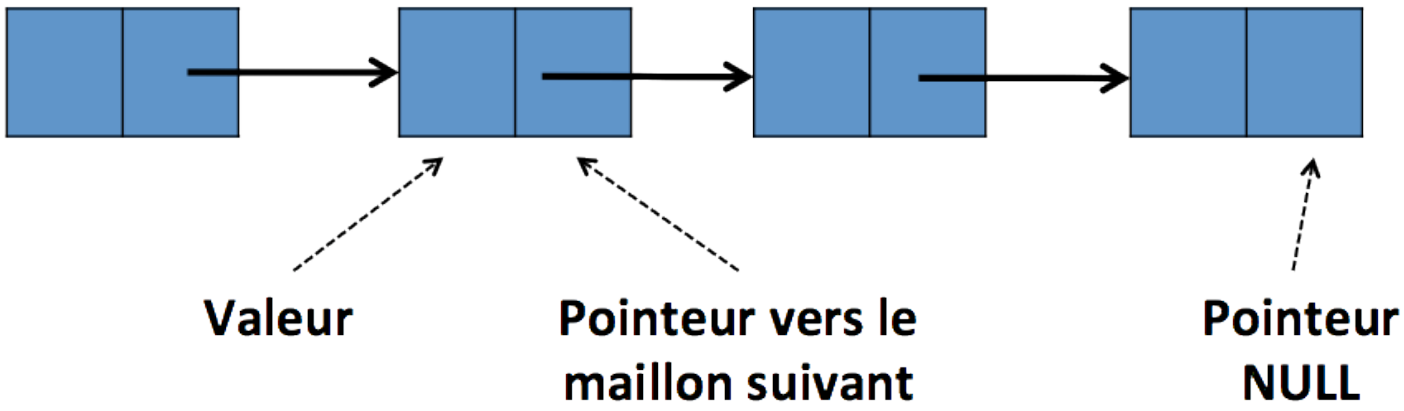
Réalisation d'une Liste Contiguë (3)

```
element keme(Liste l, int k) {
// pas de sens que si 1<= k <= longueur(l)
 if (k<1 || k>longueur(l)) {
   printf("Erreur : rang non valide !\n");
   exit(1);
 }
 return l.tab[k-1];
}
```

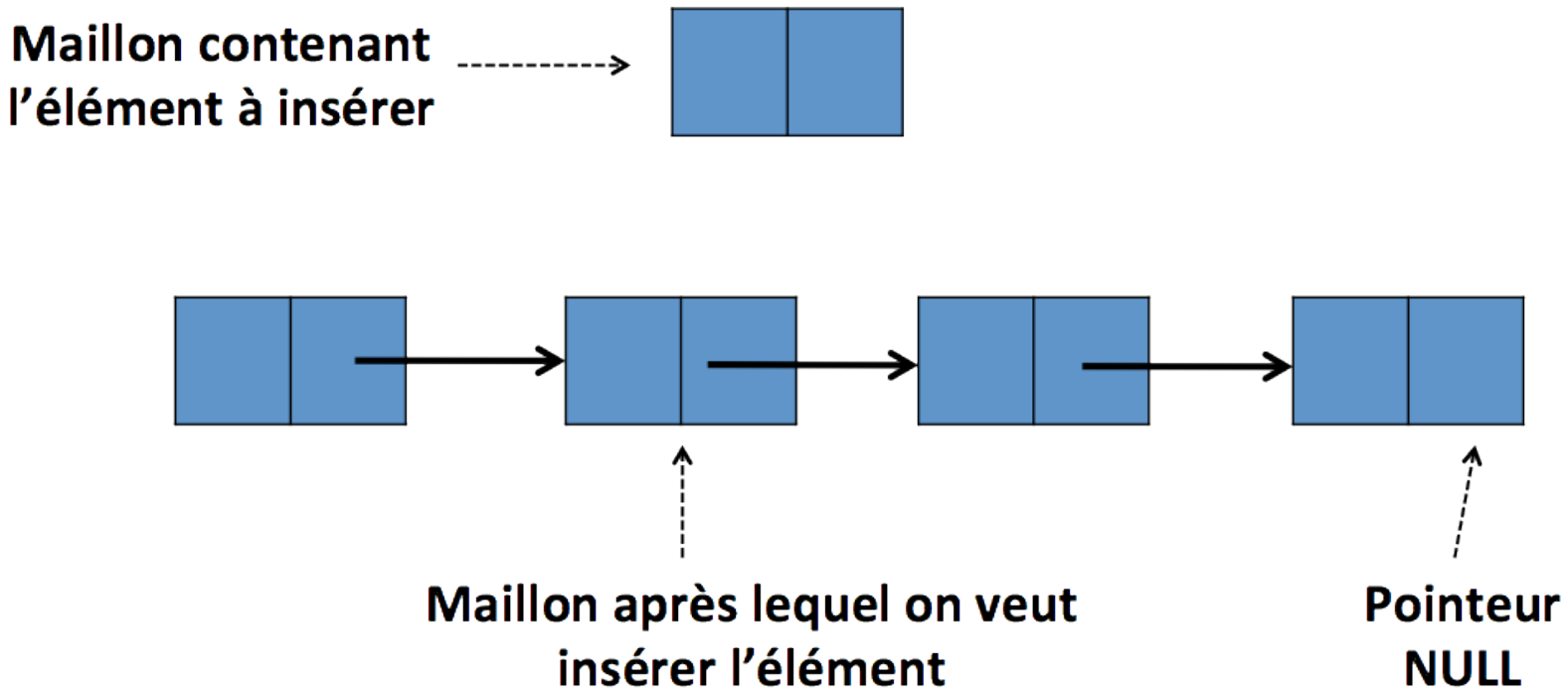
Représentation Chaînée d'une Liste

- Les éléments ne sont pas rangés les uns à côté des autres
  - La place d'un élément est l'adresse d'une structure qui contient l'élément ainsi que la place de l'élément suivant
  - Utilisation de pointeurs pour chaîner entre eux les éléments successifs
- La liste est représentée par un pointeur sur une structure en langage C
  - Une structure contient un élément de la liste et un pointeur sur l'élément suivant
  - La liste est déterminée par un pointeur sur son premier élément
  - La liste vide est représentée par la constante prédéfinie NULL

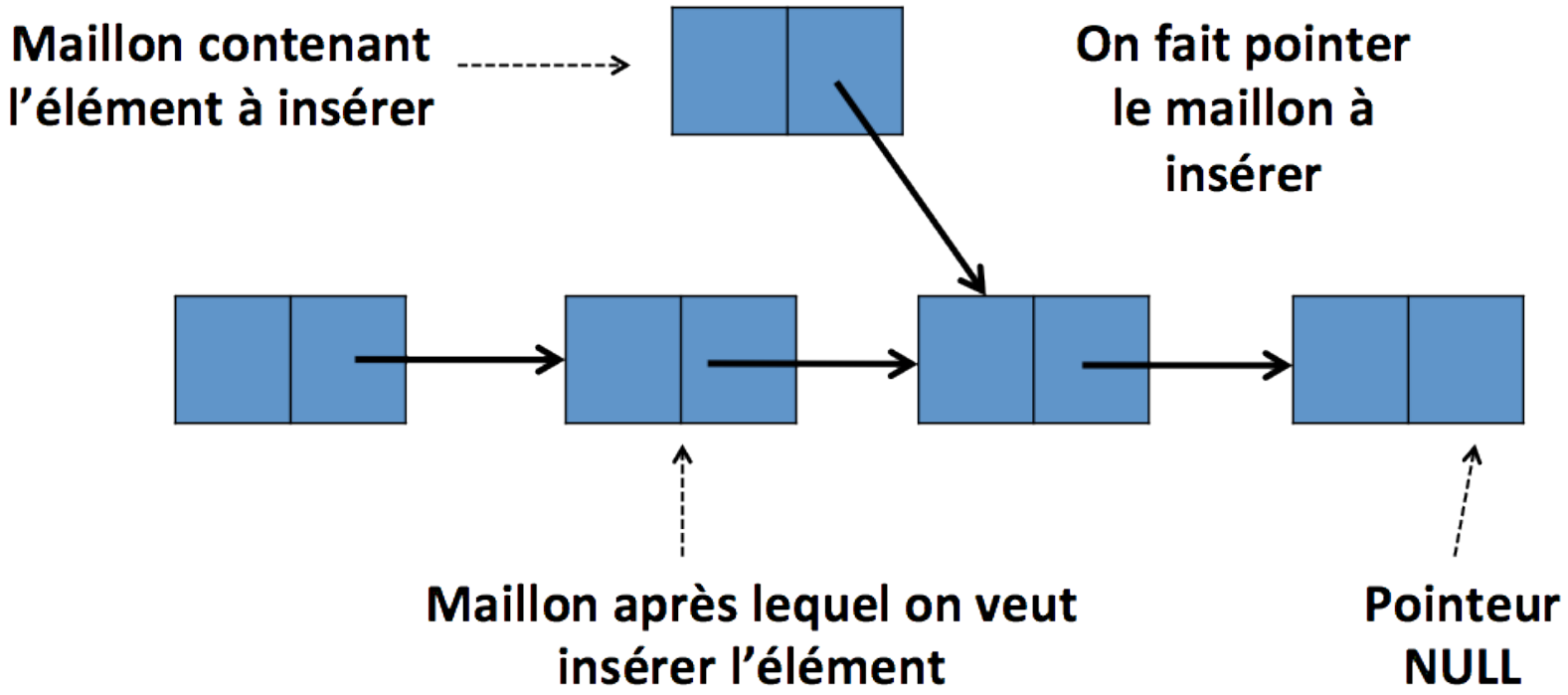
Représentation Chaînée d'une Liste



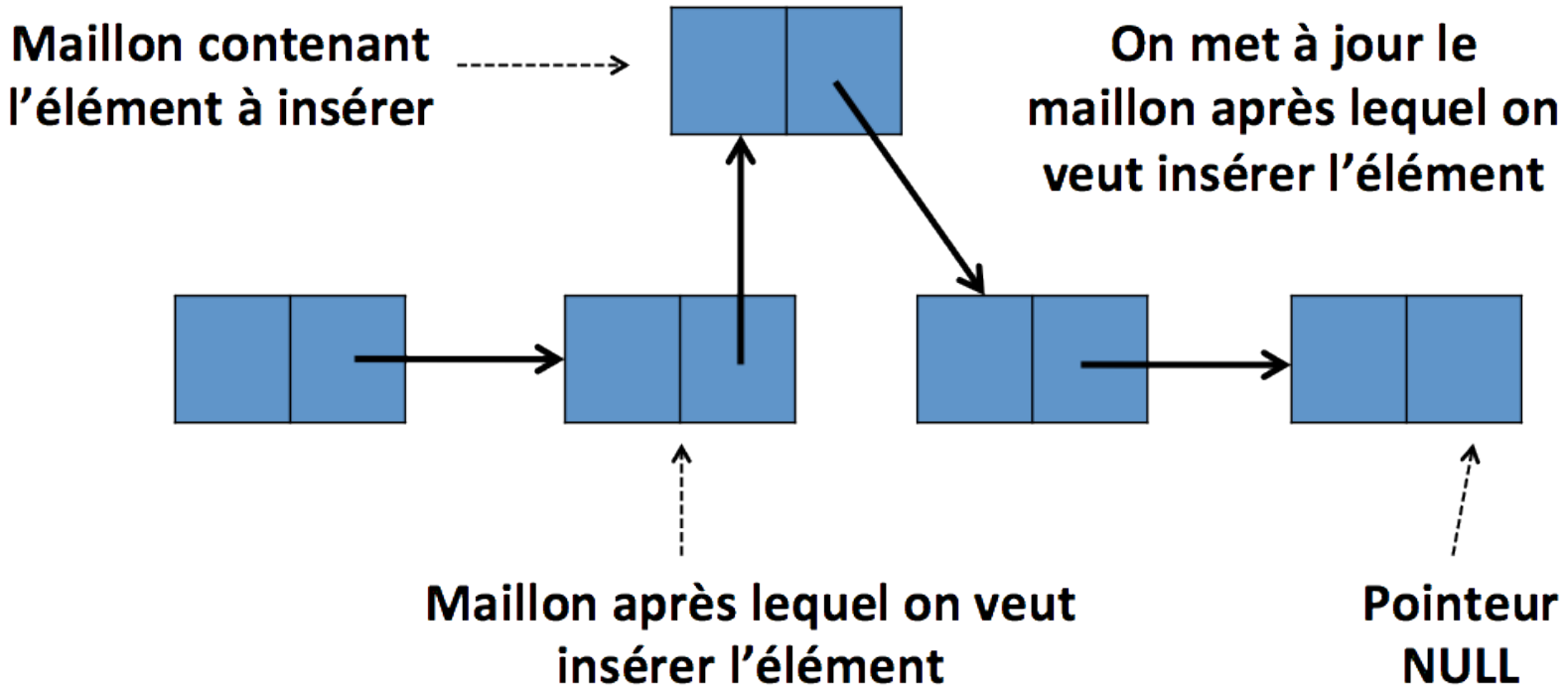
Ajout dans une liste chaînée



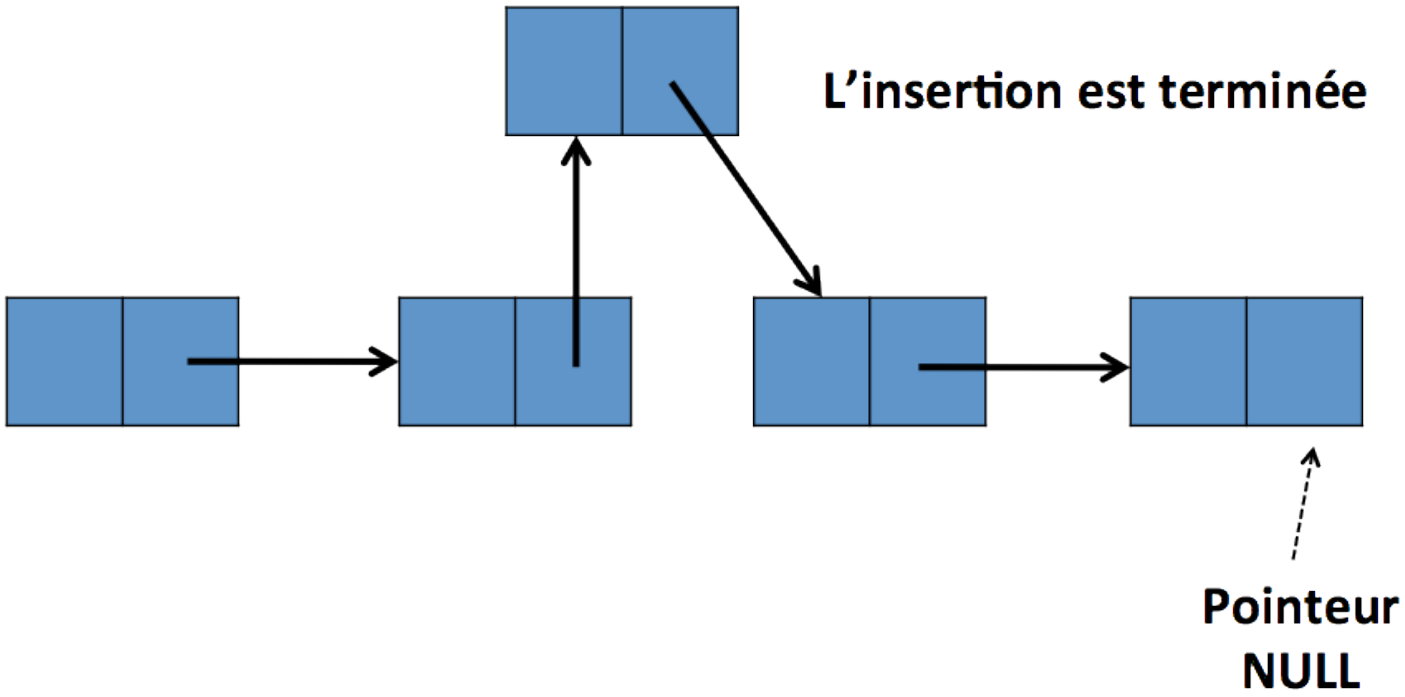
Ajout dans une liste chaînée



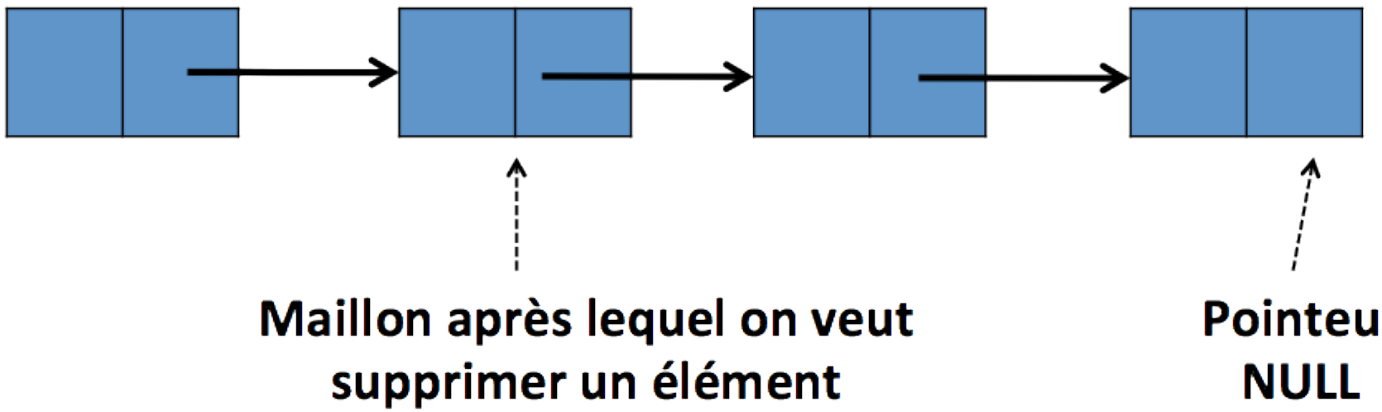
Ajout dans une liste chaînée



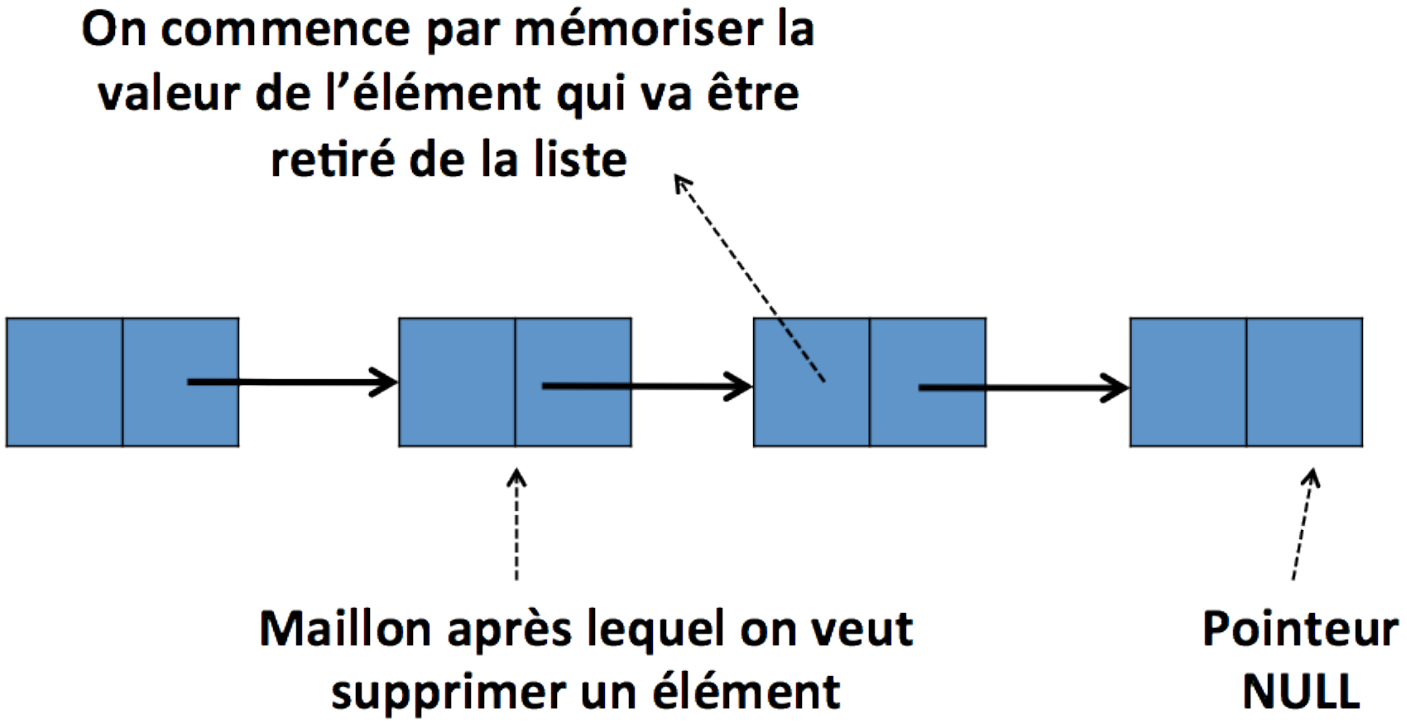
Ajout dans une liste chaînée



Suppression dans une liste chaînée

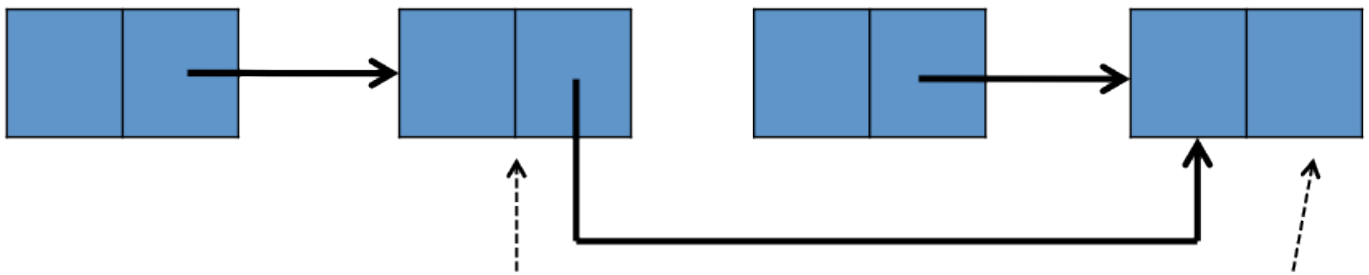


Suppression dans une liste chaînée



Suppression dans une liste chaînée

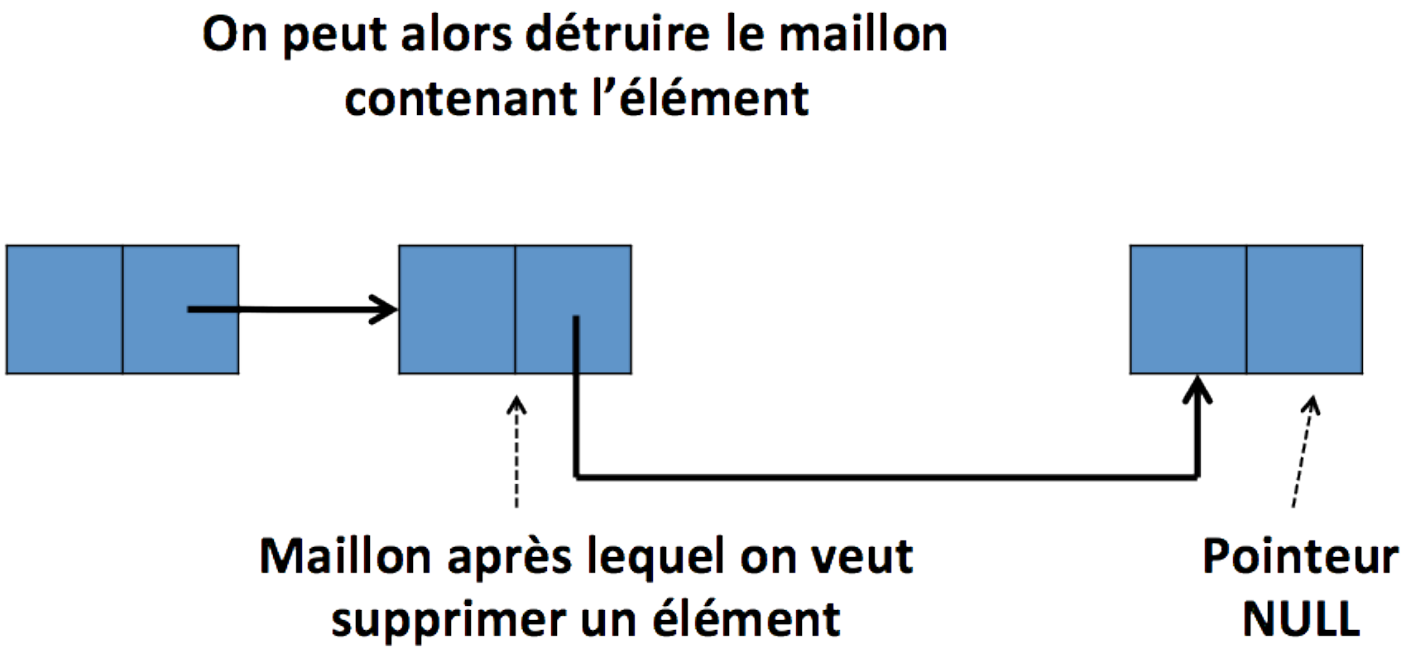
On fait pointer le maillon après lequel on veut supprimer l'élément vers son nouveau successeur



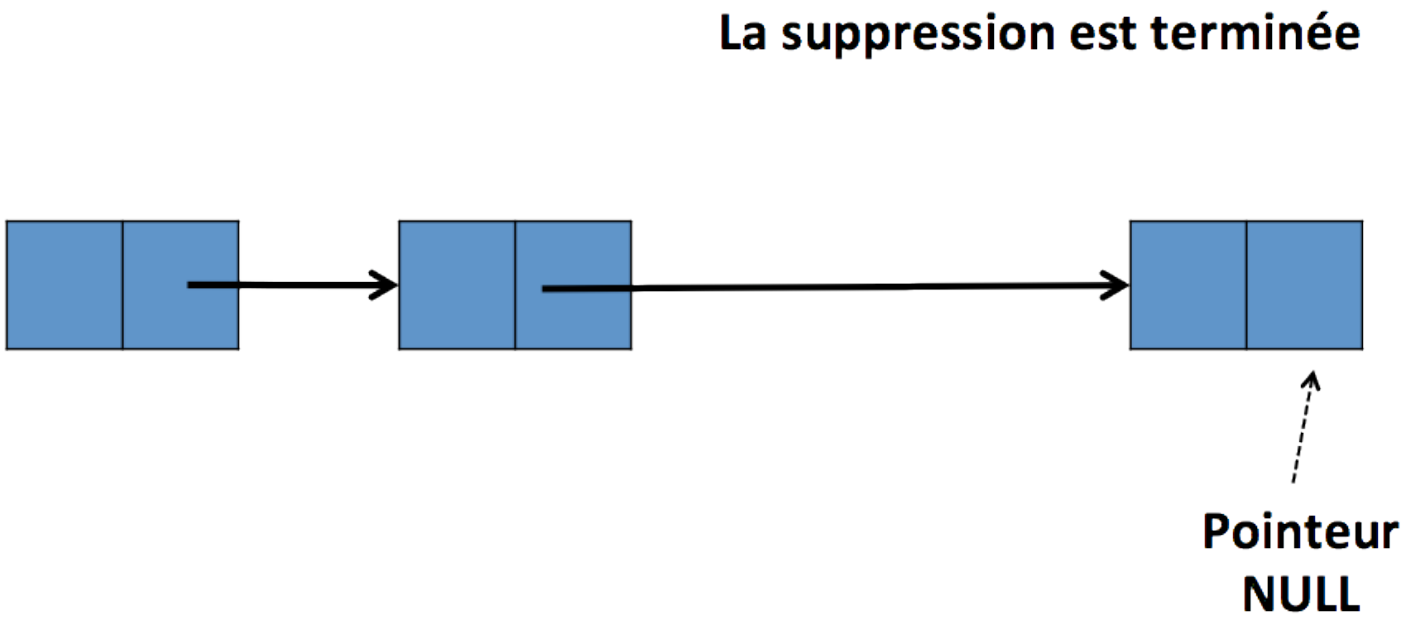
Maillon après lequel on veut supprimer un élément

Pointeur NULL

Suppression dans une liste chaînée



Suppression dans une liste chaînée



Liste Chaînée (Linked List)

/* Liste chaînée en C */

```
// type des éléments
typedef int element;

// type Place
typedef struct cellule* Place;

// type Cellule
typedef struct cellule {
  element valeur;
  struct cellule *suivant;
 } Cellule;

// type Liste
typedef Cellule *Liste;
```

Spécification d'une Liste Chaînée

```c
/* fichier "CListe.h" */
#ifndef _LISTE_CHAINEE
#define _LISTE_CHAINEE
typedef int element; /* les éléments sont des int */
typedef struct cellule *Place; /* la place = adresse cellule */
typedef struct cellule {
 element valeur; // un éléments de la liste
 struct cellule *suivant; // adresse cellule suivante
} Cellule;
typedef Cellule *Liste; // Définition du type liste
Liste liste_vide (void); // Déclaration des fonctions
int longueur (Liste l);
Liste inserer (Liste l, int i, element e);
Liste supprimer (Liste l, int i);
element keme (Liste l, int k);
Place acces (Liste l, int i);
element contenu (Liste l, Place i);
Place succ (Liste l, Place i);
#endif
```

Réalisation d'une Liste Chaînée (1)

```
Liste liste_vide(void) {return NULL;}

int longueur(Liste l) {
 int taille=0;
 Place p=l;
 while (p) {taille++; p=p->suivant;}
 return taille;
}

Place acces(Liste l, int k) {
// pas de sens que si 1 <= k <= longueur(l)
 int i;
 Place p;
 if (k<1 || k>longueur(l)) {
   printf("Erreur: rang invalide !\n"); exit(-1);}
 if (k == 1) return l;
 else {
   p=l;
   for(i=1; i<k; i++) p=p->suivant;
   return p;}}
```

Réalisation d'une Liste Chaînée (2)

```
element contenu(Liste l, Place p) {
// pas de sens si longueur(l)=0 (liste vide)
 if (longueur(l) == 0) {
   printf("Erreur: liste vide !\n"); exit(-1); }
 return p->valeur;}

element keme(Liste l, int k) {
// pas de sens que si 1 <= k <= longueur(l)
 if (k<1 || k>longueur(l)) {
   printf("Erreur : rang non valide !\n"); exit(-1);}
 return contenu(l, acces(l,k));
}

Place succ(Liste l, Place p) {
// pas de sens si p dernière place de liste
 if (p->suivant == NULL) {
   printf("Erreur: suivant dernière place!\n"); exit(-1);}
 return p->suivant;
}
```

Réalisation d'une Liste Chaînée (3)

```
Liste inserer(Liste l, int i, element e) {
// précondition :1 <= i <= longueur(l)+1
 if (i<1 || i>longueur(l)+1) {
  printf("Erreur : rang non valide !\n");
  exit(-1);}
 Place pc = (Place)malloc(sizeof(Cellule));
 pc->valeur=e; pc->suivant=NULL;
 if (i==1){ pc->suivant=l; l=pc;}
 else {
   int j;
   Place p=l;
   for (j=0; j<i-1; j++)
     p=p->suivant;
   pc->suivant=p->suivant;
   p->suivant=pc;}
 return l;
 }
```

Réalisation d'une Liste Chaînée (4)

```
Liste supprimer(Liste l, int i) {
// précondition : 1 <= i <= longueur(l)
 int j;
 Place p;
 if (i<1 || i>longueur(l)) {
   printf("Erreur: rang non valide!\n");
   exit(-1);
 }
 if (i == 1) {
   p=l;
   l=l->suivant;
 }
 else {
   Place q;
   q=acces(l,i-1);
   p=succ(l,q);
   q->suivant=p->suivant;
 }
 free(p);
 return l;}
```

Remarques (1)

- Ajout au milieu d'une liste connaissant la place qui précède celle où s'effectuera l'ajout
  - ajouter : Liste $\times$ Place $\times$ Élément $\rightarrow$ Liste
    - ajouter(L,p,e) : liste obtenue à partir de L en ajoutant une place contenant l'élément e, juste après la place p
- Enlever un élément d'une liste connaissant sa place
  - enlever : Liste $\times$ Place $\rightarrow$ Liste
    - enlever(L,p) : liste obtenue à partir de L en supprimant la place p et son contenu

Remarques (2)

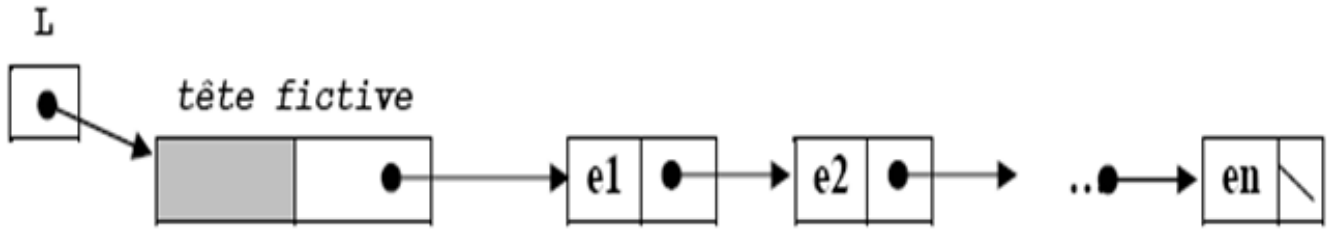
```
Liste ajouter(Liste l, Place p, element e) {
Place pc;

 pc=(Place)malloc(sizeof(Cellule));

 if (pc == NULL) {
   printf("Erreur: Problème de mémoire\n");
   exit(-1);
 }

 pc->valeur = e;
 pc->suivant = p->suivant;
 p->suivant = pc;

 return l;
}
```

Remarques (3)

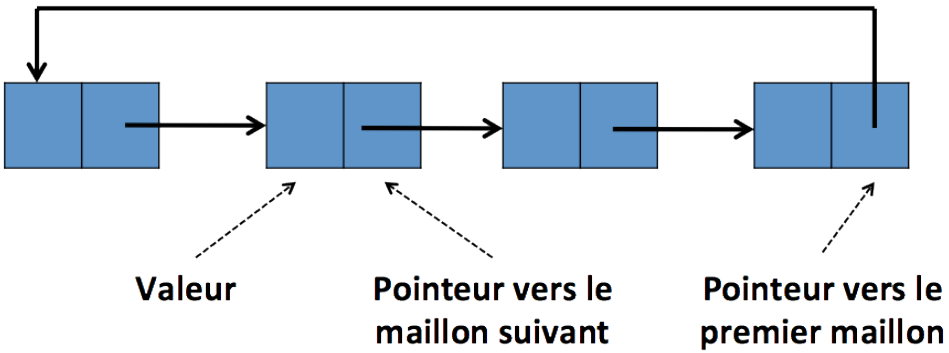
```
Liste enlever(Liste l, Place p) {
// p pointe sur l'élément àsupprimer
 Place pred; //pred pointe avant p

 if (p == l)
   l = succ(l,p);

 else {
     pred=l;
     while (succ(l,pred) != p)
          pred = succ(l,pred);

     pred->suivant = p->suivant;
      }
 free(p);
 return l;
}
```

Variantes de Listes Chaînées

- Liste triée
- Liste avec tête fictive
  - Mettre en tête de liste une zone qui ne contient pas de valeur et reste toujours en tête

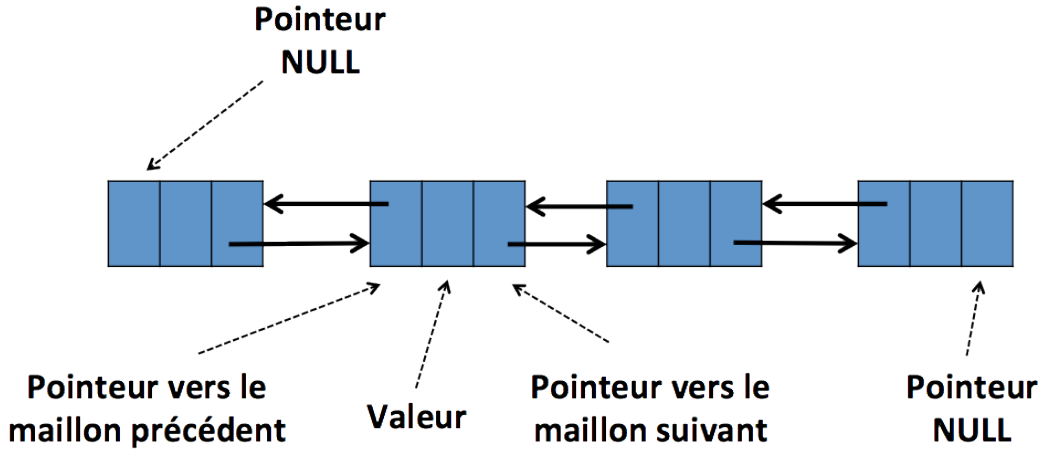


- Liste chaînée circulaire

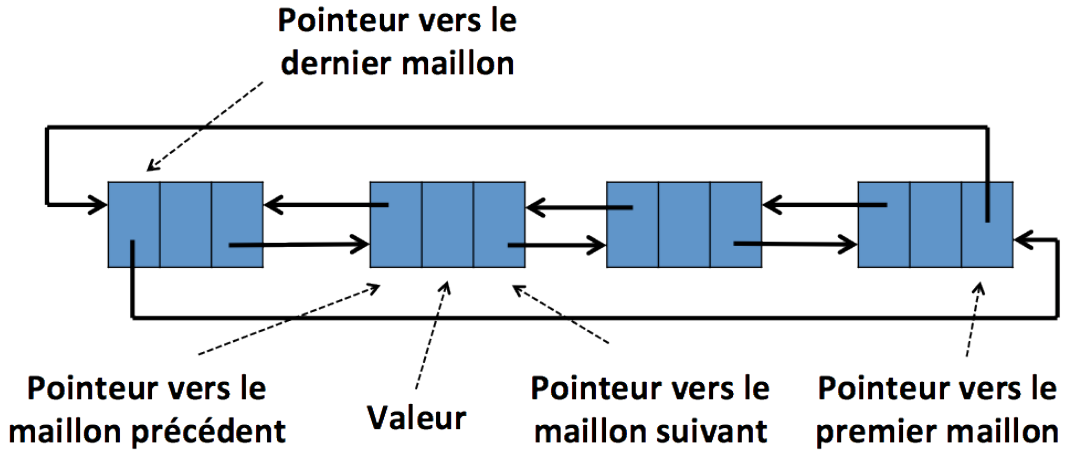


Variantes de Listes Chaînées

- Liste doublement chaînée



- Liste doublement chaînée circulaire



Exemples d'Applications d'une Liste

- Codage des polynômes
  - la liste d'entiers (0,1,0,-1,4) représente le polynôme à une indéterminée X : $4X^4 - X^3 + X$
- Codage des grands nombres
- Codage des matrices creuses
- Implémentation d'un logiciel de traitement de texte
  - Un texte est représenté par une liste de paragraphes
- Représentation d'un graphe

Complexité

Opération	Représentation contiguë	Représentation chaînée
liste_vide	O(1)	O(1)
ajout/suppression en tête	O(n)	O(1)
ajout/suppression générale	O(n)	O(n)
ajout/suppression en queue	O(1)	O(n)
accès	O(1)	O(n)
concaténation	O(n)	O(1)