Algorithmique & programmation

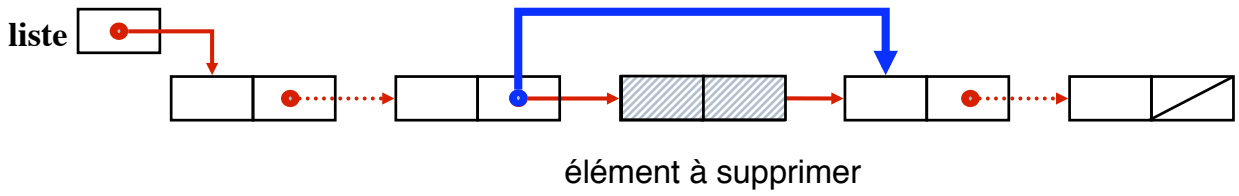
Chapitre 4 : Listes chaînées

Mise à jour

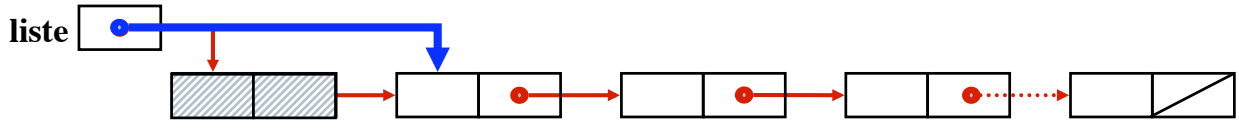
Suppression

Suppression d'un élément

- Cas général

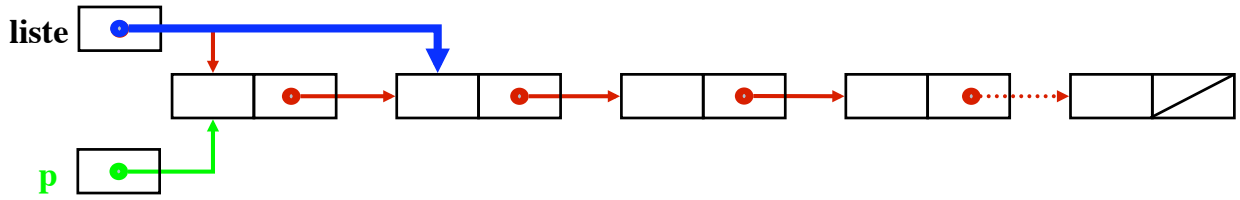


- Cas particulier du premier élément



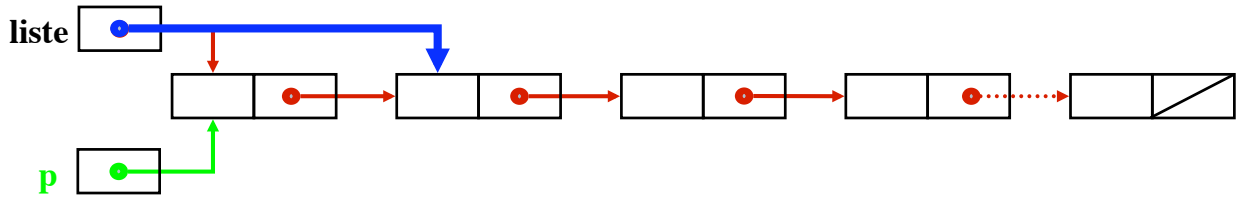
Suppression du premier élément

procédure supptête (dr liste : pointeur) ;
spécification *{liste ≠ nil}* ➔ *{1ier élément supprimé}*
    p : pointeur ;
debproc
    p := liste ;
    liste := liste↑.suivant ;
    laisser (p) ;
finproc ;



Suppression du premier élément (**ada**)

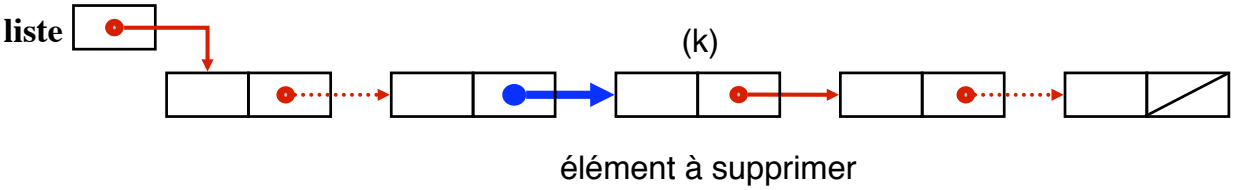
```
procedure Liberer is new
        Unchecked_Deallocation(Tr_ListEnt, Ta_ListEnt);

procedure supptête (liste : in out Ta_ListEnt) is
   p : Ta_ListEnt;
begin
   p := liste ;
   liste := liste.all.suivant ;
   Liberer(p) ;
end supptête;
```



Suppression par position

procédure supprimek (dr liste : pointeur ; d k : entier ;
r possible : booléen) ;

spécification *{k>0}* ➔ *{(possible, on a supprimé le $k^{ième}$ élément)*
v (¬ possible, suppression impossible)}



On va se débrouiller pour faire une suppression en tête

k = 1

Suppression par position

- Schéma récursif

➤ liste⁺ = <> ⇨ possible := faux ; ✳

➤ liste⁺ ≠ <>

➤➤ k = 1

⇨ suptête (liste) ; possible := vrai ; ✳

➤➤ k ≠ 1

⇨ supprimek (liste↑.suivant, k-1, possible) ;

Suppression par position

procédure supprimek (dr liste : pointeur ; d k : entier ;
r possible : booléen) ;

spécification *{k>0}* ➔ *{(possible, on a supprimé le k^ième élément)*
v (¬ possible, suppression impossible)}

debproc
  si liste = nil **alors**
    possible := faux ;
  sinonsi k = 1 **alors**
    possible := vrai ;
    supptête (liste) ;
  sinon
    supprimek (liste↑.suivant, k-1, possible) ;
  finsi ;
finproc ;

- **Si** k≤0 **alors** parcours complet de la liste

Suppression par position (**ada**)

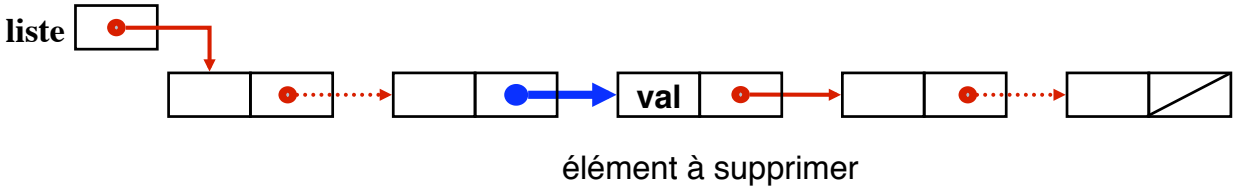
```
procedure supprimek (liste : in out Ta_ListEnt;
          k : in integer; possible : out boolean) is
begin
   if liste = null then
     possible := false ;
   else if k = 1 then
     possible := true;
     supptête (liste) ;
   else
     supprimek (liste.all.suivant, k-1, possible) ;
   end if ;
end supprimek;
```

- **Si** k≤0 **alors** parcours complet de la liste

Suppression associative

procédure suppval (dr liste : pointeur ; d val : t ;
r possible : booléen) ;

spécification { } ➔ *{(possible, suppression de la 1ère occurrence de val)*
v (¬ possible, suppression impossible)}



élément à supprimer

On va se débrouiller pour pointer sur la cellule qui contient la première occurrence de val

On fera alors une suppression en tête

Suppression associative

- Schéma récursif

➤ liste⁺ = <> ⇨ possible := faux ; ✳

➤ liste⁺ ≠ <>

➤➤ liste↑.info = val
⇨ suptête (liste) ; possible := vrai ; ✳

➤➤ liste↑.info ≠ val
⇨ suppval (liste↑.suivant, val, possible) ;

Suppression associative

procédure suppval (dr liste : pointeur ; d val : t ; r possible : booléen) ;
spécification *{ }* ➔ *{(possible, suppression de la 1ère occurrence de val)*
v (¬ possible, suppression impossible)}

debproc
  si liste = nil **alors**
    possible := faux ;
  sinonsi liste↑.info = val **alors**
    supptête (liste) ;
    possible := vrai ;
  sinon
    suppval (liste↑.suivant, val, possible) ;
  finsi ;
finproc ;

Suppression associative (**ada**)

```
procedure suppval (liste : in out Ta_ListEnt;
        val : in integer; possible : out boolean) is
begin
   if liste = null then
     possible := false ;
   else if liste.all.info = val then
     supptête (liste) ;
     possible := true ;
   else
     suppval (liste.all.suivant, val, possible) ;
   end if ;
end suppval ;
```