# Les listes (2)

Nour-Eddine Oussous, Éric Wegrzynowski

Licence ST-A, USTL - API2

2 novembre 2009

## 1 Parcours de listes

- Longueur d'une liste
- Accès à un élément par son rang
- Recherche dans une liste

## 2 Modification de listes

- Ajout d'un élément en fin de liste
- Insertion d'un élément à un rang donné

# Spécification

## Spécification

$$\begin{array}{rcrcl} \texttt{longueur} & : & \mathit{Liste}(E) & \longrightarrow & \mathbb{N} \\ & & \ell & \longmapsto & \text{nbre d'élts de } \ell \end{array}$$

# Spécification



Nour-Eddine Oussous, Éric Wegrzynowski



## Spécification

$$\begin{array}{rcccl} \texttt{longueur} & : & \mathit{Liste}(E) & \longrightarrow & \mathbb{N} \\ & & \ell & \longmapsto & \text{nbre d'élts de } \ell \end{array}$$

## Exemples

$$\begin{array}{rcl} \texttt{longueur}(()) & = & 0 \\ \texttt{longueur}((3,1,4,1,5,9,2)) & = & 7 \end{array}$$

# Algorithme récursif

```
longueur(L) =
```

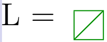
# Algorithme récursif

$L =$ 

`longueur(L) =`

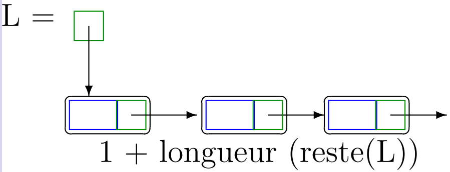
- 0 si `L` est vide

# Algorithme récursif




`longueur(L) =`

- 0 si L est vide




- `1+longueur(reste(l))` sinon

# Algorithme récursif en Pascal

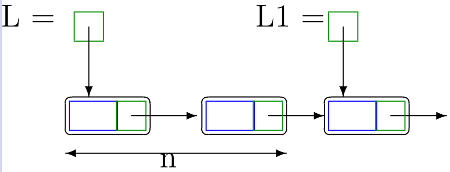


Nour-Eddine Oussous, Éric Wegrzynowski



```
// longueur(l) = nombre d'éléments de l
function longueur(l : LISTE) : CARDINAL;
begin
   if estListeVide(l) then
      longueur := 0
   else
      longueur := 1+longueur(reste(l));
end {longueur};
```

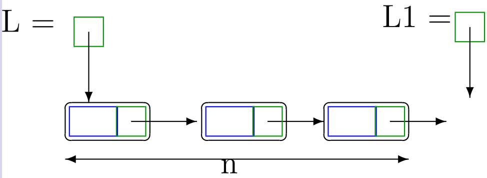
# Algorithme itératif




{n = nbre d'élts de L déjà comptés
L1 = liste des élts de L restant à compter }

# Algorithme itératif



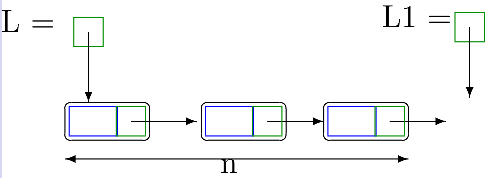



{`n` = nbre d'élts de `L` déjà comptés
`L1` = liste des élts de `L` restant à compter }

```
n := n+1
L1 := reste(L1)
```

{`n` = nbre d'élts de `L` déjà comptés
`L1` = liste des élts de `L` restant à compter }

# Algorithme itératif




{`n` = nbre d'élts de `L` déjà comptés
`L1` = liste des élts de `L` restant à compter }

```
n := n+1
L1 := reste(L1)
```

{`n` = nbre d'élts de `L` déjà comptés
`L1` = liste des élts de `L` restant à compter }

À faire tant que L1 non vide

# Algorithme itératif en Pascal



```
// longueur(l) = nombre d'éléments de l
function longueur(l : LISTE) : CARDINAL;
var
   n  : CARDINAL;
   l1 : LISTE;
begin
   n := 0;
   l1 := l;
   { n = nbre d'élts de l déjà parcourus}
   while not(estListeVide(l1)) do begin
      n := n+1;
      l1 := reste(l1);
      { n = nbre d'élts de l déjà parcourus}
   end {while};
   longueur := n;
end {longueur};
```

# Coût du calcul de la longueur



Nour-Eddine Oussous, Éric Wegrzynowski



$c(\ell) =$ nombre d'additions, ou d'appels récursifs, ou d'étapes du `TantQue`.

# Coût du calcul de la longueur



Nour-Eddine Oussous, Éric Wegrzynowski



$c(\ell)$ = nombre d'additions, ou d'appels récursifs, ou d'étapes du `TantQue`.

$$\begin{array}{rcl} c(()) & = & 0 \\ c(< x; \ell >) & = & 1 + c(\ell) \end{array}$$

# Coût du calcul de la longueur



$c(\ell)$ = nombre d'additions, ou d'appels récursifs, ou d'étapes du `TantQue`.

$$\begin{array}{rcl} c(()) & = & 0 \\ c(< x; \ell >) & = & 1 + c(\ell) \end{array}$$

## Conclusion

$$c(\ell) = \texttt{longueur}(\ell)$$

algorithme linéaire en fonction de la longueur de la liste

# Rang d'un élément dans une liste

## Définition

Le rang d'un élément (ou d'une cellule) d'une liste est la position de cet élément dans la liste. Les positions sont numérotées à partir de 0.

# Rang d'un élément dans une liste



## Définition

Le rang d'un élément (ou d'une cellule) d'une liste est la position de cet élément dans la liste. Les positions sont numérotées à partir de 0.

## Exemples

Dans la liste (3,1,4,1,5,9,2)

- l'élément de rang 0 est 3,
- l'élément de rang 1 est 1,
- l'élement de rang 2 est 4, ...

# Spécification



## Spécification

$$\begin{array}{rcccl} \texttt{acces} & : & \mathbb{N} \times \mathit{Liste}(E) & \longrightarrow & \mathit{Liste}(E) \\ & & k, \ell & \longmapsto & \ell' \end{array}$$

$\ell' =$ sous-liste de $\ell$ débutant par l'élément de rang $k$.

**CU :** $\ell$ non vide et $k < \texttt{longueur}(\ell)$.

# Spécification



Nour-Eddine Oussous, Éric Wegrzynowski



## Spécification

$$\begin{array}{rcccl} \texttt{acces} & : & \mathbb{N} \times \mathit{Liste}(E) & \longrightarrow & \mathit{Liste}(E) \\ & & k, \ell & \longmapsto & \ell' \end{array}$$

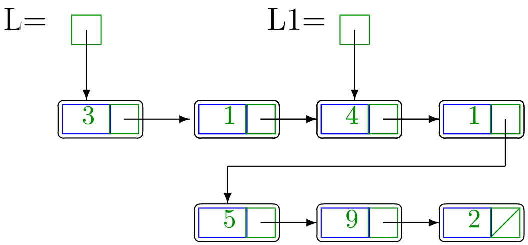
$\ell'$ = sous-liste de $\ell$ débutant par l'élément de rang $k$.
**CU :** $\ell$ non vide et $k < \texttt{longueur}(\ell)$.

## Exemples

$$\begin{array}{rcl} \texttt{acces}(2, (3, 1, 4, 1, 5, 9, 2)) & = & (4, 1, 5, 9, 2) \\ \texttt{acces}(7, (3, 1, 4, 1, 5, 9, 2)) & = & \text{Non défini} \end{array}$$

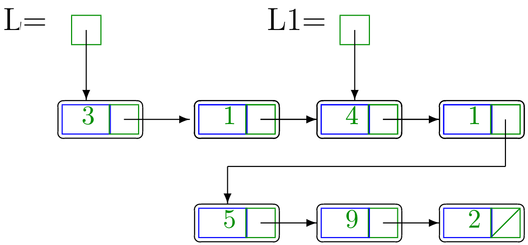
# Exemple

$$L1 := \texttt{acces}(2, (3, 1, 4, 1, 5, 9, 2))$$



# Exemple

$$L1 := \texttt{acces}(2, (3, 1, 4, 1, 5, 9, 2))$$




Aucune nouvelle cellule créée !

# Algorithme récursif en Pascal



```
// donne la liste débutant au k-ième élément de l
// les éléments étant numérotés à partir de 0
// CU : 0≤ k < long(l)
function acces(k : CARDINAL; l : LISTE) : LISTE;
begin
  if k=0 then
    acces := l
  else
    acces := acces(k-1,reste(l));
end {acces};
```

# Algorithme itératif en Pascal

```
// donne la liste débutant au k-ième élément de l
// les éléments étant numérotés à partir de 0
// CU : 0≤ k < long(l)
function acces(k : CARDINAL; l : LISTE) : LISTE;
var
   l1 : LISTE;
   i  : CARDINAL;
begin
   l1 := l;
   i := 0;
   { tete de l1  = élément de rang i de l }
   while i<k do begin
      l1 := reste(l1);
      i := i+1;
      { tete de l1 = élément de rang i de l }
   end {while};
   acces := l1;
end {acces};
```

# Coût de l'accès



Nour-Eddine Oussous, Éric Wegrzynowski



$c(k, \ell)$ = nombre d'appels récursifs, ou d'étapes du `TantQue`.

# Coût de l'accès



$c(k, \ell) =$ nombre d'appels récursifs, ou d'étapes du `TantQue`.

$$\begin{array}{rcl} c(0, \ell) & = & 0 \\ c(k, \ell) & = & 1 + c(k-1, \mathit{reste}(\ell)) \end{array}$$

# Coût de l'accès



$c(k, \ell) =$ nombre d'appels récursifs, ou d'étapes du `TantQue`.

$$\begin{array}{rcl} c(0, \ell) & = & 0 \\ c(k, \ell) & = & 1 + c(k-1, \mathit{reste}(\ell)) \end{array}$$

**Conclusion**

$$c(k, \ell) = k$$

le coût dépend uniquement de $k$.
Algorithme linéaire en fonction de $k$

# Différentes recherches dans une liste



Nour-Eddine Oussous, Éric Wegrzynowski



Étant donné un élément $e \in E$ et une liste $\ell \in Liste(E)$, on peut s'intéresser à

# Différentes recherches dans une liste



Nour-Eddine Oussous, Éric Wegrzynowski



Étant donné un élément $e \in E$ et une liste $\ell \in Liste(E)$, on peut s'intéresser à

- la première occurrence de $e$ dans $\ell$,

# Différentes recherches dans une liste



Nour-Eddine Oussous, Éric Wegrzynowski



Étant donné un élément $e \in E$ et une liste $\ell \in Liste(E)$, on peut s'intéresser à

- la première occurrence de $e$ dans $\ell$,
- la dernière occurrence de $e$ dans $\ell$,

# Différentes recherches dans une liste



Étant donné un élément $e \in E$ et une liste $\ell \in Liste(E)$, on peut s'intéresser à

- la première occurrence de $e$ dans $\ell$,
- la dernière occurrence de $e$ dans $\ell$,
- toutes les occurrences de $e$ dans $\ell$.

# Différentes issues d'une recherche



Nour-Eddine Oussous, Éric Wegrzynowski



Une recherche d'occurrence d'un élément $e$ dans une liste $\ell$ peut aboutir à

# Différentes issues d'une recherche



Nour-Eddine Oussous, Éric Wegrzynowski



Une recherche d'occurrence d'un élément $e$ dans une liste $\ell$ peut aboutir à

- un <u>succès</u> lorsque $e$ est dans $\ell$,

# Différentes issues d'une recherche



Une recherche d'occurrence d'un élément $e$ dans une liste $\ell$ peut aboutir à

- un <u>succès</u> lorsque $e$ est dans $\ell$,
- un <u>échec</u> lorsque $e$ n'est pas dans $\ell$

# Première occurrence



Nour-Eddine Oussous, Éric Wegrzynowski



## Spécification

$$\begin{array}{rcrcl} \texttt{premiereOccur} & : & E \times \mathit{Liste}(E) & \longrightarrow & \mathit{Liste}(E) \\ & & e, \ell & \longmapsto & \ell' \end{array}$$

$\ell'$ = la sous-liste de $\ell$ débutant à la première occurrence de $e$ si $e \in \ell$, $\ell' = ()$ sinon

# Première occurrence



## Spécification

$$\begin{array}{rcccl} \texttt{premiereOccur} & : & E \times \mathit{Liste}(E) & \longrightarrow & \mathit{Liste}(E) \\ & & e, \ell & \longmapsto & \ell' \end{array}$$

$\ell' =$ la sous-liste de $\ell$ débutant à la première occurrence de $e$ si $e \in \ell$, $\ell' = ()$ sinon

## Exemples

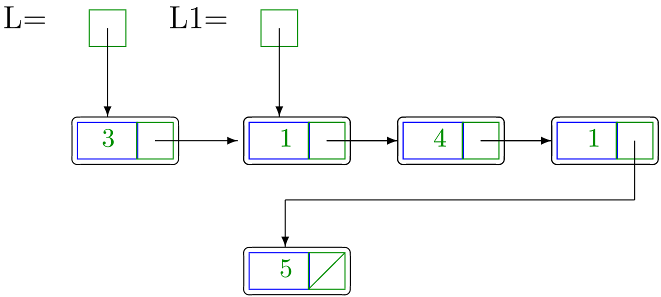
$$\begin{array}{rcl} \texttt{premiereOccur}(1, (3, 1, 4, 1, 5)) & = & (1, 4, 1, 5) \\ \texttt{premiereOccur}(6, (3, 1, 4, 1, 5)) & = & () \end{array}$$

Nour-Eddine Oussous, Éric Wegrzynowski



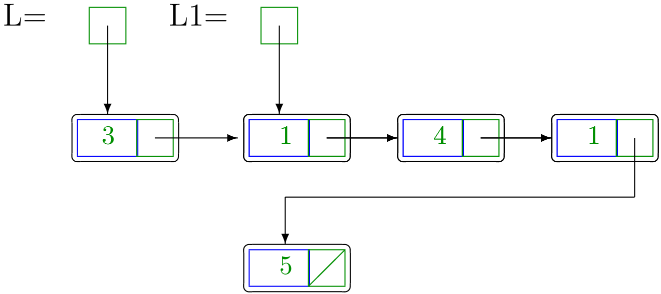
# Exemple

$$L1 := \texttt{premiereOccur}(1, (3, 1, 4, 1, 5))$$



# Exemple

$$L1 := \texttt{premiereOccur}(1, (3, 1, 4, 1, 5))$$




Aucune nouvelle cellule créée !

# Algorithme récursif en Pascal



```
// premiereOccur(e,l) =
//   la liste débutant à la première occurrence
//      de e dans l si elle existe
//   la liste vide sinon
function premiereOccur(const e : ELEMENT;
                       const l : LISTE) : LISTE;
begin
  if estListeVide(l)  then
    premiereOccur := LISTEVIDE
  else
    if tete(l)=e then
      premiereOccur := l
    else
      premiereOccur := premiereOccur(e,reste(l));
end {premiereOccur};
```

# Coût de la recherche de la première occurrence

$c(e, \ell)$ = nombre d'appels récursifs (ou d'accès à la tête d'une liste).

# Coût de la recherche de la première occurrence

$c(e, \ell) =$ nombre d'appels récursifs (ou d'accès à la tête d'une liste).

- dépend de $e$ et de $\ell$

# Coût de la recherche de la première occurrence

$c(e, \ell) =$ nombre d'appels récursifs (ou d'accès à la tête d'une liste).

- dépend de $e$ et de $\ell$
- pour une recherche qui échoue $c(e, \ell) = \texttt{longueur}(\ell)$

# Coût de la recherche de la première occurrence



Nour-Eddine Oussous, Éric Wegrzynowski



$c(e, \ell) =$ nombre d'appels récursifs (ou d'accès à la tête d'une liste).

- dépend de $e$ et de $\ell$
- pour une recherche qui échoue $c(e, \ell) = \texttt{longueur}(\ell)$
- pour une recherche avec succès

# Coût de la recherche de la première occurrence



$c(e, \ell) =$ nombre d'appels récursifs (ou d'accès à la tête d'une liste).

- dépend de $e$ et de $\ell$
- pour une recherche qui échoue $c(e, \ell) = \texttt{longueur}(\ell)$
- pour une recherche avec succès
  - dans le meilleur des cas $c(e, \ell) = 1$

# Coût de la recherche de la première occurrence



$c(e, \ell)$ = nombre d'appels récursifs (ou d'accès à la tête d'une liste).

- dépend de $e$ et de $\ell$
- pour une recherche qui échoue $c(e, \ell) = \texttt{longueur}(\ell)$
- pour une recherche avec succès
  - dans le meilleur des cas $c(e, \ell) = 1$
  - dans le pire des cas $c(e, \ell) = \texttt{longueur}(\ell)$

# Coût de la recherche de la première occurrence



$c(e, \ell)$ = nombre d'appels récursifs (ou d'accès à la tête d'une liste).

- dépend de $e$ et de $\ell$
- pour une recherche qui échoue $c(e, \ell) = \texttt{longueur}(\ell)$
- pour une recherche avec succès
  - dans le meilleur des cas $c(e, \ell) = 1$
  - dans le pire des cas $c(e, \ell) = \texttt{longueur}(\ell)$

**Conclusion**

Coût de la recherche compris entre 0 (liste vide) et $\texttt{longueur}(\ell)$
Algorithme linéaire (dans le pire des cas) en fonction de la longueur.

# Dernière occurrence

## Spécification

$$\begin{array}{rcccl} \texttt{derniereOccur} & : & E \times \mathit{Liste}(E) & \longrightarrow & \mathit{Liste}(E) \\ & & e, \ell & \longmapsto & \ell' \end{array}$$

$\ell'$ = la sous-liste de $\ell$ débutant à la dernière occurrence de $e$ si $e \in \ell$, $\ell' = ()$ sinon

# Dernière occurrence



Nour-Eddine Oussous, Éric Wegrzynowski



## Spécification

$$\begin{array}{llcll} \texttt{derniereOccur} & : & E \times Liste(E) & \longrightarrow & Liste(E) \\ & & e, \ell & \longmapsto & \ell' \end{array}$$

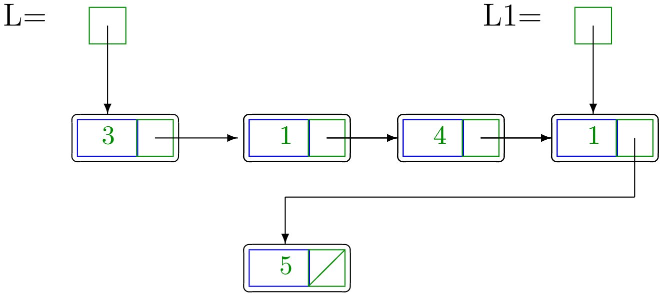
$\ell'$ = la sous-liste de $\ell$ débutant à la dernière occurrence de $e$ si $e \in \ell$, $\ell' = ()$ sinon

## Exemples

$$\begin{array}{lcl} \texttt{derniereOccur}(1, (3, 1, 4, 1, 5)) & = & (1, 5) \\ \texttt{derniereOccur}(6, (3, 1, 4, 1, 5)) & = & () \end{array}$$

# Exemple

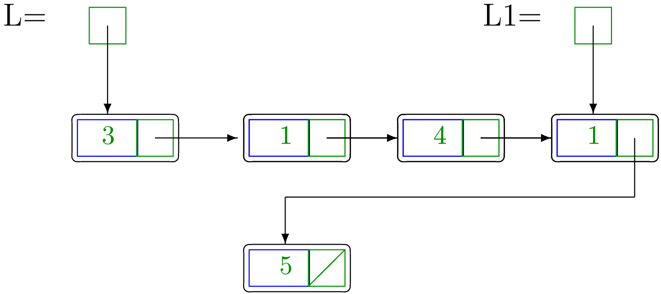
$$L1 := \texttt{derniereOccur}(1, (3, 1, 4, 1, 5))$$



# Exemple



$$L1 := \texttt{derniereOccur}(1, (3, 1, 4, 1, 5))$$




Aucune nouvelle cellule créée !

# Algorithme récursif en Pascal



```
// dernereOccur(e,l) =
//   la liste débutant à la dernière occurrence
//      de e dans l  si elle existe
//   la liste vide sinon
function derniereOccur(const e : ELEMENT;
                       const l : LISTE) : LISTE;
var
   l1 : LISTE;
begin
  if estListeVide(l)  then
     derniereOccur := LISTEVIDE;
  else begin
     l1 := derniereOccur(e, reste(l));
     if estListeVide(l1) and (tete(l) = e) then
        derniereOccur := l
     else
        derniereOccur := l1;
  end {if};
end {derniereOccur};
```

# Coût de la recherche de la dernière occurrence



Nour-Eddine Oussous, Éric Wegrzynowski



$c(e, \ell) =$ nombre d'appels récursifs (ou d'accès à la tête d'une liste).

# Coût de la recherche de la dernière occurrence



$c(e, \ell)$ = nombre d'appels récursifs (ou d'accès à la tête d'une liste).

- dans tous les cas $c(e, \ell) = \texttt{longueur}(\ell)$

# Coût de la recherche de la dernière occurrence



Nour-Eddine Oussous, Éric Wegrzynowski



$c(e, \ell)$ = nombre d'appels récursifs (ou d'accès à la tête d'une liste).

- dans tous les cas $c(e, \ell) = \texttt{longueur}(\ell)$
- ne dépend donc pas de $e$

# Coût de la recherche de la dernière occurrence



Nour-Eddine Oussous, Éric Wegrzynowski



$c(e, \ell) =$ nombre d'appels récursifs (ou d'accès à la tête d'une liste).

- dans tous les cas $c(e, \ell) = \texttt{longueur}(\ell)$
- ne dépend donc pas de $e$

**Conclusion**

Coût de la recherche égal à $\texttt{longueur}(\ell)$
Algorithme linéaire en fonction de la longueur.

# Dernier élément



## Spécification

$$\begin{array}{rcccl} \texttt{dernier} & : & \mathit{Liste}(E) & \longrightarrow & \mathit{Liste}(E) \\ & & \ell & \longmapsto & \ell' \end{array}$$

$\ell' =$ la sous-liste de $\ell$ débutant au dernier élément de $\ell$ si $\ell$ n'est pas vide, $\ell' = ()$ sinon

# Dernier élément



## Spécification

$$\begin{array}{rcrcl} \texttt{dernier} & : & \mathit{Liste}(E) & \longrightarrow & \mathit{Liste}(E) \\ & & \ell & \longmapsto & \ell' \end{array}$$

$\ell' =$ la sous-liste de $\ell$ débutant au dernier élément de $\ell$ si $\ell$ n'est pas vide, $\ell' = ()$ sinon

## Exemples

$$\begin{array}{rcl} \texttt{dernier}(()) & = & () \\ \texttt{dernier}((3,1,4,1,5)) & = & (5) \end{array}$$

# Algorithme récursif en Pascal



```
// derner(l) =
//   la liste débutant au  dernier élément
//      de  l  si l n'est pas vide
//   la liste vide sinon
function dernier(const l : LISTE) : LISTE;
begin
  if estListeVide(l)  then
     dernier := LISTEVIDE
  else if estListeVide(reste(l)) then
          dernier := l
       else
          dernier := dernier(reste(l));
end {dernier};
```

# Coût de la recherche du dernier élément



Nour-Eddine Oussous, Éric Wegrzynowski



$c(\ell) =$ nombre d'appels récursifs

# Coût de la recherche du dernier élément



Nour-Eddine Oussous, Éric Wegrzynowski



$c(\ell) =$ nombre d'appels récursifs

- $c(\ell) = 0$ si $\ell$ est vide

# Coût de la recherche du dernier élément



$c(\ell)$ = nombre d'appels récursifs

- $c(\ell) = 0$ si $\ell$ est vide
- dans tous les autres cas $c(\ell) = \texttt{longueur}(\ell) - 1$

# Coût de la recherche du dernier élément

$c(\ell)$ = nombre d'appels récursifs

- $c(\ell) = 0$ si $\ell$ est vide
- dans tous les autres cas $c(\ell) = \texttt{longueur}(\ell) - 1$

**Conclusion**

Coût de la recherche égal à $\texttt{longueur}(\ell) - 1$
Algorithme linéaire en fonction de la longueur.

# Spécification

## Spécification

$$\begin{array}{llcll} \texttt{ajouteEnFin} & : & E \times Liste(E) & \longrightarrow & Liste(E) \\ & & e, \ell & \longmapsto & \ell' \end{array}$$

$\ell' =$ liste $\ell$ <u>modifiée</u> par l'ajout d'un nouvel élément $e$ à la fin

# Spécification



## Spécification

$$\begin{array}{rcccl} \texttt{ajouteEnFin} & : & E \times Liste(E) & \longrightarrow & Liste(E) \\ & & e, \ell & \longmapsto & \ell' \end{array}$$

$\ell'$ = liste $\ell$ modifiée par l'ajout d'un nouvel élément $e$ à la fin

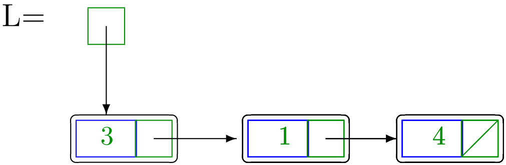
## Exemples

$$\begin{array}{rcl} \texttt{ajouteEnFin}(1, ()) & = & (1) \\ \texttt{ajouteEnFin}(9, (3, 1, 4, 1, 5)) & = & (3, 1, 4, 1, 5, 9) \end{array}$$

# Algorithme

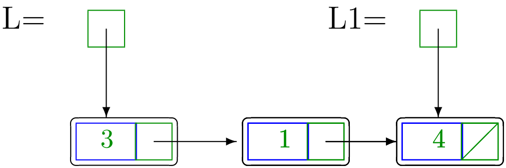


Ajout de 1 en fin de la liste L=(3,1,4).



# Algorithme

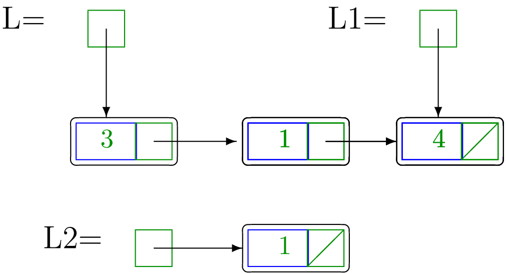
Ajout de 1 en fin de la liste L=(3,1,4).




```
L1 := dernier(L)
```

# Algorithme

Ajout de 1 en fin de la liste L=(3,1,4).

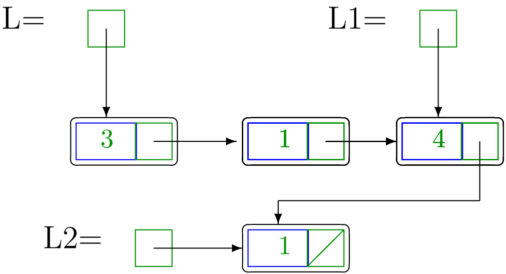



```
L1 := dernier(L)
L2 :=
ajouteEnTete(1,LISTEVIDE)
```

# Algorithme



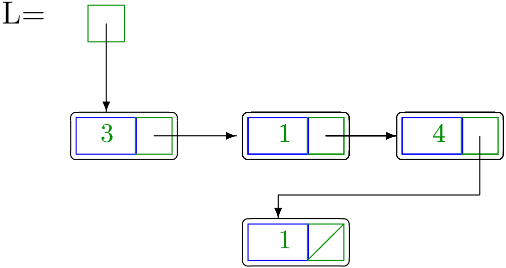
Ajout de 1 en fin de la liste L=(3,1,4).




```
L1 := dernier(L)
L2 :=
ajouteEnTete(1,LISTEVIDE)
modifierReste(L1,L2)
```

# Algorithme

Ajout de 1 en fin de la liste L=(3,1,4).

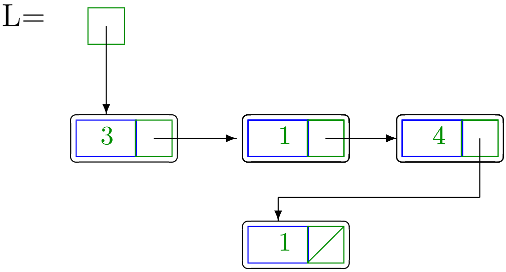



```
L1 := dernier(L)
L2 :=
ajouteEnTete(1,LISTEVIDE)
modifierReste(L1,L2)
```

Terminé !

# Algorithme

Ajout de 1 en fin de la liste L=(3,1,4).




```
L1 := dernier(L)
L2 :=
ajouteEnTete(1,LISTEVIDE)
modifierReste(L1,L2)
```

Terminé !

Création d'une cellule

# Algorithme en Pascal

```
// ajouteEnFin(e,l) : ajoute un nouvel
// élément e à la fin de  l
// la liste l est modifiée
procedure ajouteEnFin(const e : ELEMENT;
                      var l : LISTE);
var
   l1,l2 : LISTE;
begin
  l1 := dernier(l);
  l2 := ajouteEnTete(e,LISTEVIDE);
  if estListeVide(l1) then
     l := l2
  else
     modifierReste(l1,l2);
end {ajouterEnFin};
```

# Spécification



## Spécification

$$\begin{array}{rcccc} \texttt{inserer} & : & E \times \mathit{Liste}(E) \times \mathbb{N} & \longrightarrow & \mathit{Liste}(E) \\ & & e, \ell, k & \longmapsto & \ell' \end{array}$$

$\ell'$ = liste $\ell$ modifiée par l'insertion d'un nouvel élément $e$ au rang $k$.

**CU :** $1 \leq k \leq \texttt{longueur}(\ell)$

# Spécification

## Spécification

$$\begin{array}{rcccc} \mathtt{inserer} & : & E \times Liste(E) \times \mathbb{N} & \longrightarrow & Liste(E) \\ & & e, \ell, k & \longmapsto & \ell' \end{array}$$

$\ell'$ = liste $\ell$ modifiée par l'insertion d'un nouvel élément $e$ au rang $k$.

**CU :** $1 \leq k \leq \mathtt{longueur}(\ell)$

## Exemples

$$\begin{array}{rcl} \mathtt{inserer}(4, (3, 1, 1, 5), 2) & = & (3, 1, 4, 1, 5) \\ \mathtt{inserer}(9, (3, 1, 4, 1, 5), 5) & = & (3, 1, 4, 1, 5, 9) \\ \mathtt{inserer}(9, (3, 1, 4, 1, 5), 6) & = & \text{Non défini !} \end{array}$$

# Algorithme en Pascal



```
// inserer(e,l,k): insere un nouvel élément e au
// rang k dans  l
// la liste l est modifiée
// CU : 1≤ k≤longueur(l)
procedure inserer(const e : ELEMENT;
                  const l : LISTE;
                  const k : CARDINAL);
var
   l1,l2 : LISTE;
begin
  l1 := acces(k-1,l);
  l2 := ajouteEnTete(e,reste(l1));
  modifierReste(l1,l2);
end {inserer};
```