Algorithmique & programmation

Chapitre 4 : Listes chaînées

Définition récursive

Parcours

Définition récursive d'une liste

- Une liste chaînée est
  - soit une liste vide
  - soit composée d'une cellule (la première) chaînée à une liste (obtenue après suppression de la première cellule de la liste d'origine)
- Si une liste **liste** n'est pas vide
  - elle est composée
    - d'une cellule : **liste** *(on considère uniquement la première cellule)*
    - d'une liste : **liste↑.suivant**

Utilisation de la définition récursive

- On veut écrire une **procédure** **écritlistegd** qui affiche les éléments contenus dans une liste définie comme suit :

  - **type cellule = structure**
    info : t ;
    suivant : pointeur ;
    fin ;
  - **type pointeur = ↑cellule ;**

- **liste** est de type pointeur

Parcours de gauche à droite

procédure écritliste**gd** (d liste : pointeur) ;
spécification *{}* ➔ *{les éléments de* liste$^+$ *ont été écrits de gauche à droite}*

- Réfléchissons à l'écriture de **écritlistegd** en considérant **écritlistegd** connue
  - **si liste$^+$** = <> **alors** il n'y a rien à faire
  - **si liste$^+$** ≠ <> **alors**
    1. afficher l'**info** contenue dans la cellule pointée par **liste**
    2. afficher les **info** contenues dans le reste de la **liste**
       - **Comment ????**
       - **On va utiliser écritlistgd sur liste↑.suivant**

Parcours de gauche à droite

- Pour résumer

➤ liste⁺ = <> ⇨ *parcours terminé* ✳
➤ liste⁺ ≠ <> ⇨ écritexte (liste↑.info) ;
écritlistegd (liste↑.suivant) ;

procédure écritliste**gd** (d liste : pointeur) ;
spécification *{}* ➔ *{les éléments de* liste⁺ *ont été écrits de gauche à droite}*
debproc
si liste ≠ nil **alors**
écritexte (liste↑.info) ;
écritliste**gd** (liste↑.suivant) ;
finsi ;
finproc ;

C'est une procédure récursive

Parcours de gauche à droite (ada)

```
procedure écritlistegd (liste : in Ta_ListEnt) is
--spec {} ➔ {les éléments de liste⁺ ont été écrits de
   gauche à droite}
begin
   if liste /= null then
     écritexte (liste.all.info) ;
     écritlistegd (liste.all.suivant) ;
   end if ;
end écritlistegd;
```

Parcours de gauche à droite

Schéma récursif

- liste$^+$ = <> ⇨ *parcours terminé* ✳
- liste$^+$ ≠ <>
  ⇨ traiter la première cellule ;
  effectuer le parcours de liste↑.suivant ;

```
procédure parcoursgd (d liste : pointeur) ;
debproc
    si liste ≠ nil alors
        traiter (liste) ;
        parcoursgd (liste↑.suivant) ;
    finsi ;
finproc ;
```


Parcours de gauche à droite


```
procédure parcoursgd (d liste : pointeur) ;
debproc
    si liste ≠ nil alors
        traiter (liste) ;
        parcoursgd (liste↑.suivant) ;
    finsi ;
finproc ;
```


Attention !

- Le paramètre **liste** est de type donnée
- ➔ **traiter (liste)** ne doit pas modifier la valeur de **liste**
  - donc **liste** est un paramètre donnée de **traiter**

Parcours de gauche à droite (ada)

```
procedure parcourslistegd (liste : in Ta_ListXX) is
--spec {} ➔ {les éléments de liste+ ont été traités
   de gauche à droite}
begin
   if liste /= null then
      traiterinfo (liste.all.info) ;
      parcourslistegd (liste.all.suivant) ;
   end if ;
end parcourslistegd;
```


Parcours de gauche à droite

- Schéma itératif (parcours complet)

```
procédure parcoursgd (d liste : pointeur);
    liste1 : pointeur ;
debproc
    liste1 := liste ;
    tantque liste1 ≠ nil faire
        traiter (liste1) ;
        liste1 := liste1↑.suivant ;
    finfaire ;
finproc ;
```

Attention

- liste est un paramètre donnée
- il faut le recopier dans une variable locale pour pouvoir passer au suivant

Parcours de gauche à droite (ada)

- Schéma itératif (parcours complet)

```
procedure traiterlistegd (liste : in Ta_ListXX) is
--spec {} ➔ {les éléments de liste+ ont été
   traités de gauche à droite}
   liste1 : Ta_ListeXX
begin
   liste1 := liste ;
   while liste1 /= null loop
     traiterinfo (liste1.all.info) ;
     liste1 := liste1.all.suivant ;
   end loop ;
end traiterlistegd ;
```


Parcours de droite à gauche

procédure écritlistedg (d liste : pointeur) ;
spécification *{}* ➔ *{les éléments de* liste+ *ont été écrits de droite à gauche}*

- Réfléchissons à l'écriture de **écritlistedg** en considérant **écritlistedg** connue
  - **si liste+** = <> **alors** il n'y a rien à faire
  - **si liste+** ≠ <> **alors**
    1. il faut afficher les **info** contenues dans le reste de la **liste**
       - **Comment ????**
       - **On va utiliser écritlistdg sur liste↑.suivant**
    2. afficher l'**info** contenue dans la cellule pointée par **liste**

Parcours de droite à gauche

Pour résumer

- liste⁺ = <> ⇨ *parcours terminé* ✳
- liste⁺ ≠ <> ⇨ écritlistedg (liste↑.suivant) ;
  écritexte (liste↑.info) ;

procédure écritlistedg (d liste : pointeur) ;
spécification *{}* ➔ *{les éléments de* liste⁺ *ont été écrits de droite à gauche}*
debproc
    si liste ≠ nil **alors**
        écritlistedg (liste↑.suivant) ;
        écritexte (liste↑.info) ;
    finsi ;
finproc ;

C'est une procédure récursive

Parcours de droite à gauche

Schéma récursif

- liste⁺ = <> ⇨ *parcours terminé* ✳
- liste⁺ ≠ <>
  ⇨ effectuer le parcours de liste↑.suivant ;
  traiter la première cellule ;

procédure parcoursdg (d liste : pointeur) ;
debproc
    si liste ≠ nil **alors**
        parcoursdg (liste↑.suivant) ;
        traiter (liste) ;
    finsi ;
finproc ;

Parcours de droite à gauche (ada)

- Schéma récursif

```
procedure traiterlistedg (liste : in Ta_ListXX) is
begin
   if liste /= null then
     parcoursdg (liste.all.suivant) ;
     traiter (liste) ;
   end if;
end traiterlistedg ;
```


Parcours de droite à gauche

- Schéma itératif
  - Plus dur !!! Pourquoi ??

procédure parcours**gd** (d liste : pointeur) ;
debproc
  si liste ≠ nil **alors**
    traiter (liste) ;
    parcours**gd** (liste↑.suivant) ;
  finsi ;
finproc ;

Appel récursif terminal
on ne fait rien après
vers itératif aisé

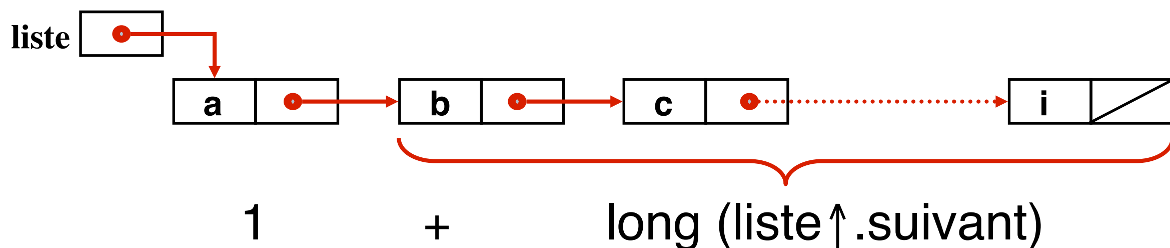
procédure parcours**dg** (d liste : pointeur) ;
debproc
  si liste ≠ nil **alors**
    parcours**dg** (liste↑.suivant) ;
    traiter (liste) ;
  finsi ;
finproc ;

Appel récursif non terminal
on fait quelque chose après
vers itératif plus dur

Calcul de la longueur d'une liste

- Vers le schéma récursif
  - **essayer de résoudre le problème sachant que l'on sait le résoudre**

- Schéma
  - **fonction** long (d liste : pointeur): entier ;



Calcul de la longueur d'une liste

- Schéma récursif

➤ liste⁺ = <> ⇨ résultat = 0 ; ✳

➤ liste⁺ ≠ <> ⇨ résultat = 1 + long (liste↑.suivant) ;

fonction long (d liste : pointeur) : entier ;

spécification *{ }* ➔ *{résultat = nombre d'éléments de liste⁺}*

debfonc

  si liste = nil **alors**

    retour 0 ;

  sinon

    retour 1 + long (liste↑.suivant) ;

  finsi ;

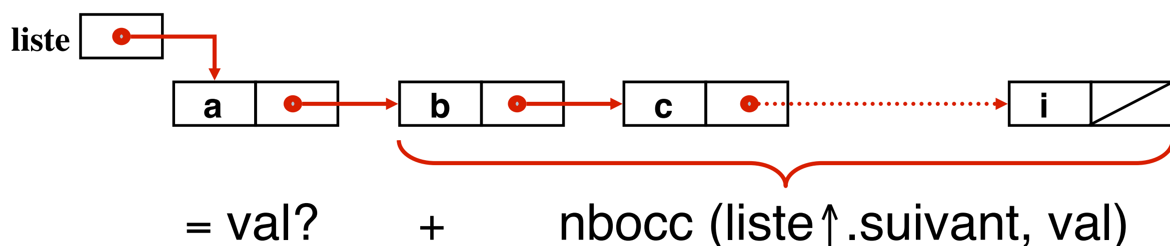
finfonc ;

Calcul de la longueur d'une liste (ada)

```
function long (liste : in Ta_ListEnt) return integer is
--spec { } ➔ {résultat = nombre d'éléments de liste+}
begin
   if liste = null then
     return 0 ;
   else
     return 1 + long (liste.all.suivant) ;
   end if ;
end long ;
```


Nombre d'occurrence de val

- Vers le schéma récursif
  - essayer de résoudre le problème sachant qu'on sait le résoudre (on sait calculer le nb d'occurrences)
- Schéma
  - **fonction** nbocc (d liste : pointeur , d val : t): entier ;



= val? + nbocc (liste↑.suivant, val)

Nombre d'occurrence de val

Schéma récursif

- liste$^+$ = <> ⇨ résultat = 0 ; ✳
- liste$^+$ ≠ <>
  - liste↑.info = val
    ⇨ résultat = 1 + nbocc (liste↑.suivant, val) ;
  - liste↑.info ≠ val
    ⇨ résultat = nbocc (liste↑.suivant, val) ;

Nombre d'occurrence de val

fonction nboccr (d liste : pointeur ; d val : t) : entier ;
spécification *{ }* ➔ *{résultat = nombre d'occurrences de val dans liste$^+$}*
debfonc
    si liste = nil **alors**
        retour 0 ;
    sinonsi liste↑.info = val **alors**
        retour 1 + nboccr (liste↑.suivant, val);
    sinon
        retour nboccr (liste↑.suivant, val) ;
    finsi ;
finfonc ;

- Toujours tester d'abord la **liste vide**, car liste↑.info et liste↑.suivant ne sont définis que si liste ≠ nil

Nombre d'occurrence de val (**ada**)

```
function nboccr (liste : in Ta_ListEnt; val : in integer)
                                             return integer is
--spec {}➔{résultat=nombre d'occ. de val dans liste+}
begin
   if liste = null then
     return 0 ;
   else if liste.all.info = val then
     return 1 + nboccr (liste.all.suivant, val) ;
   else
     return nboccr (liste.all.suivant, val) ;
   end if ;
end nboccr ;
```

- Toujours tester d'abord la **liste vide**, car liste.all.info et liste.all.suivant ne sont définis que si liste ≠ null