

# Structures de données, IMA S6

## Récurtivité

*d'après le cours de N. Devésa, Polytech'lille*

Laure Gonnord

<http://laure.gonnord.org/pro/teaching/>

[Laure.Gonnord@polytech-lille.fr](mailto:Laure.Gonnord@polytech-lille.fr)

Université Lille 1 - Polytech Lille

Février 2011



- 1 Rappels et Généralités
- 2 Algorithmes récursifs sur les listes chaînées
- 3 Fonctionnement de la récursivité

- 1 Rappels et Généralités
- 2 Algorithmes récursifs sur les listes chaînées
- 3 Fonctionnement de la récursivité

# Définition - Rappels

(S5,cours 3)

Un algorithme (une fonction, une procédure) est dit **récuratif** si sa définition (son code) contient un appel à lui-même.

Un algorithme qui n'est pas récuratif est dit **itératif**.

# Utilisations usuelles

Utilisations variées (liste non exhaustive) :

- Calcul de suite **récurive** (numérique, graphique. . .) (cf S5, Fibonacci, Factorielle)
  - Calcul de type « diviser pour régner » : recherche, tri, . . . (cf S5, tri fusion, recherche dichotomique)
  - Calcul sur des structures de données **inductives** (listes, arbres, . . .)
- ▶ Dans ce cours, utilisation de la récursivité sur les listes.

# Quelques Rappels du S5

**Factorielle** :  $n! = n \cdot (n - 1)!$

**Fonction** *fact*(*n*) : entier

**D**: *n* : entier positif ou nul

**Si** *n=0* **alors**

**Retourner** 1

**Sinon**

**Retourner** *n*\**fact*(*n-1*)

**Fsi**

{Appel récursif}

**FFonction**

## Quelques exemples classiques - 2

**Fonction** *somme* ( $n,r$ ) :entier

**D:**  $n,r$  : entier positifs ou nul

**Si**  $n = 1$  **alors**

| **Retourner**  $r + 1$

**Sinon**

| **Retourner** *somme* ( $n - 1$  ,  $r + n$  )

**Fsi**

**FFonction**

- ▶  $r$  est appelé paramètre d'**accumulation**.
- ▶ L'appel `somme(N,0)` calcule  $1 + 2 + \dots + N$ .

## Réversivité terminale (ou pas ?)

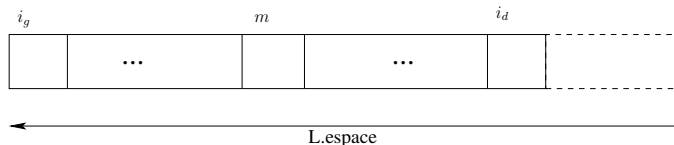
Un algorithme récursif est dit récursif **terminal** si l'appel récursif est la dernière instruction réalisée.

► Stockage non nécessaire de la valeur obtenue par récursivité.

- Factorielle :  $\text{fact}(n-1)$  puis multiplication par  $n$ , donc non récursif terminal.
- Somme : récursif terminal :  
 $\text{somme}(5, 0) = \text{somme}(4, 5) = \text{somme}(\dots) \dots = 15$

## Quelques exemples classiques - 3

**Recherche dichotomique** de  $elem$  dans une liste contiguë  $L$  :

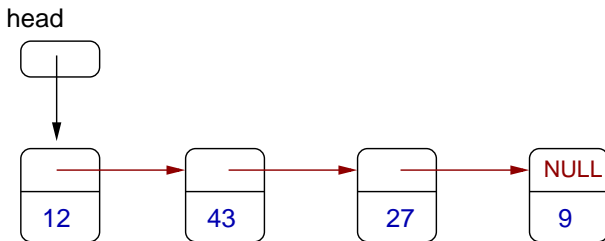


- Cas général : appel récursif à gauche ou à droite de  $m$  suivant le résultat de la comparaison  $L.espace[m] > elem$ ,
  - Cas terminaux :  $i_g > i_d$  (élément non trouvé) ou  $elem = L.espace[m]$  (trouvé)
- ▶ Écrire l'algorithme.

- 1 Rappels et Généralités
- 2 Algorithmes récursifs sur les listes chaînées
- 3 Fonctionnement de la récursivité

## Pourquoi ?

Parce que la définition des listes est elle même récursive :



- ▶ Une liste est un Pointeur de Cellule.
- ▶ Une liste est soit :
  - vide (cas terminal)
  - un élément (éventuellement cas terminal) “suivi d’une” liste (cas général)

# Exemples - 1

Longueur d'une liste  $l$  :

- si  $l = NULL$  : 0
- si  $l = elem \cdot l2$  :  $1 + \text{longueur}(l2)$ .

## Exemples - 2

Insertion de  $x$  dans une liste ordonnée  $l$  :

- si  $l = NULL$  : résultat =  $\langle x \rangle$
- si  $l = elem \cdot l2$  :
  - Si  $x = elem$  : résultat =  $l$
  - Si  $x < elem$  : résultat =  $x \cdot l$
  - Si  $x > elem$  : résultat =  $elem \cdot (\text{insertion de } x \text{ dans } l2)$

- 1 Rappels et Généralités
- 2 Algorithmes récursifs sur les listes chaînées
- 3 **Fonctionnement de la récursivité**

## Un autre algo récursif

Inversion d'une suite de caractères :  $s = \langle c_1, c_2, \dots, c_n, \cdot \rangle$ .

- Si  $s = \langle \cdot \rangle$  : terminé
- Si  $s = \langle c_1, c_2, \dots, c_n, \cdot \rangle$  : inverser  $\langle c_2, \dots, c_n, \cdot \rangle$  puis écrire  $c_1$ .

(la suite est lue au clavier par exemple)

**Action** *inverser()*

**L**: c : caractère

lire(c)

**Si**  $c \neq '.'$  **alors**

    inverser()

    ecrire(c)

**Fsi**

**FAction**

## Version itérative

On va mémoriser les caractères et les restituer en sens inverse :

► utilisation d'une **SD pile** (cf suite du cours).

**Action** *inverser\_it()*

L: c : caractère, P : pile de caractères

lire(c)

**Tq** *c! = '.'* **faire**

empiler(P,c)

lire(c)

**Ftq**

**Tq** *non pilevide(P)* **faire**

depiler(P,c)

ecrire(c)

**Ftq**

**Action**

# La pile d'exécution - 1

Lors de l'exécution d'un programme, cette pile :

- mémorise le contexte appelant lors de chaque appel de fonction (adresse des variables, du code de la prochaine instruction, ...)
- restitue ce contexte au retour.

## La pile d'exécution - 2

**Exemple** :  $p$  appelle  $int\ q(int)$  :

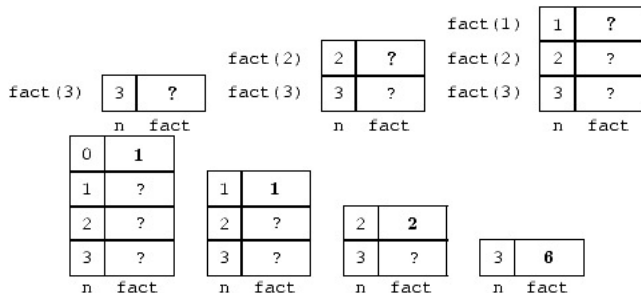
- sauver le contexte de  $p$  (adresse de l'instruction qui suit l'appel)
- empiler le paramètre transmis à  $q$  (valeur entière)
- préparer de la place pour le résultat

Ensuite on exécute  $q$

- place pour les objets locaux
  - exécution du code
  - dépilage des objets locaux
  - (le résultat est sur un registre ou dans une adresse précise empilée par  $p$ )
- ▶ en haut de pile on a l'adresse de l'instruction à faire !

# La pile d'exécution - 3

Exemple sur `Facto(3)` : (images F. Morain, X)



► Plus d'infos sur la pile d'exécution en Programmation Avancée (ima S6).