

# Programmation avancée

## Recursive

Walter Rudametkin

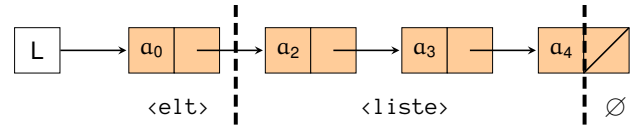
Walter.Rudametkin@polytech-lille.fr  
https://rudametw.github.io/teaching/

Bureau F011  
Polytech'Lille

11 février 2016

1/20

## La récursivité: Liste chaînée



- Structure de données récursive :  
 $\langle \text{liste} \rangle ::= \langle \text{elt} \rangle \langle \text{liste} \rangle \mid \emptyset$

### Déclaration

```
type Liste = pointeur de Cellule
type Cellule = structure
  valeur : <T>
  suivant : Liste
fin
```

Récursivité croisée  
(ou indirecte)

2/20

## La récursivité

- Une entité (SD, algorithme) est récursive si elle se définit à partir d'elle même
- Algorithmes récursifs (exemple : factorielle, fibonacci)

### Exemple d'algo récursive: Factorielle

- Analyse récurrente
  - $n! = n * (n - 1)!$
  - $0! = 1$
- Écriture fonctionnelle
  - $\text{fact}(n) = n * \text{fact}(n-1)$
  - $\text{fact}(0) = 1$

- Cas générale, récursif
- Cas primitif, terminal

3/20

## Factorielle

### Algorithme

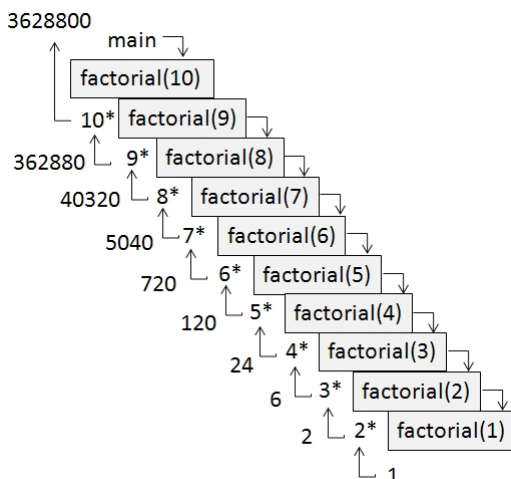
```
fonction fact(n) : entier
  D : n : entier
  L : f : entier
  si n = 0 alors
    f ← 1
  sinon
    f ← n * fact(n-1)
  fsi
  retourner(f)
ffonction
```

### Fonction en C

```
int fact (int n) {
  if (n==0)
    return 1;
  else
    return(n * fact(n-1));
}
```

4/20

## Exemple d'exécution d'une factorielle



5/20

## Conception récursive d'algorithmes

### 3 parties

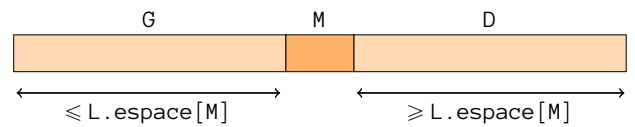
- Cas généraux récursifs:**  
Résolution du problème par lui même
- Cas terminaux non récursifs:**  
Résolution immédiate du problème)
- Conditions de terminaison**

6/20

## Exemple : Suite de Fibonacci

7/20

## Recherche dichotomique dans une liste contiguë: trouver l'élément x



- ▶ Dichotomie sur  $L.espace$
- ▶ **Cas général:**  $X \neq L.espace[M] \Rightarrow$  dichotomie à gauche ou à droite
- ▶ **Cas terminal :**  $X = L.espace[M]$
- ▶ **Condition de terminaison :**  $G > D$  (non trouvé)

8/20

## Recherche dichotomique: liste contiguë

```
Action Dichotomie(L,X,G,D,pos,existe)
  D : L : liste contiguë d'entiers
  X, G, D : entier
  R : pos: entier ; existe : booléen
  L : M : entier
  Si G>D Alors
    existe ← faux
  Sinon
    M ← (G + D) / 2
    Si X = L.espace[M] Alors
      existe ← vrai
      pos ← M
    Sinon
      Si X < L.espace[M] Alors
        dichotomie(L,X,G,M-1,pos,existe)
      Sinon
        dichotomie(L,X,M+1,D,pos,existe)
    Fsi
  Fsi
Fsi
Faction
```

9/20

## Récursivité sur les listes

### SD récurives $\Rightarrow$ algorithmes récurifs

- ▶  $\langle \text{liste} \rangle ::= \emptyset \mid \langle \text{elt} \rangle \langle \text{liste} \rangle$

où :

- ▶  $\emptyset \rightarrow$  cas terminal
- ▶  $\langle \text{elt} \rangle \rightarrow$  traitement de l'élément (éventuellement cas terminal)
- ▶  $\langle \text{liste} \rangle \rightarrow$  traitement récursif (cas général)

10/20

## Récursivité sur les listes

### Longueur d'une liste

- ▶  $L = \langle \text{elt} \rangle \langle \text{liste} \rangle$   
 $\text{longueur}(L) = 1 + \text{longueur}(L \uparrow \bullet \text{suivant})$
- ▶  $L = \emptyset$   
 $\text{longueur}(L) = 0$

### Algorithme

```
fonction longueur (L) : entier
  D : L : liste
  Si L = NULL Alors
    retourner(0)
  Sinon
    retourner(1 + longueur(L↑•suivant))
  Fsi
ffonction
```

11/20

## La récursivité : inverser() récursive

### Inverser une suite de caractères

- ▶  $s = \langle c_1, c_2, \dots, c_n, \bullet \rangle : \text{inverser} \langle c_n, \dots, c_2, c_1 \rangle$
- ▶ cas généraux et terminaux ? conditions de terminaison ?

### Algorithme

```
Action inverser()
  L : c : caractère
  lire(c)
  Si c ≠ '•' Alors
    inverser()
  écrire(c)
  Fsi
Faction
```

12/20

## La récursivité : inverser() itérative

- ▶ mémoriser les caractères lus séquentiellement
- ▶ les restituer en ordre inverse de leur mémorisation
- ▶  $\Rightarrow$  mémorisation en pile

### Algorithme

```
Action inverser()
  L: c : caractère, P : Pile de caractères
  lire(c)
  TQ c  $\neq$  '.'
    Faire empiler(P, c); lire(c);
  Fait
  {restituer en ordre inverse}
  TQ non pileVide(P) Faire
    dépiler(P,c) ; écrire(c);
  Fait
Faction
```

13/20

## La récursivité : pile d'exécution d'un langage

- ▶ Mémorise le contexte appelant lors d'un appel de fonction
- ▶ Restitue ce contexte lors du retour

### Exemple

```
void inverse(){
  char c;
  c = getchar();
  if (c != '.' ) {
    inverse(); putchar(c);
  }
}
```

14/20

## La récursivité : pile d'exécution d'un langage

### Schéma d'exécution

15/20

## La récursivité : conséquences

- ▶ Fournit une méthode pour traduire itérativement (à l'aide d'une pile) des algorithmes récursifs = la dérécursivisation
- ▶ Récursivité  $\Rightarrow$  surcoût dû à la pile
  - ▶ exemple : dichotomie, factorielle, longueur
  - ▶ nonexemple : inverser (en général pour une récursivité non terminale)
- ▶ Intérêt général quand elle facilite l'analyse algorithmique d'un problème (récursif par nature; ex : SD récursive)
- ▶ Intérêt pour la parallélisation des tâches

16/20

## La récursivité : insertion liste ordonnée

### Insertion de x dans une liste ordonnée

- ▶  $L = \emptyset \Rightarrow L = \langle x \rangle$
- ▶  $L = \langle elt \rangle \langle L' \rangle$ 
  - ▶  $x \leq \langle elt \rangle \Rightarrow L = \langle x, elt \rangle \langle L' \rangle$
  - ▶  $x > \langle elt \rangle \Rightarrow$  insérer x dans  $\langle L' \rangle$

17/20

## La récursivité : insertion liste ordonnée

```
Action insérer(L, x)
  D/R : L : liste de <T>
  D : x : <T>
  Si L =  $\emptyset$  Alors
    ajoutTête(L, x)
  Sinon
    Si  $x \leq L \uparrow \bullet \text{valeur}$  Alors
      ajoutTête(L, x)
    Sinon
      insérer( $L \uparrow \bullet \text{suivant}$ , x)
  Fsi
Fsi
Faction
```

18/20

## La récursivité : insertion liste ordonnée

```
1 void inserer(liste *pL, int x){
2     if ( (*pL == NULL) || (x <= (*pL)->valeur)
3         ajoutTête(pL, x);
4     else
5         inserer( &(*pL)->suivant, x);
6 }
7
8 void ajoutTête(liste *pL, int x){
9     Ptcellule P; allouer(&P);
10    P->valeur=x; P->suivant = *pL; *pL = P;
11 }
```

19/20

## La récursivité : insertion liste ordonnée

### Schéma d'exécution

20/20