



Université de Monastir

Cours: ASD2

Chapitre 4: La récursivité (version simplifiée)

Réalisé par:
Dr. Sakka Rouis

1



Chapitre 3: La récursivité

I. Définition et classifications

La récursivité est un outil très puissant en programmation. Lorsqu'elle est bien utilisée, elle rend la programmation plus facile. C'est avant tout une méthode de résolution de problème.

On distingue plusieurs types de récursivité :

➤ **récursivité directe** : lorsqu'un module fait appel à lui-même.

➤ **récursivité indirecte ou croisée** : lorsqu'un module A fait appel à un module B qui appelle A.

➤ **récursivité circulaire** : lorsqu'un module A fait appel à un module B, B fait appel à un module C qui appelle A.

2

Chapitre 3: La récursivité

I. Définition et classifications

Concernant les méthodes, on peut trouver d'autres classifications de récursivité :

récursivité non terminale : Une méthode récursive est dite non terminale si le résultat de l'appel récursif est utilisé pour réaliser un traitement (en plus du retour du module).

récursivité terminale : Une méthode récursive est dite terminale si aucun traitement n'est effectué à la remontée d'un appel récursif (sauf le retour du module).

3

Chapitre 3: La récursivité

I. Définition et classifications

Illustration

Cas 1 : récursivité directe	Cas 2 : récursivité indirecte
Procédure ProcRecursive (paramètres) Début ... ProcRecursive (valeurs) ... Fin proc	Procédure A (paramètres) Début ... B (valeurs) {appel de la procédure B dans A} ... Fin proc Procédure B (paramètres) Début ... A (valeurs) {appel de la procédure A dans B} ... Fin proc

4

Chapitre 3: La récursivité

I. Définition et classifications

Étude d'un exemple : la fonction factorielle

Dénotée par $n!$ (se lit factorielle n), c'est le produit de nombres entiers positifs de 1 à n inclus.

Exemples :

$$4! = 1 * 2 * 3 * 4,$$

$$5! = 1 * 2 * 3 * 4 * 5,$$

Noter que $4!$ peut s'écrire $4 * 3 * 2 * 1 = 4 * 3!$ et que $5!$ peut s'écrire $5 * 4 * 3 * 2 * 1 = 5 * 4!$

→ On peut conclure que : $n! = 1$ si ($n=1$)
 $n! = n * (n-1)!$ si non

5

Chapitre 3: La récursivité

I. Définition et classifications

Étude d'un exemple : la fonction factorielle (Solution non terminale)

Algorithmiquement	Traduction en C
Fonction Facto (n : Entier) : Entier Début Si ($n = 1$) Alors Facto $\leftarrow 1$ Sinon Facto $\leftarrow n * \text{Facto} (n-1)$ FinSi Fin Fn	int facto (int n) { if ($n == 1$) return 1 ; else return $n * \text{facto} (n-1)$; }

6

Chapitre 3: La récursivité

I. Définition et classifications

Étude d'un exemple : la fonction factorielle (Solution terminale)

Algorithmiquement	Traduction en C
Fonction Facto (n: entier, resultat: entier): entier Début Si (n=1) alors retourne resultat ; Sinon retourne Facto (n-1, n* resultat); Fin Fn	<pre>int facto (int n, int res) { if (n == 1) return res ; else return facto (n-1, n*res) ; }</pre>

7

Chapitre 3: La récursivité

II. Mécanisme de fonctionnement de la récursivité

Chaque cas est réduit à un cas plus simple. Le calcul de 4! se ramène à celui de 3!, le calcul de 3! se ramène au calcul de 2! ... jusqu'à arriver à 1! qui donne directement 1.

Après on fait un retour arrière. Le résultat d'une ligne i sert au calcul de la ligne i-1.

Illustration : Considérons le calcul de 4! par la fonction récursive définie ci-dessus :

Facto(4) renvoie $4 * \text{Facto}(3)$

 Facto(3) renvoie $3 * \text{Facto}(2)$

 Facto(2) renvoie $2 * \text{Facto}(1)$

 Facto(1) renvoie 1 (arrêt de la récursivité)

 Facto(2) renvoie $2 * 1 = 2$

 Facto(3) renvoie $3 * 2 = 6$

 Facto(4) renvoie $4 * 6 = 24$

8

Chapitre 3: La récursivité

III. Conseils d'écriture d'une méthode récursive :

Ces conseils sont illustrés par l'exemple suivant :

Écrire une fonction récursive permettant de calculer la somme des chiffres d'un entier n positif

Exemple : n = 528, donc la somme des chiffres de n est 15

1. Observer le problème afin de :

a) **Paramétriser le problème** : on détermine les éléments dont dépend la solution et qui caractérisent la taille du problème.

b) **Décrire la condition d'arrêt** : quand peut-on trouver "facilement" la solution ? (**une solution triviale**) :

Si on a le choix entre n = 528 et n = 8, il est certain qu'on choisit n = 8. La somme des chiffres de 8 est 8.

→**Conclusion** : Si n a un seul chiffre, on arrête. La somme des chiffres est n lui-même.

9

Chapitre 3: La récursivité

III. Conseils d'écriture d'une méthode récursive :

c) **réduire le problème à un problème d'ordre inférieur** pour que la condition d'arrêt soit atteinte un moment donné : somChif (528) =8+somChif(52)

$$= 8 + (2 + \text{somChif}(5)) = 8 + (2 + 5)$$

2. Écriture de la fonction :

Fonction somChif (n : entier) : entier

Début

Si (n < 10) **alors** { condition d'arrêt }
 somChif ← n;

Sinon { réduction du problème : }
 somChif ← n **mod** (10) + somChif (n **div** (10));

FinSi

Fin Fn

10

Chapitre 3: La récursivité

III. Conseils d'écriture d'une méthode récursive :

Exercice :

Illustrer les conseils précédents pour écrire une fonction récursive qui permet de calculer le produit de deux entiers positifs a et b sans utiliser l'opérateur de multiplication (*).

Solution :

a) la solution de ce problème dépend des deux opérandes n1 et n2

b) Si vous avez le choix entre : 12×456 , 12×0 , 12×1 Lesquels des trois calculs sont le plus simple ?

$$\begin{aligned} c) 12 \times 9 &= 12 + 12 + 12 + \dots + 12 && (9 \text{ fois}) \\ &= 12 + (12 + 12 + \dots + 12) && (8 \text{ fois}) \\ &= 12 + 12 \times 8 \end{aligned}$$

etc ...

11

Chapitre 3: La récursivité

IV. Exercices d'application

Exercice 2: Récursivité simple

Écrire **une fonction récursive** qui calcule les valeurs de polynôme d'Hermite $H_n(x)$ définie comme suit :

$$H_0(x) = 1 ;$$

$$H_1(x) = 2*x ;$$

$$H_n(x) = 2*x*H_{n-1}(x) - 2*(n-1)*H_{n-2}(x) \quad \text{pour tout } n > 1$$

Indication cette fonction possède deux paramètres **n** et **x**

Exercice 3: Récursivité Simple avec appelle imbriquée:

Écrire **une fonction récursive** qui calcule les valeurs de la fonction d'Ackermann, définie comme suit :

$$A(m, n) = \begin{cases} n+1 & \text{si } m=0 \\ A(m-1, 1) & \text{si } m > 0 \text{ et } n=0 \\ A(m-1, A(m, n-1)) & \text{sinon} \end{cases}$$

12

Chapitre 3: La récursivité

IV. Exercices d'application

Exercice 4: Récursivité simple

Ecrire une fonction PGCD_Euc qui retourne le PGCD de 2 entiers a et b en utilisant l'algorithme d'Euclide :

L'algorithme d'Euclide consiste à répéter plusieurs fois le traitement : $\text{PGCD}(a,b) = \text{PGCD}(b,a \bmod b)$

jusqu'à obtenir $\text{PGCD}(x,0)$. Le PGCD est alors x.

Exemple : $\text{PGCD}(36,16) = \text{PGCD}(16,4) = \text{PGCD}(4,0) = 4$.

Exercice 5: Récursivité croisée

Écrire deux fonctions C ou Java qui permettent de calculer les nèmes (n passé en argument) termes des suites entières U_n et V_n définies ci-dessous.

$$\begin{cases} U_0=1 \\ U_n=V_{n-1}+n \end{cases}$$

$$\begin{cases} V_0=0 \\ V_n=2*U_{n-1} + n \end{cases}$$

13

Chapitre 3: La récursivité

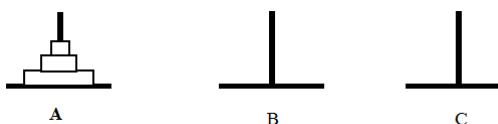
IV. Exercices d'application

Exercice 6: Tours de Hanoi

Le problème des tours de Hanoi est un grand classique de la récursivité car la solution itérative est relativement complexe. On dispose de 3 tours appelées A, B et C. La tour A contient n disques empilés par ordre de taille décroissante qu'on veut déplacer sur la tour B dans le même ordre en respectant les contraintes suivantes :

- On ne peut déplacer qu'un disque à la fois
- On ne peut empiler un disque que sur un disque plus grand ou sur une tour vide.

Illustration



14

Chapitre 3: La récursivité

IV. Exercices d'application

1) Observation

a) Ainsi, le paramétrage de la procédure déplacer sera le suivant :

Procédure déplacer(*n* : Entier ; A, B, C : Caractère)

b) Lorsque la tour A ne contient qu'un seul disque (*n*=1), la solution est évidente :

il s'agit de réaliser un transfert de la tour A vers B. Ce cas constitue donc la condition de sortie

c) Ainsi, pour déplacer *n* (*n*>1) disques de A vers B en utilisant éventuellement C, il faut :

- 1- déplacer (*n*-1) disques de A vers C en utilisant éventuellement
- 2- réaliser un transfert du disque de A sur B
- 3- déplacer (*n*-1) disques de C vers B en utilisant éventuellement A.

2) Écriture de la procédure :

15