Chapitre 2: La récursivité

Objectif :

Savoir résoudre des problèmes récurrents.

I. Définition & classification

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.

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).

Illustration algorithmique :

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**

II. Etude 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

A. Solution non terminale

Algorithmiquement	Traduction en C

Fonction Facto (n : Entier) : Entier
Début
Si (n = 1) **Alors**
Facto ← 1
Sinon
Facto← n * Facto (n-1)
FinSi
Fin Fn
	int facto (int n)
	{
	if (n == 1)
	return 1 ;
	else
	return n*facto (n-1) ;
	}

B. Solution terminale

Algorithmiquement	Traduction en C

Fonction Facto (n: entier, resultat: entier): entier
Début
Si (n=1) **alors**
Facto ← **resultat;**
Sinon
Facto ← Facto (n-1, n* resultat);
Fin Fn
	int facto (int n, int res)
	{
	if (n == 1)
	return res ;
	else
	return
	facto (n-1, n*res) ;
	}

III. 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 0! qui donne directement 1. Après quoi, 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 * Facto(3)

Facto(3) renvoie 3 * Facto(2)

Facto(2) renvoie 2 * 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

IV. Conseils d'écriture d'une fonction 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, la somme des chiffres de n est 15

1. Observer le problème afin de :

a) Paramétrage du 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.

c) de 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 + somChif(5))
<===> 8 + 2 + 5

2. Écriture de la fonction :

```
Fonction somChif (n : entier ) : entier
Debut
  Si ( n < 10 ) alors        /* condition d'arrêt */
       somChif ←n;
  Sinon                      /* réduction du problème : */
       somChif ← n mod (10) + somChif ( n div ( 10) ) ;
  FinSi
Fin Fn
```

V. Exercices d'application

Exercice1 :

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

Solution :

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

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

c) 12 x 9 = 12 + 12 + 12 + ... + 12 (9 fois)
= 12 + (12 + 12 + ... + 12) (8 fois)
= 12 + 12 x 8

etc ...

Exercice 2: Récursivité simple

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

$H_0(x) = 1$;
$H_1(x) = 2*x$;
$H_n(x) = 2*x*H_{n-1}(x) - 2*(n-1)*H_{n-2}(x)$ 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 :

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

Exercice 5: Récursivité croisée

Écrire deux fonctions qui permettent de calculer les $n^{èmes}$ (n passé en argument) termes des suites entières U_n et V_n définies ci-dessous.

$U_0=1$ $\quad$ $V_0=0$

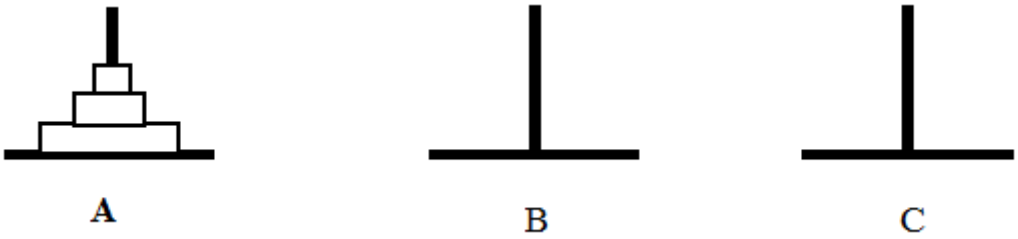
$U_n= V_{n-1}+n$ $\quad$ $V_n= 2*U_{n-1}+n$

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



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 :