Support de cours
Module : ASD1
Chapitre 5: La récursivité

Réalisé par:

Dr. Sakka Rouis Taoufik

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.

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

II. 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 ou n=0)**

n! = n* (n-1) ! si non

II. Étude d'un exemple : la fonction factorielle

```
unsigned facto (unsigned n) {
      if (n == 1 || n == 0)
            return 1 ;
      else
            return n*facto (n-1) ;
}
```

Illustration

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

II. Étude d'un exemple : la fonction factorielle

➔**Illustration du mécanisme de fonctionnement:**

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

III. 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, donc la somme des chiffres de n est 15

1. Observer le problème afin de :

a) Paramétrer 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.

III. Conseils d'écriture d'une fonction 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 + 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
```

III. Conseils d'écriture d'une fonction 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 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 ...

IV. Exercices d'application

Exercice 1: Récursivité simple

Soit la suite numérique Un suivante : Si n = 0 alors U0 = 4 sinon si n > 0 alors Un =2*Un-1 + 9
Écrire une fonction C qui calcul le terme Un.

Exercice 2: Récursivité croisée

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

$$\begin{cases} U_0 = 1 \\ U_n = V_{n-1} + 1 \end{cases} \qquad \begin{cases} V_0 = 0 \\ V_n = 2 * U_{n-1} \end{cases}$$

IV. Exercices d'application

Exercice 3: Récursivité simple

Ecrire une méthode récursive qui retourne la somme des carrés des x premiers entiers si x > 0 ; -1 sinon.

$$sommeCarre(x) = \begin{cases} \sum_{i=1}^{x} i^2 & \text{si } x \geq 0 \\ -1 & \text{si } x < 0 \end{cases}$$

Exemple : on prend x = 4, le résultat retournera la valeur 30.

IV. Exercices d'application

Exercice 4: Récursivité terminale

Ecrire une fonction récursive (basée sur l'algorithme d'Euclide) permettant de vérifier si a est un diviseur de b.

Algorithme Récursif	Algorithme Itératif

```
Fonction Diviseur (a,b) : Bool
Si (a <=0) Alors
   Retourner(Faux)
Sinon
   Si (a>=b) Retourner (a=b)
   Sinon
   Retourner (Diviseur (a,b-a))
   Fin Si
Fin Si
Fin
```

IV. Exercices d'application

Exercice 5:

En désire implementer une fonction permettant de calculer le PGCD de deux nombres naturels non nul (a et b) en utilisant l'algorithme d'Euclide.

1) Proposer une version itérative
2) Proposer une version récursive non terminale
3) Proposer une version récursive terminale

$$PGCD(a,b) = \begin{cases} a & \text{si } a=b \\ PGCD(a-b\ ,\ b) & \text{si } a>b \\ PGCD(a\ ,\ b-a) & \text{si } b>a \end{cases}$$