Cours: Algorithmes et Structures des Données 2 & Complexité

Chapitre 1: La récursivité

https://github.com/srtaoufik/Algo2_CD_Poly

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

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).

I. Définition et Classifications

NB. Un algorithme est dit récursif terminal s'il ne contient aucun traitement après un appel récursif.

Procédure **ALGO (X) // Liste des paramètres**
Si (**condition) alors**
<Traitement 1> // Traitement de terminaison (dépendant de X)
Sinon //cas générale: condition portant sur X
<Traitement 2> // traitement de base de l'algorithme (dépendant de X)
ALGO (F(X)) **// F(X) représente la transformation des paramètres**
// Rien !!!!! pas de code après l'appel récursive
Finsi
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.

<u>Exemples :</u>
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

A. Solution non terminale

```
unsigned facto (unsigned n) {
    if (n == 1)
        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

A. Solution non terminale

➔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
```

II. Étude d'un exemple : la fonction factorielle

B. Solution terminale

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


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

3. Traduction en C:

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.

$$\begin{cases} U0=1 \\ Un=Vn_{-1}+1 \end{cases} \qquad \begin{cases} V0=0 \\ Vn=2*Un_{-1} \end{cases}$$

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.

Chapitre 1: La récursivité

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

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

Chapitre 1: La récursivité

IV. Exercices d'application

Exercice 6

Écrire une fonction récursive qui permet de calculer le nombre d'occurrence d'un entier dans un tableau T.

Exercice 7

Écrire une fonction récursive qui permet de chercher le maximum dans un tableau T.

IV. Exercices d'application

Exercice 8

La suite de Fibonacci est définie mathématiquement par la formule si dessous :

Fib(n)=1 si n=0 ou n=1

Fib(n)=Fib(n-1)+Fib(n-2) si non (somme des deux derniers termes si n >= 2)

1) Proposer une solution récursive non terminale pour cette suite
2) Proposer une solution récursive terminale pour cette suite
3) Proposer une solution itérative pour cette suite

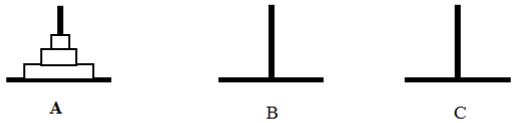
IV. Exercices d'application

Exercice 9: 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



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 :