La Récursivité

Rappel : Le principe de base

Un algorithme récursif est simplement un algorithme qui s'appelle lui-même, avec :

1. une ou plusieurs *conditions d'arrêt*, pour traiter les cas de base,
2. un ou plusieurs appels récursifs, qui permettent de résoudre le problème à traiter à partir d'appels sur des arguments plus petits.

Pour que l'algorithme termine, il faut veiller à ce que la suite des appels récursifs conduise toujours à un cas de base, géré par une condition d'arrêt.

- Écrivez une procédure récursive qui prend en argument un tableau d'entiers, et qui renverse ce tableau, c'est-à-dire que le premier élément devient le dernier, le deuxième élément devient l'avant-dernier, et ainsi de suite.

La suite de Fibonacci

La suite de Fibonacci est définie de la manière suivante :

$$\begin{cases} F_n &= F_{n-1} + F_{n-2} \\ F_0 &= F_1 = 1 \end{cases}$$

- Programmez trois versions du calcul du terme de rang n de la suite de Fibonacci :
  - récursive non terminale
  - itérative
  - récursive terminale
- Afin de valider ces trois versions, testez chacune sur des valeurs $n = 1..9$. Effectuez également ce test (pour chacune séparement) sur les valeurs $n = 10, 20, 30, 40, 50, 60, 70, 80, 90$. Que constatez vous?
- Modifiez la fonction récursive non terminale de manière à afficher la valeur de l'argument N pour chaque appel récursif, ainsi que le nombre total d'appels récursifs. Que constatez-vous ?

Récursivité terminale

$$\begin{cases} f(0,m) &= m \\ f(n+1,m) &= f(n,m)+1 \end{cases}$$

$$\begin{cases} g(0,m) &= m \\ g(n+1,m) &= g(n,m+1) \end{cases}$$

- Que calculent les fonctions f et g ? (si vous ne trouvez pas, vous pouvez les implémenter et les tester : le résultat saute aux yeux)
- Simulez à la main le comportement de chacune des fonctions.

  Pour la fonction f, il faut construire une pile de hauteur n, pour stocker les valeurs $f(n-1,m)$, $f(n-2,m)$, ..., $f(0,m)$. Le calcul de la valeur du sommet, $f(0,m)$, se fait avec la condition d'arrêt $(n=0)$, puis on descend la pile jusqu'à $f(n-1,m)$ en appliquant l'opération $+1$ à chaque pas.

  *Pour la fonction g, l'appel récursif est la dernière opération effectuée. On parle de **récursivité terminale**. Dans ce cas, il n'est pas nécessaire de construire une pile pour stocker les valeurs intermédiaires. L'évaluation de $g(n,m)$ se fait avec la suite d'égalités $g(n-1,m+1) = g(n-2,m+2) = \ldots = g(0,m+n)$, en espace mémoire constant. La condition d'arrêt permet alors d'avoir directement le résultat, $n+m$.*

- On considère maintenant la fonction Y, qui utilise la fonction X:

```
type TABLEAU is array(POSITIVE range <>) of NATURAL;

function X(T: TABLEAU; E: INTEGER; I: NATURAL) return BOOLEAN is
begin
   if I > T'last then
     return False;
   else
      return T(I) = E or X(T, E, I+1);
   end if;
end X;

function Y(T: TABLEAU; E: INTEGER) return BOOLEAN is
begin
   return X(T,E,T'first);
end Y;
```

Que font les fonctions X et Y ? Réécrivez-les pour obtenir une fonction récursive terminale.

La fonction exponentielle

On se propose de définir une fonction exponentielle, x^n (x flottant ou entier, n naturel), sans utiliser la fonction prédéfinie d'ADA. Pour cela, deux formules de récurrence sont possibles :

$$(A)\ \begin{cases} \exp(x,\ 0\ ) &=& 1 \\ \exp(x, i+1) &=& \exp(x,i) \times x \qquad\qquad (i \geq 0) \end{cases}$$

$$(B)\ \begin{cases} \exp(x,\ 0\ ) &=& 1 \\ \exp(x,\ 2i\ ) &=& \exp(x \times x,\ i) \qquad\qquad (i > 0) \\ \exp(x, 2i+1) &=& \exp(x \times x,\ i) \times x \end{cases}$$

- Implémentez la fonction exponentielle pour chacune des deux récurrences (A) et (B) (et vérifiez que les deux formules donnent bien le même résultat). D'après vous, laquelle des deux récurrences est la plus efficace ?
- Modifiez les deux fonctions de manière à afficher le nombre d'appels récursifs. Quelle est la complexité pour (A), pour (B) ?
- Écrivez une fonction itérative qui calcule l'exponentielle de deux entiers en utilisant la décomposition (B).

Les tours de Hanoï

Au départ, tous les disques sont placés sur la tour (1), par taille croissante (le plus grand au dessous, le plus petit au dessus). Le but est de transférer tous les disques sur la tour (3) en respectant les règles suivantes :

1. à chaque étape, on peut déplacer un disque, et un seul
2. on ne peut placer un disque sur une tour que si tous les disques déjà empilés sur cette tour sont de taille supérieure

- Programmez une procédure (récursive) qui affiche dans l'ordre tous les déplacements qui doivent être effectués pour déplacer tous les disques de la tour (1) vers la tour (3) en respectant toutes les régles.

Fonction d'Ackermann

La fonction d'Ackermann est définie par les équations de récurrences suivantes :

$$\begin{cases} A(0,n) &=& n+1 \\ A(m+1,0) &=& A(m,1) \\ A(m+1,n+1) &=& A(m,A(m+1,n)) \end{cases}$$

- Ecrivez une fonction qui retourne le terme de rang (m,n) de la fonction.
- Testez en calculant $A(4,4)$. Que se passe-t-il ?