

## La Récursivité

### 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 fonction récursive qui calcule la factorielle d'un entier.
- 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 peut-être calculée par la fonction suivante :

```
function Fibonacci(N: NATURAL) return NATURAL is
begin
  if N<=1 then
    return N;
  else
    return Fibonacci(N-1)+Fibonacci(N-2);
  end if;
end Fibonacci;
```

- Modifiez la fonction 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 ?
- Ecrivez une fonction itérative, avec une boucle **for**, qui calcule la fonction de Fibonacci. Quelle est la complexité en espace, en temps ?

### 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  $f(n - 1, m)$ ,  $f(n - 2, m)$ ,  $\dots$ ,  $f(0, m)$ . 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  $g(n - 1, m + 1) = g(n - 2, m + 2) = \dots = g(0, m + n)$ , 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,  $x^y$ , sans utiliser la fonction prédéfinie d'ADA. Pour cela, deux formules de récurrence sont possibles :

$$(A) \begin{cases} \exp(x, 0) & = 1 \\ \exp(x, y + 1) & = \exp(x, y) * x \end{cases}$$

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

- 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).