Informatique – TP n° 2 – Récursivité

Une fonction ou une procédure récursive est une fonction ou une procédure qui contient un ou plusieurs appels à elle-même.
Par exemple, le calcul du terme de rang n d'une suite définie par une relation de récurrence $u_n = f(u_{n-1})$ peut se faire dans une fonction nommée `u` par l'instruction
`u :=f(u(n-1))`
Par exemple, pour calculer u_8, l'ordinateur doit calculer $f(u_7)$, et pour cela, doit calculer u_7, en faisant appel à la fonction u; il doit ainsi calculer $f(u_6)$, etc. Il faut espérer que le programmeur, comme le mathématicien, a bien fondé sa récurrence en donnant explicitement un terme (le terme initial), faute de quoi l'ordinateur empile les appels à la fonction u en cherchant à calculer des termes d'indice de plus en plus petit, jusqu'à $-\infty$, sans jamais s'arrêter.
Ainsi, **il ne faut pas oublier d'initialiser une fonction ou une procédure récursive,** ce qui se fait souvent à l'aide d'une structure conditionnelle. Par exemple, si la valeur initiale de la suite ci-dessus est $u_0 = 2$, on écrira :

```
function u(n:integer):real;
begin
  if n<0 then
    begin
      writeln('Erreur');
      halt;
    end;
  if n=0 then
    u:=2
  else
    u:=f(u(n-1));
end;
```

La première structure conditionnelle arrête le programme, grâce à l'instruction halt, en affichant un message d'erreur (car la suite n'est pas définie au rang demandé), la deuxième structure conditionnelle donne l'initialisation pour $n = 0$, ainsi que la relation de récurrence, pour $n > 0$. Remarquez qu'il n'est pas nécessaire d'imbriquer les structures conditionnelles l'une dans l'autre ici, car si la première condition est satisfaite, on sort du programme, et on ne passe pas par la deuxième structure.
Il est important de noter qu'**on peut toujours « dérécursiver » un algorithme récursif,** c'est-à-dire l'écrire à l'aide de boucles, sans appel récursif. Pour exemple, nous avons déjà eu l'occasion de calculer le n-ième terme d'une suite définie par récurrence à l'aide d'une boucle, sans appel récursif.
Néanmoins, cette dérécursification n'est pas toujours évidente, et nécesite parfois d'« empiler » en mémoire les résultats successifs obtenus afin de les réutiliser ultérieurement. Cette gestion des « piles » de résultats s'avère parfois difficile à programmer directement.

Exercice 1 –

1. Écrire une fonction récursive calculant le n-ième terme de la suite définie par $u_0 = 3$ et pour tout $n > 0$, $u_n = u_{n-1}^2 - 5$.
2. Est-il possible facilement d'écrire une fonction récursive calculant la plus petite valeur de n pour laquelle $u_n > 10000$, ainsi que la valeur u_n correspondant ?

On remarquera que la récursivité peut être intéressante pour les suites définies par une récurrence, pour calculer un terme d'indice donné à l'avance, mais pas pour gérer des conditions d'arrêt un peu plus compliquées.

Exercice 2 –

1. Écrire une fonction récursive calculant le n-ième terme de la suite de Fibonacci, définie par $F_0 = 0$, $F_1 = 1$ et pour tout $n \in \mathbb{N}$, $F_{n+2} = F_{n+1} + F_n$.
2. Testez votre fonction pour $n = 10$, $n = 20$, $n = 30$, $n = 40$, $n = 50$, $n = 60$.
3. Si l'ordinateur a du mal à faire ces derniers calculs :
   (a) Expliquez cette difficulté en déterminant le nombre d'opérations effectuées dans votre algorithme pour calculer F_n, et comparer au nombre d'opérations que vous effectueriez si vous calculiez F_n à la main.
   (b) Écrire une fonction récursive pour calculer F_n, ne faisant qu'un appel récursif, et comparer ses performances à celles de la première fonction.

Un des grands dangers de la récursivité est l'explosion du nombre d'appels. Si, pour calculer le terme u_n d'une suite définie par une relation de récurrence d'ordre 2, on fait deux appels récursifs pour le calcul de u_{n-1} et u_{n-2}, on fera, au rang précédent, 4 appels récursifs à u_{n-3} et u_{n-2} pour le calcul de u_{n-1} et à u_{n-4} et u_{n-3} pour le calcul de u_{n-2}. Vous observez que certains de ces calculs sont redondants : on effectue deux fois le calcul de u_{n-3} et on réeffectue le calcul de u_{n-2}, qu'on avait déjà demandé au rang supérieur pour le calcul de u_n.
On fera donc à peu près 2^n appels récursifs pour le calcul de u_n. On obtient donc un algorithme exponentiel, alors que le deuxième algorithme programmé est linéaire, ainsi que l'algorithme non récursif que nous avons déjà eu l'occasion d'implémenter (qui n'est autre que la dérécursification du deuxième algorithme de cette question).

Exercice 3 –

1. En remarquant que $y^n = (y^p)^2 * y^r$, où $n = 2p + r$, $r \in \{0, 1\}$, écrire une fonction récursive calculant, pour tout réel y et tout entier n, la valeur de y^n.
2. Calculer, en fonction de n, le nombre maximal d'opérations effectuées pour le calcul de y^n, et justifier le nom d'« exponentiation rapide » donné à cet algorithme.
3. Modifier cette fonction afin de l'adapter à l'exponentiation de matrices carrées d'ordre 2.
4. En se basant sur une relation matricielle satisfaite par $\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}$, $(F_n)_{n\in\mathbb{N}}$ étant la suite de Fibonacci, écrire une fonction calculant le n-ième terme F_n en temps logarithmique.

Exercice 4 – Fonction 91 de McCarthy – Écrire une fonction calculant $f(n)$, $n \in \mathbb{Z}$ où f est définie par

$$\begin{cases} f(n) = n - 10 & \text{si } n > 100 \\ f(n) = f(f(n+11)) & \text{si } n \leqslant 100 \end{cases}$$

Calculer f pour plusieurs valeurs (positives ou négatives) inférieures ou égales à 101 ; de même pour plusieurs valeurs supérieures à 101. Qu'observe-t-on ? Le démontrer...

Exercice 5 –

1. Justifier que la fonction f définie sur $\mathbb{R}_+^*$ par $f(x) = \mathrm{e}^x + \ln x$ est strictement croissante sur $\mathbb{R}_+^*$, et qu'elle admet un unique zéro x_0, appartenant à l'intervalle $]0, 1[$.
2. La méthode de dichotomie pour trouver les zéros d'une fonction consiste à diviser l'intervalle en 2, et à déterminer dans quelle moitié se trouver le zéro de f, en étudiant le signe de f au milieu de l'intervalle, puis à recommencer la même opération sur cette moitié d'intervalle, jusqu'à obtenir la précision souhaitée. Écrire une fonction récursive déterminant une valeur approchée de x_0 à 10^{-10} près, par dichotomie.

Exercice 6 –

1. Soit $n > m$, et r le reste de la division euclidienne de n par m. Montrer que le pgcd de n et m est égal au pgcd de m et r.
2. En remarquant que si $r = 0$, le pgcd de n et m est n, écrire une fonction récursive calculant le pgcd de deux nombres, et justifier que cette fonction s'arrête toujours.

Exercice 7 – Calculer récursivement le coefficient binomial $\binom{n}{p}$:

1. en utilisant la relation $\binom{n}{p} = \frac{n}{p}\binom{n-1}{p-1}$;
2. en utilisant la formule de Pascal.

Comparer les performances de ces deux algorithmes, en expliquant les observations faites.