Université de Nice – Sophia Antipolis — Deug MASS MIAS SM 1re année
UEF1 : Informatique & Programmation — 2000–2001

TP N° 13

Récursivité

Ça va mieux en le révisant . . . Une méthode est dite *récursive* quand elle s'appelle elle-même. Quand on écrit une fonction récursive, il faut s'assurer que son exécution termine pour tout appel initial possible.

- il faut un (ou plusieurs) *cas d'arrêt*. Les instructions d'un cas d'arrêt sont *explicites*. Dans le cas d'une *fonction récursive*, ce sera la donnée explicite d'un résultat ; un cas d'arrêt ne comporte donc pas d'appel récursif.
- on veillera à ce que les jeux de variables distincts qui accompagnent des appels récursifs successifs doivent *converger* vers les conditions d'un cas d'arrêt. Dans la factorielle par exemple, la valeur de l'argument n se rapproche à chaque appel du cas d'arrêt $n <= 0$.

Jusqu'à présent, on savait *boucler* de façon *itérative*, c'est-à-dire en utilisant les instructions `for`, `while` ou `do...while`. Maintenant, on va apprendre à *boucler* avec la récursivité.

Les implémentations utilisant la récursivité ont souvent un code très court et concis. Néanmoins, l'efficacité n'est pas forcémenent l'apanage de la récursivité. Dès lors que la récursivité conduit à faire de multiples fois des calculs identiques, c'est très une mauvaise idée que de l'employer.

1 Méthodes récursives

Exercice 1) La suite de Fibonnaci est définie par la relation :

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}$$

Il est tentant de programmer une méthode récursive `fib`$(...)$ qui à un entier positif `n`, associerait `fib(n)`. Pourtant, une telle méthode est des plus inefficaces. Pour de grandes valeurs de `n`, l'exécution serait d'une lenteur inouïe.

1. Ecrivez un programme qui engendre et affiche le n^{e} nombre de Fibonacci, en utilisant une méthode récursive (mauvaise donc).
2. Imaginez à la main ce que serait l'appel à `fib(5)` et comprenez à quel point cette méthode est inefficace.
3. Retrouvez dans un précédent TP une autre implémentation -itérative- du calcul du n^{e} nombre de Fibonacci. Comparez les temps d'exécution de ces deux méthodes pour les valeurs 25, 30 et 35.

Exercice 2) On se propose d'écrire une version récursive de la recherche dichotomique d'un élément dans un tableau trié.

1. Dans la méthode `main`(...) de votre classe, construisez et faites afficher un tableau trié T comportant 20 entiers pseudo-aléatoires (à la manière de l'exercice 1 du TP n^o 12).
2. Demandez à l'utilisateur un entier `valeur` entre 0 et 199 à rechercher dans ce tableau trié. On complètera la méthode `main`(...) plus tard.
3. Recopiez après la méthode `main`(...) la méthode récursive `rechDichotomique`(...) suivante :

```
static int rechDichotomique(int [] T, int iDébut, int iFin, int valeur)
{  if (iDébut > iFin)
      return -1 ;     //   valeur ne figure pas dans le tableau
   int iMilieu = (iDébut + iFin) / 2 ;
   int différence  = T[iMilieu] - valeur ;
   if (différence == 0)
      return iMilieu ;
   else if (différence < 0)
      return rechDichotomique (T, iMilieu + 1, iFin, valeur) ;
   else
      return rechDichotomique (T, iDébut, iMilieu - 1, valeur) ;
}
```

4. Quel est l'appel initial à la méthode `rechDichotomique`(...) qui va lancer la recherche de l'entier `valeur` dans tout le tableau ? Ajoutez-le à la méthode `main`(...) puis achevez de la compléter.

Exercice 3) On se propose de compter le nombre de voyelles d'un mot, de façon récursive.

1. Ecrivez ensuite une méthode `estVoyelle`(...) permettant de savoir si un caractère est ou non une voyelle. Voici l'entête de cette méthode :

```
static boolean estVoyelle(char lettre)
```

2. Ecrivez une méthode récursive `nombreVoyelles`(...) qui prend en argument une chaîne de caractères. Dans le cas où la chaîne passée en paramètres est réduite à la chaîne vide, on arrête la récursion. Dans le cas où la chaîne a au moins une lettre, on regarde si sa première lettre est une voyelle ou pas et, dans chaque cas, on réalise un appel récursif adéquat à `nombreVoyelle`(...).

2 Approfondissement

Exercice 4) Ensembles de Cantor Les ensembles de Cantor à une dimension sont des courbes fractales, c'est-à-dire des dessins intrinsèquement récursifs. Ils sont bâtis selon le scénario suivant :

- le premier ensemble E_0 est un trait
- pour obtenir E_n à partir de E_{n-1}, on divise en trois chaque segment de droite de E_{n-1}, puis on fait disparaître chacun des sous-segments obtenus correspondant au tiers central

Ecrivez une applet de façon à dessiner les 6 premiers ensembles de Cantor.