Algorithmique et programmation 2 : Récursivité
Corrigé

Résumé

Ce document décrit l'écriture dans le langage C des éléments vus en algorithmique.

Table des matières

Liste des exercices

1 Définition

Définition : Un sous-programme récusif est un sous-programme dont l'implémentation contient un appel à lui-même.
Exemple : Fonction récursive qui calcule la factorielle :

```
1  Fonction factorielle(n: in Entier): Entier Est
2          -- Factorielle de n
3          --
4          -- Nécessite :
5          --      n >= 0
6      Début
7          Si n <= 1 Alors
8              Résultat <- 1
9          Sinon
10             Résultat <- n * factorielle(n-1)
11         FinSi
12     Fin
```

Remarque : `factorielle(n-1)` est l'appel récursif.

2 Fondement mathématique

Le corps de la fonction `factorielle` précédente correspond à la définition mathématique de la factorielle donnée sous forme de récurrence :

$$n! = \begin{cases} 1 & \text{si } n = 0 \text{ ou } n = 1 \\ n \times (n-1)! & \text{sinon} \end{cases}$$

Remarque : On pourrait mathématiquement utiliser la notation suivante pour définir la factorielle :

$$n! = \prod_{i=1}^{i=n} i = 1 \times 2 \times 3 \ldots \times (n-1) \times n$$

Elle conduirait à un sous-programme non récursif utilisant une boucle **`Pour`**.
Remarque : Les définitions par récurrence (et donc les SP récursifs) sont souvent plus concises et claires que leur équivalent itératif.

3 Terminaison

Danger : Il faut s'assurer que les appels récursifs s'arrêtent pour garantir la terminaison du programme.
Régle : Un SP récursif doit toujours faire apparaître deux éléments :
- le cas de base où l'on sait écrire le code sans nouvel appel récursif ;
- le cas général dans le quel on fait des appels récursifs.

Terminaison : Les appels récursifs doivent porter sur un problème de taille strictement inférieure, supérieure à la taille du cas de base.
Attention : Cette condition est suffisante mais pas nécessaire !
Exemple : Dans le cas de la factorielle :
- on définit la taille du problème de `factorielle(n)` comme étant n ;
- le cas terminal correspond à $n \leq 1$ ($n = 0$ ou 1) où la factorielle est 1 ;
- dans le cas général ($n > 1$), on utilise l'appel récursif `factorielle(n-1)` de taille strictement inférieure ($n - 1 < n$).

4 Récursivité terminale

Définition : Une fonction est *récursive terminale* quand le résultat de l'appel initial est directement celui du dernier appel récursif.
Régle : Aucune opération n'est réalisée sur le retour d'un appel récursif.

```
1  Fonction factorielle(n: in Entier; p: in Entier): Entier Est
2      Début
3          Si n <= 1 Alors
4              Résultat <- p
5          Sinon
6              Résultat <- factorielle(n-1, n * p)
7          FinSi
8      Fin
```

Le calcul de la factorielle de 4 donne :

```
1  fact(4) =  f(4, 1) -> f(3, 4) -> f(2, 12) -> f(1, 24)
2       <--24--    <--24--    <--24--     <--24--
```

Remarque : On a : fact(4) = f(4, 1) = f(3, 4) = f(2, 12) = f(1, 24) = 24
Intérêt : Inutile de remonter les appels récursifs (gain de temps).
Cette propriété peut être exploitée par certains compilateurs.
Remarque : Toute fonction récursive terminale peut être réécrite simplement en utilisant une itération.

```
1  Fonction factorielle(a: in Entier): Entier Est
2      Variable
3          n: Entier       -- équivalent de a mais modifiable !
4      Début
5          n <- a
6          Résultat <- 1
7          TantQue n > 1 Faire
8              Résultat <- n * Résultat
9              n <- n - 1
10         FinTQ
11     Fin
```

Remarque : On constate que :
- **`Résultat`** est équivalent à p ;

– initialisée à 1, valeur à fournir lors du premier appel à factorielle ;
– le **TantQue** correspond au cas général.

5 Récursivité mutuelle

Définition : On dit que deux SP sont mutuellement récursifs si si chacun des deux appels l'autre (éventuellement indirectement).

Plus généralement, un ensemble de SP est mutuellement récursif si la relation « f appelle g » admet un cycle : f_1 appelle ...appelle f_n appelle f_1.

Exercice 1 La parité possède les propriétés suivantes :
– un nombre est pair s'il est nul ou si son prédecesseur est impair ;
– un nombre est impair s'il est non nul et si son prédecesseur est pair.
Écrire deux SP qui indiquent si un entier est pair ou impair.

```
1  Fonction pair(n: in Entier): Booléen
2          -- Est-ce que n est pair ?
3          -- Nécessite : n >= 0
4      Début
5          Si n = 0 Alors
6              Résultat <- VRAI
7          Sinon
8              Résultat <- impair(n - 1)
9          FinSi
10     Fin
11
12 Fonction impair(n: in Entier): Booléen
13         -- Est-ce que n est impair ?
14         -- Nécessite : n >= 0
15     Début
16         Si n = 0 Alors
17             Résultat <- FAUX
18         Sinon
19             Résultat <- pair(n - 1)
20         FinSi
21     Fin
```

6 Récursivité en C

Aucun changement par rapport à ce qui a été présenté en Algorithmique. Rappelons que c'est **return** qui indique la valeur calculée par une fonction.

Voici la définition de la fonction factorielle en C.

```
1  /* Factorielle de n.
2   *
3   * Nécessite :
4   *      n >= 0
```

```
5   */
6  int factorielle(int n)
7  {
8      if (n <= 1) {
9          return 1;
10     }
11     else {
12         return n * factorielle(n - 1);
13     }
14 }
```