Algorithmique fonctionnelle - Récursivité

Récursivité

Définition

Une fonction est dite **récursive** si elle s'appelle elle-même dans sa propre déinition

Correspondance mathématique

- Principe de récurrence
- Exemple : définition des entiers :
  - 0 est un entier
  - n est un entier, alors $n + 1$ est un entier

Récursivité

Exemple 1 : Factorielle

- n ! = n * (n-1) * (n-2) * .. * 2 * 1
- On constate : n * (n-1) !, et 1 ! = 1

Récursivité

Exemple 1 : Factorielle

- n ! = n * (n-1) * (n-2) * .. * 2 * 1
- On constate : n * (n-1) !, et 1 ! = 1

Factorielle en pseudo-code

```
Fonction facto(n : Entier) : Entier
Début
   Si (n ≤ 1) Alors
      Retourner 1
   Sinon
      Retourner n*(facto(n-1))
   FinSi
Fin
```

Récursivité

Exemple 2 : PGCD

Fonction pgcd(a : Entier, b : Entier) : Entier
Début
  Si (b = 0) **Alors**
    Retourner a
  Sinon
    Retourner pgcd(b,a mod b)
  FinSi
Fin

Récursivité

Exercice

Ecrire une fonction F qui fait la somme de tous les nombres pairs jusqu'à un nombre entier positif donné.

Exemple :

$F(7) = 0+2+4+6 = 12$

$F(14) = 0+2+4+6+8+10+12+14 = 56$

Récursivité

Représentation

$f(x) =$ Si $c(x)$ Alors
$g(x)$
Sinon
$h(f(s(x)), x)$
FinSi

où

- $c(x)$ est le **critère d'arrêt**, fonction de x, sa valeur est un booléen
- $g(x)$ est le **cas trivial**, en général, retourne une valeur constante lorsque $c(x)$ est vrai
- $s(x)$ est le **successeur** de x, fonction de x
- $h(f(s(x)), x)$ est l'**appel récursif**, fonction de x et de son successeur

Types de récursivité

Simple vs Multiple

- **simple** : chaque appel de la fonction implique, au plus, un nouvel appel récursif de la fonction
- **multiple** : chaque appel de la fonction implique, plusieurs appels récursifs de la fonction

Types de récursivité

Terminale vs Non terminale

- **non terminale** : l'appel récursif n'est pas la dernière instruction de la fonction
- **terminale** : l'appel récursif est la dernière instruction de la fonction

Récursivité terminale

$f(x) =$ Si $c(x)$ Alors
$g(x)$
Sinon
$f(s(x))$
FinSi

Problème de terminaison

- Une fonction récursive doit s'arrêter
- La condition d'arrêt doit **impérativement** prendre la valeur **vrai**

Pas si évident...

```
Fonction exemple1(n : Entier) : Entier
Début
  Si (n ≤ 0) Alors
    Retourner 1
  Sinon
    Retourner n + exemple1(n-1) + exemple1(n-2)
  FinSi
Fin
```

Problème de terminaison

Cet algorithme se termine-t-il ?

```
Fonction exemple2(n : Entier) : Entier
Début
   Si (n = 0) Alors
      Retourner 1
   Sinon
      Retourner n*exemple2(n-2)
   FinSi
Fin
```

Problème de terminaison

Cet algorithme se termine-t-il ?

```
Fonction exemple3(n : Entier) : Entier
Début
   Si estPremier(n) Alors
      Retourner n
   Sinon
      Retourner exemple3(n+2)
   FinSi
Fin
```

Problème de terminaison

Cet algorithme se termine-t-il ? (conjecture de Syracuse)

```
Fonction exemple4(n : Entier) : Entier
Début
   Si (n = 1) Alors
      Retourner 1
   Sinon Si ((n mod 2) = 0) Alors
      Retourner exemple4(n/2)
   Sinon
      Retourner exemple4(3*n+1)
   FinSi
Fin
```

Problème de terminaison

- La terminaison est un problème indécidable
- Il n'existe pas d'algorithme qui a pour donnée un programme et obtient comme résultat "vrai" ssi ce programme termine

Avantages et inconvénients de la récursivité

- très économique pour le programmeur :
  - simple et intuitive (élégant et compact)
  - proche d'une description mathématique (récurrence)
  - plus facile à analyser
  - bien adapté aux structures de données récursives
- très dispendieuse en ressources machine :
  - utilisation de pile pour conserver les informations relatives à la fonction en cours d'exécution
  - une solution : récursivité terminale (placer les opérations qui suivent l'appel récursif à l'intérieur de l'appel lui-même !)

Factorielle

Récursivité non terminale

Fonction facto(n : Entier) : Entier
Début
  Si (n $\leq$ 1) **Alors**
    Retourner 1
  Sinon
    Retourner n * facto(n-1)
  FinSi
Fin

Factorielle

Récursivité terminale

```
Fonction factoRT(n : Entier, acc : Entier) : Entier
Début
   Si (n ≤ 1) Alors
      Retourner acc
   Sinon
      Retourner factoRT(n-1, acc*n)
   FinSi
Fin

Fonction facto(n : Entier) : Entier
Début
   Retourner factoRT(n, 1)
Fin
```

Suite de Fibonacci

Exercice

La suite de Fibonacci est définie par :

$$\begin{cases} U_0 = 0,\ U_1 = 1 \\ \forall\ n \geq 2,\ U_{n+2} = U_{n+1} + U_n \end{cases}$$

Écrivez la suite de Fibonacci en utilisant un algorithme récursif non terminal et puis un algorithme récursif terminal.

Fibonacci non terminal

Solution

Fonction fibo (n : Entier) : Entier
Début
  Si ($n < 2$) **Alors**
    Retourner n
  Sinon
    Retourner fibo(n-1) + fibo(n-2)
  FinSi
Fin

Fibonacci terminal

Solution

```
Fonction fiboRT(n : Entier, valeur : Entier, suivant : Entier) : Entier
Début
   Si (n = 0) Alors
      Retourner valeur
   Sinon
      Retourner fiboRT(n-1, suivant, valeur+suivant)
   FinSi
Fin

Fonction fibo(n : Entier) : Entier
Début
   Retourner fiboRT(n, 0, 1)
Fin
```

Récursivité

Exercice

Ecrire une fonction récursive terminale (!) F qui fait la somme de tous les nombres pairs jusqu'à un nombre entier donné.
Exemple :

$F(7) = 0+2+4+6 = 12$

$F(14) = 0+2+4+6+8+10+12+14 = 56$

Récursivité

Exercice

Ecrire une fonction récursive terminale (!) F qui fait la somme de tous les nombres pairs jusqu'à un nombre entier donné.

Solution

```
Fonction FRT(n : Entier, m : Entier, acc : Entier) : Entier
Début
   Si (n<= 1) Alors
      Retourner acc
   Sinon
      Retourner FRT(n-2, m+2, acc+m)
   FinSi
Fin

Fonction F(n : Entier) : Entier
Début
   Retourner FRT(n, 2, 0)
Fin
```