SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

PCOMP(LU3IN032)
Année 2022/2023
Séance 1 de TD/TME

Objectifs

1. Comparaison des styles de programmation impérative et fonctionnelle
   - instructions & expressions
   - calculs en place et par copie
   - exemple sur les tableaux et arbres
2. Mélange des styles

Travaux Dirigés

Exercice 1 : Des classiques en fonctionnel : id, fact, fib, odd et even

Donner dans votre langage de prédilection (C, Java, OCaml, Python) les types des fonctions que vous définissez.

Q.1.1 Définir la fonction identité (celle qui rend son argument intact)

Q.1.2 Définir une fonction calculant la factorielle

Q.1.3 La fonction calculant le `n`-ième nombre de la suite de Fibonacci peut être ainsi définie pour tout nombre entier strictement positif :

$$\begin{cases} \texttt{fib}(1) & = & 1 \\ \texttt{fib}(2) & = & 1 \\ \texttt{fib}(n) & = & \texttt{fib}(n-2) + \texttt{fib}(n-1) \end{cases}$$

Q.1.4 Définissez une version de complexité linéaire pour le calcul de n-ième nombre de la suite de Fibonacci.

Q.1.5 Définir une fonction `even` rendant true si son argument est pair, false sinon, et une fonction `odd` rendant true si son argument est impair, false sinon, et en n'utilisant pour seule fonction arithmétique que le prédécesseur.

Exercice 2 : Crible d'Ératosthène

On cherche à réaliser un programme du crible d'Ératosthène qui calcule la suite des nombres premiers entre 2 et n. Donner le type des fonctions de mandées avant de les écrire.

Q.2.1 Écrire une fonction `interval` qui prend deux entiers tels que `interval n m` calcule la liste des nombres entiers de `n` à `m`.

Q.2.2 (filtrage) Écrire une fonction `filter_out` qui prend un prédicat `p` et une liste `l` et retourne une nouvelle liste des éléments de `l` qui ne satisfont pas le prédicat `p`.

Q.2.3 Écrire une fonction `is_multiple` qui prend deux entiers `m` et `x` et qui retourne `true` si m est un multiple de `x`.

Q.2.4 Écrire une fonction `remove_multiple_of` qui prend un entier `n` et une liste d'entiers `l` et retourne la liste `l` privée des multiples de n. une liste des éléments de `l` sans les multiples de `n`

Q.2.5 Écrire la fonction `sieve` qui prend un entier `max` et qui applique l'algorithme du crible d'Ératosthène avec comme valeur maximale `max`. Cet algorithme consiste à créer la liste des entiers de 2 à `max`, puis à supprimer successivement, pour chaque élément de cette liste, les multiples de cet élément. On s'arrêtera dès lors que le carré du plus petit élément de la liste résultante est supérieur à `max`.

Exercice 3 : Transposée d'une matrice carrée

On se donne le code OCaml suivant :

```
let itrans m = let l = Array.length m in
  for i=0 to l-1 do
    for j=i to l-1 do
      let v = m.(i).(j) in
        m.(i).(j) <- m.(j).(i);
        m.(j).(i) <- v
    done
  done
```

Q.3.1 Donner son type.

Q.3.2 Indiquer ce que retourne l'appel suivant ainsi que les modification mémoire effectuées :

```
let v = [| [| 1; 2; 4|]; [|3; 3; 3|]; [|9; 8; 7|] |] ;;
itrans v;;
```

Q.3.3 Ecrire `itrans` dans deux autres langages.

Q.3.4 Indiquer les principales différences de ces solutions.

Exercice 4 : Composition de calculs

On cherche à effectuer en style fonctionnel et en style impératif des compositions de calculs.

Q.4.1 Ecrire une fonction `map` qui prend une fonction `f` et une liste `l` et retourne la liste des applications de `f` aux éléments de `l`.

Q.4.2 Ecrire une procédure `map_i` qui prend une fonction `f` et un tableau `t` et modifie chaque élément `e` du tableau par `f(e)`.

Q.4.3 Ecrire les différents appels pour enchaîner les calculs suivants sur une liste ou un tableau d'éléments entre 2 et n donné : additionner 3 et multiplier par 5

Q.4.4 Ecrire une fonction `compose` qui prend deux calculs, une liste ou un tableau, et enchaîne les calculs. La tester sur l'exemple précédent.

Exercice 5 : Arbre binaire

Soit le type OCaml suivant pour les arbres binaires à étiquettes entières.

```
type int_tree =
| Nil
| Node of int * int_tree * int_tree
```

On donnera les types en OCaml mais vous pouvez utiliser un autre langage si vous le désirez. Après avoir défini le type int_tree, écrire les fonctions suivantes.

Q.5.1 Donnez une définition de ce type int_tree dans un autre langage. Que pouvez-vous en dire ?

Q.5.2 'size : int_tree -> int' renvoyant la taille d'un arbre, c'est-à-dire son nombre de nœuds.

Q.5.3 'depth : int_tree -> int' renvoyant la profondeur d'un arbre, c'est-à-dire la longueur de sa plus longue branche.

Q.5.4 'sum : int_tree -> int', qui renvoie la somme des étiquettes d'un arbre.

Q.5.5 'contains : int -> int_tree -> bool', telle que 'contains x a' renvoie 'true' si et seulement si l'un des noeuds de l'arbre 'a' est étiqueté par 'x'.

Q.5.6 'elements : int_tree -> int list' qui renvoie la liste des éléments présents dans l'arbre en les collectant de la gauche vers la droite (étiquettes du sous-arbre gauche, étiquette à la racine, étiquettes du sous-arbre droit). Sauriez-vous écrire une version sans utiliser la concaténation ('@' en OCaml) des listes ?

Travaux sur Machines Encadrés

Exercice 6 : Création de programmes exécutables

Q.6.1 (Ligne de commande) On reprend le code de la fonction `fib` du TD pour construire une commande qui prend un paramètre en entrée `./fib 5` et affiche le résultat de l'appel de `fib 5`. Vous le testerez dans les 4 langages suivants :

- en OCaml, utilisez ocamlc ou ocamlopt ; pour accéder aux arguments du programme, vous pouvez utiliser le tableau `Sys.argv`. Pour accéder à l'élément `n` de `Sys.argv`, il suffit d'écrire `Sys.argv.(n)`.
- en Java, vous récupérer les arguments de la ligne de commande dans l'argument de la méthode `main(String[] args)`. Pour lancer l'exécutable utilisez `java maclasse n`.
- En C vous récupérez aussi les arguments dans l'argument de la fonction `main(int argc, char **argv)`.
- En Python les arguments sont dans le tableau `sys.argv`.

Exercice 7 : Exponentiation

Le but de cet exercice est de définir dans plusieurs style la fonction puissance.

On considère deux algorithmes

- Algorithme Naïf : $x^{(n+1)} = x * x^n$
- Exponentiation rapide : si $n = 2 * p$, alors $x^n = (x^2)^p$ et si si $n = 2 * p + 1$, alors $x^n = x * (x^2)^p$

Q.7.1 Programmer en Python les deux algorithmes de façon récursive et récursive terminale. Tenter de faire exploser la pile d'exécution.

Q.7.2 Programmer en Java les deux algorithmes de façon récursive et récursive terminale. Tenter de faire exploser la pile d'exécution.

Q.7.3 Programmer en OCaml les deux algorithmes de façon récursive et récursive terminale. Tenter de faire exploser la pile d'exécution.

Q.7.4 Appliquer l'algorithme de récursivation pour obtenir une version itérative des versions récursives terminales des deux algorithmes. Vous programmerez en C.

Q.7.5 Écrire l'arbre de syntaxe des programmes implémentant l'exponentiation naïve récursive et récursive terminale. Identifier les parties de l'arbre qui correspondent à des expressions et les noeuds qui correspondent à des instructions. Décrire un algorithme qui prend l'arbre de syntaxe d'une fonction et qui détermine si elle est récursive terminale.

Exercice 8 : Distance de Manhantan

On reprend l'exemple du cours sur le calcul de la distance de Manhantan en se donnant le code OCaml suivant donné dans le *toplevel* OCaml (# est l'invite) :

```
# let vide = -1 ;;
val vide : int = -1
# let mur = min_int;;
val mur : int = -4611686018427387904
# let taille = 9 ;;
val taille : int = 9
# let monde = Array.make_matrix taille taille vide ;;
val monde : int array array =

# let norme x = (x + taille) mod taille ;;
val norme : int -> int = <fun>
# let rec dist (px,py) d m =
  let npx = norme px and npy = norme py in
  let v = m.(npx).(npy) in
    if v = vide || v > d then (
      m.(npx).(npy) <- d ;
       dist (px+1,py) (d+1) m ;
       dist (px-1,py) (d+1) m ;
       dist (px,py+1) (d+1) m ;
       dist (px,py-1) (d+1) m ) ;;
val dist :
 int * int -> int -> int array array -> unit = <fun>
# monde.(2).(4) <- mur ; monde.(3).(4) <- mur ;
  monde.(2).(5) <- mur ;;
- : unit = ()
# dist (2,3) 0 monde ;;
- : unit = ()
```

L'affichage de 'affiche monde' donne alors :

```
5 4 3 2 3 4 5 6 6
4 3 2 1 2 3 4 5 5
3 2 1 0 . . 5 5 4
4 3 2 1 . 5 6 6 5
5 4 3 2 3 4 5 6 6
6 5 4 3 4 5 6 7 7
7 6 5 4 5 6 7 8 8
7 6 5 4 5 6 7 8 8
6 5 4 3 4 5 6 7 7
```

Q.8.1 Ecrire ce programme dans un autre langage.

Q.8.2 Même question avec un code en récursif terminal :

```
# let  dist (px,py) d m =
   let q = Queue.create () in
   let rec aux (px,py,d) =
     let npx = norme px and npy = norme py in
     let v = m.(npx).(npy) in
     if v = vide || v > d then (
      m.(npx).(npy) <- d ;
       Queue.add (px+1,py,d+1) q ;
       Queue.add (px-1,py,d+1) q ;
       Queue.add (px,py+1,d+1) q ;
       Queue.add (px,py-1,d+1) q ) ;
     if (not(Queue.is_empty q)) then aux (Queue.take q)
   in
     aux (px,py,d) ;;
val dist :
 int * int -> int -> int array array -> unit = <fun>
```

Q.8.3 Comparer vos deux adaptations entre-elles et avec le code OCaml fourni.

Q.8.4 Modifier le programme pour prendre en argument la taille du monde et les coordonnées des murs.

Quelques Liens

- en OCaml
  - Documentation OCaml : http://caml.inria.fr/pub/docs/manual-ocaml/
  - Index des modules de la bibliothèque standard : http://caml.inria.fr/pub/docs/manual-ocaml/libref/
  - Développement d'Applications avec OCaml : https://www-apr.lip6.fr/~chaillou/Public/DA-OCAML/
- en Java
  - tutorial Oracle : https://docs.oracle.com/javase/tutorial/
  - cours L3 : https://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/2022/ue/LU3IN002-2022oct/
  - Mooc EPFL : https://www.coursera.org/learn/programmation-orientee-objet-java
- en C cours L2 : http://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/2020/ue/LU2IN018-2020oct/public/2I001-2018oct_poly.pdf
- en Python cours L1 : http://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/2021/ue/LU1IN001-2021oct/cours2020.pdf