SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

MPIL(3I008)
Année 2017/2018
Séance 4 de TD/TME

Objectifs

1. Programmation impérative
2. Bibliothèque graphique

Travaux Dirigés

Exercice 1 : Tableaux, listes référencées

Q.1.1 Redéfinissez vous-mêmes le type `'a ref`, la fonction `ref` et les opérateurs `:=` et `!`

Q.1.2 Écrire en style impératif les fonctions `array_sum` et `list_sum` (réalisant la somme d'un tableau (resp. d'une liste) d'entiers.

Exercice 2 : Streams

Q.2.1 Considérez le code suivant :

```
let in_channel = open_in "lines.txt" in
try
    while true do
    let line = input_line in_channel in
    (* do something with line *)
    done
with End_of_file ->
    close_in in_channel
```

Quels sont les inconvénients ? Écrire une fonction line_stream_of_channel qui utilise les streams.

Q.2.2 Écrire une version line_stream_of_string qui fait la même chose pour des chaines de caractères (chaque élément d'une stream sera une des lignes de la chaine). On pourra utiliser des fonctions du module Str. Donnez un exemple d'utilisation.

Q.2.3 Écrire une fonction qui génère le Stream des entiers naturels commençant à partir d'un entier donné i.

Q.2.4 Généralisez pour écrire une fonction range, similaire à celle de Python, qui génère un Stream de valeurs `start` inclus et `stop` exclus, avec un pas `step` donné.

Q.2.5 Écrire le combinateur de flux stream_filter :

```
val stream_filter : ('a -> bool) -> 'a Stream.t -> 'a
    Stream.t = <fun>
```

tel que le Stream de sortie ne comporte que les éléments pour lesquels la fonction en premier argument a renvoyé vrai.

Q.2.6 Le constructeur du Stream avec une valeur constante peut s'écrire :

```
  let const_stream k = Stream.from (fun _ -> Some k);;
val const_stream : 'a -> 'a Stream.t = <fun>
```

Comment écrire un constructeur de Stream pour obtenir la répétition d'une série de valeurs ?

```
val cycle : 'a list -> 'a Stream.t = <fun>
```

Travaux sur Machines Encadrés

A commencer en TD

Exercice 3 : Jeu de la vie

Nous allons implanter le célèbre *jeu de la vie* de Conway. Dans un tel jeu, le programme possède un quadrillage de cellules qui peuvent être mortes ou vivantes. En partant d'une configuration donnée (une valeur morte ou vivante pour chaque cellule), le jeu passe d'étape en étape en appliquant simultanément un ensemble de règles qui décident de la survie et de la naissance des cellules d'une étape à l'autre.

Q.3.1 Donnez un type pour représenter un tel quadrillage et donnez un exemple de situation initiale `init_gen`.

Q.3.2 Donnez la fonction `next_gen` qui calcule la génération suivante avec les règles de vie et de mort suivantes :

- Une cellule meurt d'isolement si elle a moins de deux voisins.
- Une cellule meurt d'étouffement si elle a plus de trois voisins.
- Une cellule naît si elle a exactement 3 voisins.

Vous donnerez aussi la fonction auxiliaire `neighbours` qui calcule le nombre de cellules voisines vivantes d'une cellule donnée.

Q.3.3 Pour avoir une représentation graphique de notre quadrillage nous allons utiliser le module Graphics de Ocaml.

Donnez la fonction `init_graph` ouvrant la fenêtre graphique à une taille correcte pour afficher un quadrillage donné. On pourra représenter les cellules par des petits carrés de taille choisie arbitrairement.

Q.3.4 Écrivez la fonction `draw_gen`.

Q.3.5 Donnez la fonction `continue` qui attend l'appui au clavier et renvoie false si l'utilisateur a tapé 'q' (pour quitter).

Q.3.6 Déduisez la boucle principale qui affiche successivement les étapes en attendant une touche entre chaque.

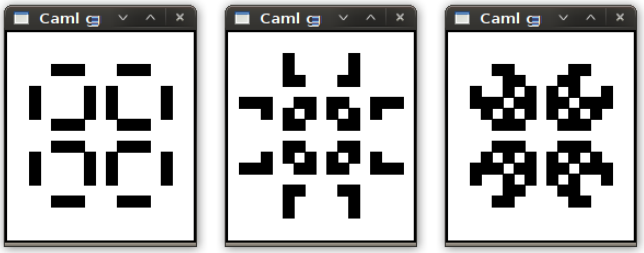

FIGURE 1 – Un oscillateur