Université Bordeaux 1 Licence S&TS Mention Informatique S5 2010/2011

UE INF353 Programmation 3

Programmation Fonctionnelle et Symbolique

Devoir surveillé No 2

Tous documents autorisés. Lundi 29 Novembre 2010 Durée : 1h20.
Le barème est donné à titre **indicatif**.

Exercice 1 *(4pts)*

On considère des listes de paires telles que pour chaque paire, le `car` est un entier et le `cdr` un élément quelconque. Une telle liste représente sous forme compressée une suite d'éléments. Par exemple, la liste de paires `((3 . A) (2 . B) (1 . A) (3 . C))` représente la suite `A A A B B A C C C`. On suppose qu'une suite d'éléments est stockée dans un vecteur (tableau à une dimension). Par exemple, la suite précédente est sera donnée par le vecteur `#(A A A B B A C C C)`.

1. Écrire une fonction **itérative** `uncompress-pairs (pairs)` qui prend en paramètre une liste de paires du type décrit ci-dessus et qui retourne dans un vecteur la suite résultant de la décompression. Exemples :

```
CL-USER> (defparameter *pairs* '((1 . 3) (4 . 2) (1 . 0) (2 . 1)))
*PAIRS*
CL-USER> (uncompress-pairs *pairs*)
#(3 2 2 2 2 0 1 1)
CL-USER> (uncompress-pairs '())
#()
```

2. Que retourne l'appel `(uncompress-pairs '((2 . 1) (3 . 0) (2 . 3)))` ?

Exercice 2 *(8pts)*

On considère des images noir et blanc de dimensions `h x w`. Une telle image sera représentée par un tableau contenant `h` lignes, chaque ligne étant elle-même représentée par un tableau de `w` entiers (`1` pour noir et `0` pour blanc). Par exemple, la variable `*sapin*` définie Figure 1 contient la représentation d'une image `8 x 9`.

1. Écrire une fonction **itérative** `print-image (image)` qui imprime sur la sortie standard l'image en imprimant un espace pour les bits à 0 et un `@` pour les bits à 1. Par exemple, l'image représentée dans la variable `*sapin*` doit s'imprimer comme montré Figure 1.
2. Que donne l'appel suivant ? `(print-image #(#(1 1 0) #(0 1 0) #(0 1 1)))`
3. Modifier la fonction `print-image` de manière à pouvoir passer dans un paramètre optionnel le caractère à afficher.

```
(defparameter *sapin*
  #(#(0 0 0 0 0 0 0 1 1)
    #(0 0 0 0 1 0 0 0 1)
    #(0 0 0 1 1 1 0 0 0)
    #(0 0 1 1 1 1 1 0 0)
    #(0 1 1 1 1 1 1 1 0)
    #(0 0 0 1 1 1 0 0 0)
    #(0 0 0 1 1 1 0 0 0)
    #(0 0 0 0 0 0 0 0 0)))
```

```
CL-USER> (print-image *sapin*)
        @@
       @   @
      @@@
     @@@@@
    @@@@@@@
      @@@
      @@@

NIL
```

FIGURE 1 – Représentations d'un sapin

En vue de compresser plus facilement une image `h x w`, on souhaite transformer sa représentation en un vecteur de taille `h x v` dans lequel les lignes sont stockées les unes à la suite des autres.

4. Écrire une fonction **itérative** `image2vector (image)` qui prend en paramètre une image représentée par un tableau de lignes et qui retourne l'image représentée dans un vecteur. Exemple :

```
CL-USER> (image2vector *sapin*)
#(0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 0 0
  0 1 1 1 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0)
```

Exercice 3 *(8pts)*

Soit la fonction `mystere-vector (v)` définie ci-dessous.

```
(defun mystere-vector (v)
  (let ((len (length v)))
    (when (zerop len)
      (return-from mystere-vector '()))
    (loop
      with pairs = (list (cons 1 (aref v 0))) ;; premier element vu une fois
      for i from 1 below len
      if (= (cdr (car pairs)) (aref v i)) ;; element courant vu une nouvelle fois
        do (incf (car (car pairs)))
      else
        do (push (cons 1 (aref v i)) pairs)
    finally (return (nreverse pairs)))))
```

1. Que retourne l'appel `(mystere-vector #())` ?
2. Que retourne l'appel `(mystere-vector #(2 2 2 0 0 1 1 1 2))` ?
3. Expliquer en une phrase ce que fait la fonction `mystere-vector`.
4. Écrire une nouvelle version de `mystere-vector` qui utilise une boucle de type `do`.

FIN