Programmation 3 : feuille 7

Structures de données

Exercice 7 .1 *tableaux*

Pour tester le générateur de nombres aléatoires de Lisp (accessible grâce à la fonction `random`), on veut imprimer un histogramme de m valeurs aléatoires comprises entre 0 et $n-1$ produites par le générateur. Pour compter le nombre de fois qu'une valeur est produite, on utilisera un tableau à n entrées. Voici un exemple d'histogramme produit pour les valeurs $m = 200$ et $n = 11$.

```
*  (histogram-print *histogram*)
 0 [ 20] ********************
 1 [ 20] ********************
 2 [ 21] *********************
 3 [ 19] *******************
 4 [ 13] *************
 5 [ 15] ***************
 6 [ 22] **********************
 7 [ 13] *************
 8 [ 17] *****************
 9 [ 17] *****************
10 [ 23] ************************
NIL
*
```

Exercice 7 .2 *Utilisation de variables spéciales*

1. Ouvrir un flot en écriture sur un fichier appelé par exemple `HISTOGRAMME` grâce à l'instruction

   ```
   CL-USER> (defparameter *flot* (open "HISTOGRAMME" :direction :output))
   *FLOT*
   ```

2. Faire des tests d'écriture sur ce flot à l'aide des instructions `print` et `format`.
   Écrire un histogramme sur le flot.
   Fermer le flot grâce à l'instruction `(close *flot*)`.
   Examiner le fichier `HISTOGRAMME` ainsi créé.

3. Simplifier la fonction `histogram-print` en supprimant le paramètre de flot optionnel et en la faisant écrire directement dans le flot `*standard-output*` ouvert sur la *sortie standard*.
   Comme faire pour imprimer l'histogramme dans un autre flot que celui mémorisé par `*standard-output*` ?
   Imprimer un histogramme dans un fichier.

Exercice 7 .3 *Tables de hachage*

Une couleur peut être considérée comme composée des couleurs rouge R, verte G et bleue B. Dans ce modèle, une couleur est représentée par trois entiers compris entre 0 et 255. La classe `color` contient des objets qui sont des couleurs. On peut créer une couleur avec `make-color`.

```
(defclass color ()
  ((r :initarg :r :reader red-of)
   (g :initarg :g :reader green-of)
   (b :initarg :b :reader blue-of)))

(defmethod print-object ((c color) stream)
  (format stream "couleur{red=~d, green=~d, blue=~d}"
          (red-of c)
          (green-of c)
          (blue-of c)))

(defun make-color (&key (r 0) (g 0) (b 0))
  (make-instance 'color :r r :g g :b b))

;; (make-color :r 255) ; rouge
```

Compiler le code ci-dessus.

1. Écrire un prédicat `warm-colorp` qui renvoie vrai si une couleur est une couleur chaude (taux de rouge supérieur au taux de bleu) et faux sinon.
2. Créer une table qui associe des noms de couleurs aux couleurs elles-mêmes. Utiliser une table de hachage dont la clé d'accès est le nom de la couleur recherchée. Utiliser la fonction `describe` pour examiner la table ainsi créée. Compléter la table en utilisant une combinaison de `setf` et `gethash`. Tester l'ajout d'un nouvel élément dont la clé est déjà présente. Tester la suppression d'un élément de la table avec `remhash`.
3. En utilisant la fonction `maphash`, écrire une fonction qui construit la liste des clés de la table des couleurs.
4. Construire des exemples de manipulation d'une table de couleurs en utilisant la macro `with-hash-table-iterator`. Utiliser la macro `multiple-value-bind` pour récupérer les valeurs retournées pas l'itérateur fourni dans `with-hash-table-iterator`.
5. Utiliser ces dernières notions pour écrire une fonction qui extrait le nom des couleurs chaudes contenues dans la table.