Département Licence	ANNÉE : 2005/2006 SESSION DE MAI 2006 **Parcours :** INF6 / MAI6 / MIAGE6 **Code UE :** INF207 **Épreuve :** Programmation Fonctionnelle et Symbolique **Date :** Mercredi 24 Mai 2006 **Heure :** 11h **Durée :** 1h30 Documents : autorisés Épreuve de Mme Irène Durand et Mr Pierre Castéran

Le barême est donné à titre **indicatif**

Exercice 1 *(5pts)*

Soit la fonction `mystere` suivante :

```
(defun mystere (size &key (initial-element 0) (suivant #'1+))
  (let ((element initial-element)
        (l '()))
    (dotimes (i size (nreverse l))
      (push element l)
      (setf element (funcall suivant element)))))
```

1. Que retourne l'appel `(mystere 4)` ?
2. Que retourne l'appel
   `(mystere 4 :initial-element 1 :suivant #'(lambda (x) (* 2 x)))` ?
3. Écrire une version récursive de cette fonction.

Exercice 2 *(4pts)*

Écrire une macro `c-while (test &body body)` similaire à la macro `while` vue en cours ou en TD mais qui au lieu de retourner `NIL` retourne le nombre d'itérations effectuées.
Par exemple,

```
(let ((n 6))
  (c-while (< n 10)
    (prin1 n)
    (incf n))))
```

affiche `6789` et retourne `4`.

Exercice 3 *(5pts)*

On considère le programme de Sudoku vu en cours. La vérification qu'un remplissage (partiel ou non) d'une grille respecte les règles de ce jeu est basée sur une interaction complexe entre les fonctions `set-digit`, `forbid-digit` et le créneau `to-fill`.
Dans une phase de mise au point du programme, on voudrait pouvoir vérifier que le remplissage (partiel ou total) d'une grille respecte les règles du jeu.

Écrire une fonction `check-coord(grid)` permettant cette vérification. On ne devra donc pas utiliser la valeur du créneau `to-fill`, dont la valeur n'est utile que lorsque la structure de données est correctement remplie.

Exercice 4 *(6pts)*

On considère le début d'implémentation d'un paquetage destiné à manipuler des figures géométriques.

```
(defclass polygon () ()
  (:documentation "abstract class for polygons"))

(defgeneric number-of-sides (p)
  (:documentation "returns the number of sides of a polygon"))

(defgeneric perimeter (p)
  (:documentation "returns the perimeter of a polygon"))

(defgeneric area (p)
  (:documentation "returns the area of a polygon"))

(defgeneric sides-lengths (p)
  (:documentation "returns the list of sides of a polygon"))

(defmethod perimeter ((p polygon))
  (reduce #'+ (sides-lengths p)))
```

On souhaite maintenant avoir des méthodes adaptées aux polygônes réguliers (côtés et angles égaux) pour lesquels on peut calculer simplement le périmètre ou l'aire. Un polygône régulier est défini entièrement par son nombre de côtés et par la longueur d'un côté.

1. Écrire une classe `regular` permettant de capturer la sous-classe des polygônes réguliers.
2. Écrire une fonction `make-regular-polygon (n side)` qui fabrique un polygône régulier à `n` côtés de longueur `side`.
3. Que manque-t'il pour qu'on puisse appeler la méthode `perimeter` standard pour tous les polygônes sur un object de la classe `regular` ?
4. Écrire une méthode `perimeter` spécifique pour la classe des polygônes réguliers.

On souhaite rajouter la possibilité d'avoir des figures géométriques colorées. On dispose d'une structure `color` permettant de représenter les couleurs.

```
(defstruct color (r 0) (g 0) (b 0))
(defconstant +black+ (make-color))
(defconstant +red+ (make-color :r 255))
```

5. Écrire une classe abstraite `colored-mixin` permettant de mémoriser et éventuellement changer la couleur d'un objet.
6. Quels changements faut-il apporter à la classe `polygon` pour transformer automatiquement les polygônes en polygônes colorés ?