Listes d'associations

Une autre structure de données

1

Définitions

- Une liste d'associations est une liste de couples (les associations)
- Chaque association est un couple (ou paire) formé par une clef et une valeur

$$`((clef_1.valeur_1)(clef_2.valeur_2)\ldots(clef_n.valeur_n))$$

- Cette structure de données permet de chercher une valeur à partir d'une clef

Primitives

- Rechercher dans la liste une association à partir d'une clef
- Rechercher dans la liste la valeur d'une association à partir d'une clef
- Ajouter une association à une liste d'association
- Enlever d'une liste les associations qui ont une clef donnée
- Remplacer dans la liste la valeur correspondant à une clef

Exemples d'utilisation (1)

```
(define vitesses '((tortue . 15) (levrier . 60)
                   (guepard . 115) (lapin . 60) (autruche . 90)))
```

```
(donne-paire vitesses 'lapin 'pas-trouve)
  → (lapin . 60)
(donne-paire vitesses 'souris 'pas-trouve)
  → pas-trouve
(donne-valeur vitesses 'autruche 'pas-trouve)
  → 90
```

Exemples d'utilisation (2)

(ajoute-paire vitesses 'souris 25)

→ ((souris . 25) (tortue . 15) (levrier . 60) (guepard . 115) (lapin . 60) (autruche . 90))

(remplace-valeur vitesses 'guepard 130)

→((tortue . 15) (levrier . 60) (guepard . 130) (lapin . 60) (autruche . 90))

(enleve-clef vitesses 'guepard)

→ ((tortue . 15) (levrier . 60) (lapin . 60) (autruche . 90))

Définition des primitives (1)

```
(define donne-paire ; → couple ou valeur par defaut
 (lambda (L  Clef  ValDef) ; alist * type clef * type quelconque
  (cond ((null? L) ValDef)
        ((eq? (caar L) Clef) (car L))
        (else (donne-paire (cdr L) Clef  ValDef)))))
```

N.B. Une fonction prédéfinie en scheme a presque le même comportement :

(assoc 'levrier vitesses) →(levrier . 60)

La valeur par défaut de la fonction assoc est #f

Définition des primitives (2)

```
(define donne-valeur ; → type valeur
  (lambda (L  Clef  ValDef)  ; alist * type clef * type quelconque
    (let ((res (donne-paire  L  Clef  ValDef))) ; res : paire
      (if (eq? res ValDef)
          ValDef
          (cdr res))))) ; cdr res : second élement de res
```

```
(define ajoute-paire ; → alist
  (lambda (L  Clef  Valeur) ; alist * type clef * type valeur
    (cons (cons Clef  Valeur) L)))
```


Définition des primitives (3)

```
(define enleve-clef ; → alist
  (lambda (L  Clef) ; alist * type clef
    (cond ((null? L) '())
          ((eq? (caar L) Clef) (enleve-clef (cdr L) Clef))
          (else (cons (car L) (enleve-clef (cdr L) Clef))))))
```

N.B. Cette fonction enlève de la liste d'association tous les couples de la Clef donnée

Définition des primitives (4)

```
(define remplace-valeur ; → alist
  (lambda (L   Clef   Valeur) ; alist * type clef * type valeur
    (cond ((null? L) (ajoute-paire '() Clef   Valeur))
          ((eq? Clef (caar L)) (ajoute-paire (cdr L) Clef   Valeur))
          (else (cons (car L) (remplace-valeur (cdr L) Clef   Valeur))))))
```


Un autre exemple

```
(define etudiants '((10302450 . (Jean Sauvert L2))
        (10405672 . (Sylvie Perrin L1)) (10304455 . (Aziz Bourras L1))))
```

(donne-valeur etudiants 10403302 'pas-trouve)

→ pas-trouve

(donne-valeur etudiants 10405672 'pas-trouve)

→ (sylvie perrin l1)

(remplace-valeur etudiants 10405672 '(sylvie perrin L2))

→ ((10302450 . (jean sauvert l2)) (10405672 . (sylvie perrin l2))
(10304455 . (aziz bourras l1)))