Programmation applicative – L2
TD 9 : Affectation

A. Chateau {annie.chateau@umontpellier.fr}
V. Boudet {vincent.boudet@umontpellier.fr}
H. Chahdi {hatim.chahdi@umontpellier.fr}

1 L'affectation en Scheme

En programmation impérative le concept de base n'est plus la fonction, comme en programmation fonctionnelle, mais *l'instruction*. Une instruction a pour but de réaliser un changement d'état en mémoire en modifiant les valeurs de certaines variables. L'instruction élémentaire est *l'affectation* (changement de la valeur d'une variable).

En mathématique la notion d'affectation n'existe pas, ainsi quand un programme génère des *effets de bord* sur les variables on ne parle plus de fonction mais de **procédure**.

Remarque — Dans une séquence d'instructions l'ordre dans lequel les instructions sont évaluées a une importance. En Scheme les sequences d'instructions sont réalisées à l'aide de de la forme spéciale `begin`.

1.1 La forme spéciale `set!`

En Scheme l'affectation est réalisée par la **forme spéciale** `set!`.
L'évaluation de `(set! var new-val)` ne renvoie pas de valeur et consiste à :

1. Rechercher la case mémoire qui stocke la valeur de `var` ;
2. Remplacer la valeur actuelle par `new-val`.

Par exemple :

```
> (define var 3)                    ⇒  rien
> var                               ⇒  3
> (set! var 5)                      ⇒  rien
> var                               ⇒  5
> (set! var (lambda (x) (* x x)))   ⇒  rien
> (var 3)                           ⇒  9
```

Remarque — Il est obligatoire que la variable sur laquelle on applique une affectation avec `set!` soit définie au préalable (par un `define`, `let`, `lambda`, ...).

Exercice 1 *Que dire des valeurs de* `u` *et* `v` *après la séquence d'instruction suivante :*

```
> (define u '(a b c))
> (define v (cdr u))
> (set! u (cons 'd u))
```

Exercice 2 *Écrivez une procédure* `(fact-affect n)` *qui calcule la factorielle de manière impérative : c'est-à-dire en utilisant des variables et l'affectation au lieu du passage de paramètre utilisé par les fonctions récursives.*

1.2 La forme spéciale `letrec`

L'affectation nous permet d'expliquer la forme spéciale `letrec` introduite au TD4.

De manière générale la forme spéciale `letrec` peut être ré-écrite à partir de `let` en utilisant `set!` :

```
(letrec ((⟨var1⟩⟨exp1⟩)
         (⟨var2⟩⟨exp2⟩)
         ...
         (⟨varn⟩⟨expn⟩))
   ⟨corps⟩)
```

⇔

```
(let ((⟨var1⟩ 'rien)
      ...
      (⟨varn⟩ 'rien))
   (begin
      (set! ⟨var1⟩⟨exp1⟩)
      ...
      (set! ⟨varn⟩⟨expn⟩)
      ⟨corps⟩))
```

Par exemple la définition d'une fonction locale pour calculer la factorielle peut être :

```
(define (fact n)
  (let ((fact-aux 'rien))
    (begin
       (set! fact-aux (lambda (x acc) (if (= x 0) acc (fact-aux (- x 1) (* x acc)))))
       (fact-aux n 1))))
```

2 Structure de donnée mutable

2.1 Les opérateurs de mutations `set-car!` et `set-cdr!`

Nous avons vu la forme spéciale `set!` qui permet de modifier le contenu d'une cellule. Il existe en Scheme 2 autres formes spéciales : `set-car!` et `set-cdr!` qui permettent de modifier respectivement le `car` et le `cdr` d'une paire. Par exemple :

```
> (define P (cons 1 2))  ⇒  rien
> P                      ⇒  (1 . 2)
> (set-car! P 3)         ⇒  rien
> P                      ⇒  (3 . 2)
> (set-cdr! P null)      ⇒  rien
> P                      ⇒  (1)
```

On dit que `set-car!` et `set-cdr!` sont les accesseurs en *écriture* de la structure de donnée paire (en opposition à accesseur en *lecture* pour `car` et `cdr`). De manière générale les interfaces

des structures de données complexes doivent donc aussi fournir les accesseurs en écriture sur les éléments de cette structure.

Exercice 3 *Définir un procédure* `(append-2! L1 L2)` *qui concatène* `L2` *à* `L1`*. La procédure ne doit pas renvoyer de valeur mais modifier la liste* `L1`*.*

Exercice 4 *Quelles sont les valeurs renvoyées par les expressions suivantes :*

```
> (define L1 '(1 2 3))
> (define L2 '(4 5 6))
> (append-2! L1 L2)
> L1
> L2
> (set-cdr! L2 null)
> L2
> L1
```

Exercice 5 *Soit une paire* `(define P (cons 1 2))`*, quelle est la valeur de* `P` *après l'évaluation de* `(set-cdr! P P)` *?*

2.2 La structure de donnée file mutable

Une *file* est une séquence d'éléments où les éléments peuvent être insérés à une extrémité (appelée *queue*) et récupérés à l'autre extrémité (appelée *tête*). On parle souvent de file FIFO (First In First Out) (en opposition à la structure de pile LIFO (Last In First Out).

Interface de la structure de file :

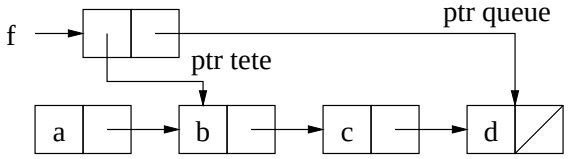
Constante	
`file-vide`	la file vide

Constructeur	
`(creer-file)`	construit une file vide

Accesseurs	
`(ajouter-file! f e))`	ajoute l'élément `e` dans la file `f`
`(oter-file! f)`	ôte l'élément en tête de la file `f`(l'élément est retourné)

Prédicats	
`(file-vide? f)`	teste si la file `f` est vide

Une file peut être représentée par une structure du type :



Exercice 6 *Écrivez les procedures de l'interface de la structure de donnée file.*