Cours 1 - Introduction à Lisp

Marie-Hélène Abel

Table des matières

I - Un peu d'Histoire *5*

II - Caractéristiques de Lisp *7*

A. Programmation interactive....*7*

B. Programmation fonctionnelle....*7*

C. Programmation symbolique....*8*

III - Concepts de base *9*

A. Objet de base : l'atome....*9*

B. Structure de base : la liste....*10*

IV - Boucle d'interaction *11*

A. Principes....*11*

B. Evaluation d'un atome....*11*

C. Evaluation d'une liste....*12*

V - Les primitives *13*

A. Préambule....*13*

B. La forme spéciale QUOTE....*13*

C. Affectation d'une valeur à un symbole....*14*

VI - Les listes *15*

A. Représentation interne....*15*

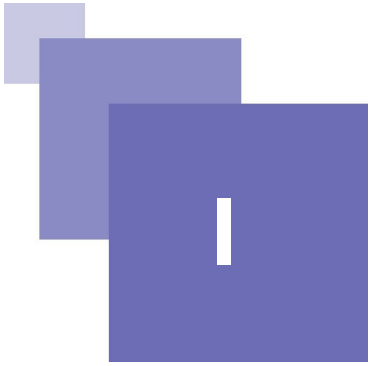
B. Construction....*16*

C. Manipulation....*16*

VII - La fonction EVAL *21*

VIII - La notion d'égalité *23*

Un peu d'Histoire



- Inventé en 1958 au MIT par J. McCarthy : un des langages de programmation les plus anciens.
- Utilisation longtemps limitée à cause de la faible puissance des ordinateurs.

Aujourd'hui, utilisé non seulement en IA, mais aussi dans d'autres domaines :

- Éditeur programmable : EMACS
- Systèmes de CAO programmable : AutoCAD
- Systèmes de manipulation symbolique

Il existe plusieurs dialectes LISP, dans ce cours : Common Lisp

Caractéristiques de Lisp

II

Programmation interactive	7
Programmation fonctionnelle	7
Programmation symbolique	8

A. Programmation interactive

Principales caractéristiques

- Fonctions et expressions peuvent être compilées et évaluées individuellement depuis un environnement interactif
- Programmation incrémentale

Rappel : Programmation classique

- Programme complet pour n'importe quelle tâche
- Fichier => compilation => exécution

B. Programmation fonctionnelle

Principales caractéristiques

- Pas de structure fixe
- Un ensemble de fonctions qui s'appellent entre elles.

La programmation fonctionnelle favorise :

- Le découpage d'un programme en fonctions (modularité)
- La récursivité

=> L' exécution est souvent beaucoup plus compliquée que le programme.

C. Programmation symbolique

Principale caractéristique

Gamme de données étendue par rapport aux langages classiques

En Lisp :

- les symboles sont crées et détruits dynamiquement : arguments de fonctions, symboles locaux, symboles globaux

- Les valeurs sont également des symboles

Rappel : Dans les langages classiques

- Ensemble fixe de variables
- Gamme de valeurs fixée par le type : nombres entiers, chaîne de caractères, etc.

Concepts de base

III

Objet de base : l'atome 9
Structure de base : la liste 10

A. Objet de base : l'atome

Définition : Atome

Un atome est soit un nombre soit un symbole

Définition : Nombre

Tout numérique

Exemple : Nombres

12
3.14
-24

Définition : Symbole

- Toute séquence non vide de caractères
- Permet de désigner les différents objets manipulés par un programme lisp : données, fonctions
- Certains caractères spéciaux, tels que le délimiteur "blanc" et les parenthèses ne peuvent être utilisés comme symbole.

Exemple : Symboles

Toto
+
*
#bidule#

B. Structure de base : la liste

Définition : Liste

Une liste est une séquence ordonnée, éventuellement vide, d'atomes ou de listes, précédée par une parenthèse ouvrante et suivie d'une parenthèse fermante.

Exemple : Listes

(* 2 3)
(ceci est une liste)

Boucle d'interaction

IV

Principes	11
Evaluation d'un atome	11
Evaluation d'une liste	12

A. Principes

Préambule

Une expression est soit un atome soit une liste d'expression.

Cycle de la boucle

La boucle d'interaction correspond à un cycle, elle est composée de 3 étapes bien distinctes :

- Read : lecture d'une expression
- Eval : évaluation de l'expression
- Print : impression du résultat du calcul dans la fenêtre d'interaction

Dès qu'un cycle est terminé, l'invite (>) réapparaît, l'environnement lisp est prêt pour une autre interaction.

B. Evaluation d'un atome

Cas d'un nombre

La valeur d'un nombre est ce nombre lui-même

Cas d'un symbole

La valeur d'un symbole est :

- Prédéfinie et non modifiable : le symbole est alors une des 2 constantes Lisp : 't' ou 'nil'
- Définie et modifiée dynamiquement par l'utilisateur ou par programme : le symbole est une variable.

Remarque

La valeur d'un nombre est ce nombre lui-même.

C. Evaluation d'une liste

Méthode : Principes

Soit la liste (s0 s1 s2 ... sN) . LISP :

- considère que s0 est un symbole qui désigne un nom de fonction -> s0 n'est pas évalué ;
- évalue les autres éléments ;
- applique la fonction associée à s0 aux arguments évalués et retourne comme résultat la valeur de cette application.

Exemple : Addition

```
> (+ 1 2)
3
>
```

Les primitives

V

Préambule	13
La forme spéciale QUOTE	13
Affectation d'une valeur à un symbole	14

A. Préambule

- L'environnement Lisp contient un certain nombre de fonctions prédéfinies : les primitives.
- Common Lisp comporte environ un millier de primitives. En particulier, on trouve les fonctions (comme celles opérant sur les nombres), les opérateurs spéciaux ou formes spéciales (defun, quote, setf, ...) et les macros.

B. La forme spéciale QUOTE

Exemple : Evaluation d'un concept de base

```
> a
Erreur
Le symbole a n'est pas connu a priori de Lisp, il n'a pas de valeur
```

Définition

La forme spéciale QUOTE permet de bloquer l'évaluation de son argument.

Syntaxe

```
(quote argument)
'argument
```

Exemple

```
> (quote a)
A
> 'a
A
>
```

C. Affectation d'une valeur à un symbole

Définition : La fonction set

Affecte une valeur à un symbole

Exemple

(set a 12)

Méthode : Processus d'évaluation d'une liste

- Evaluation séquentielle des arguments
- Affectation

Solution

Afin d'éviter la même erreur que précédemment :
(set 'a 12)

Définition : La fonction setq

Combinaison de la fonction **set** et de la forme spéciale **quote**

Exemple

(setq a 2) ≡ (set 'a 2) ≡ (set (quote a) 2)

Les listes

VI

Représentation interne	15
Construction	16
Manipulation	16

A. Représentation interne

Méthode

Une liste en Lisp est représentée par une liste simplement chaînée.
Une liste est construite à partir de paires constituées de deux cellules (pointeurs) :

- La première cellule (car) contient le premier élément.
- La deuxième cellule (cdr) contient une autre liste potentiellement vide.

Pour indiquer la fin de la liste, le **cdr** de la dernière paire est le symbole **nil.**

Exemple

image1

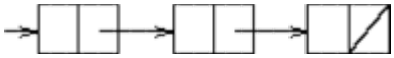


image2

Exemple

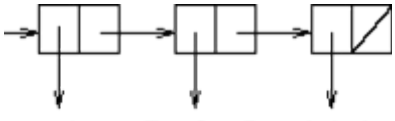


image3

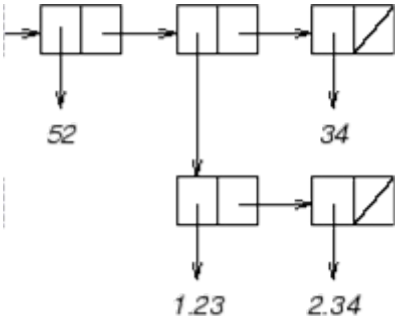


image4

B. Construction

Définition : La fonction cons

- Fonction à deux arguments x et y
- Retourne la paire dont le car est x et le cdr y

Exemple

```
>(cons 'a '(b c))
(a b c)
>(cons 'a nil)
(a)
```

Définition : La fonction list

- Admet un nombre arbitraire d'arguments
- Retourne une liste constituée de ces éléments

Exemple

```
>(list 1 2 3 4)
(1 2 3 4)
>
```

C. Manipulation

Accès aux éléments

Les principales fonctions d'accès sont : car, cdr et last

Définition : La fonction first (ou car)

Retourne le premier élément de la liste

Exemple

```
>(first '(a b c))
A
> (first '((a) (b c)))
(a)
>
```

Définition : La fonction rest (ou cdr)

Retourne le reste de la liste

Exemple

```
>(rest '(a b c))
(b c)
>(rest '((a) (b c)))
((b c))
>
```

Définition : La fonction last

Retourne la liste contenant uniquement son dernier élément.

Exemple

```
>(last '(a b e))
(e)
>
```

Chaînage des fonctions d'accès

- Les fonctions **car** et **cdr** peuvent être combinées et peuvent faire l'objet d'abréviations jusqu'à 4 niveaux.
- Dans le schéma C*R, * représente une chaîne d'au plus quatre éléments de a ou d.

Exemple

(cadr '(a b c)) ≡ (car (cdr '(a b c)))

(cdddr '(a b c d')) ≡ (cdr (cdr (cdr '(a b c d))))

(cadar '((a b c) d)) ≡ (car (cdr (car '((a b c) d))))

Autres fonctions

Voici quelques fonctions utiles de manipulation de listes : append, member et length

Définition : La fonction append

Retourne une liste constituée des arguments dans l'ordre donné.

Exemple

```
>(append '(a b) '(c 3))
(a b c 3)
>
```

Définition : La fonction member

- Vérifie si un élément fait partie d'une liste.
- Retourne nil s'il n'est pas présent ou la liste à partir de la première occurrence de l'élément.

Exemple

```
>(member 'a '(f g a e a))
```

```
(a e a)
>
```

Définition : La fonction length

Retourne le nombre d'éléments de la liste précisée en argument

Exemple

```
>(length '(a b c d))
4
>
```

Fonctions destructives

La plupart des fonctions lisp, comme **append**, construisent des copies de listes.

Exemple

```
>(setq *l1* '(a b))
(a b)
>(setq *l2* '(1 2))
(1 2)
>(append *l1* *l2*)
(a b 1 2)
>*l1*
(a b)
>*l2*
(1 2)
>(nconc *l1* *l2*)
(a b 1 2)
>*l1*
(a b 1 2)
>*l2*
(1 2)
>
```

Epilogue

- **Append** ne modifie pas les listes *l1* et *l2* car *l1* est copiée pour construire le résultat.
- La fonction **nconc** ne fait pas de copie, *l1* sera donc modifiée.

Représentation interne du append

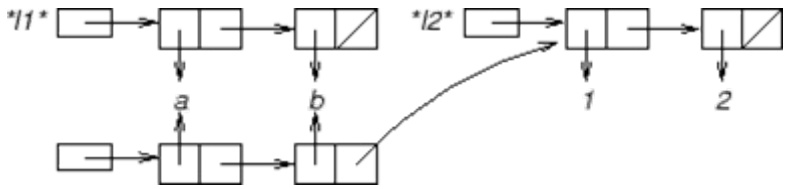


Image5

La fonction EVAL

VII

Définition

La fonction eval permet d'exécuter sur son argument la deuxième étape de la boucle d'interaction.

Exemple

```
>(setq b 'a)
A
>(setq a 3)
3
>A
3
>B
A
>(eval b)
3
>
```

La notion d'égalité

VIII

Préambule

- En lisp, une valeur est toujours un pointeur sur un objet lisp
- Il y a 4 formes d'égalité : EQ, EQUAL, = , EQL

Définition : Egalité EQ

Egalité des pointeurs, égalité des objets lisp

Exemple

```
>(setq x 3)
3
>(eq x x)
T
>(eq 'x 'x)
T
>(eq '(x) '(x))
nil
>
```

Définition : Egalité EQUAL

Egalité des symboles aux feuilles de l'objet référencé

Exemple

```
>(equal '(x) '(x))
T
>(equal 'x 'x)
T
>
```

Définition : Egalité =

Egalité pour les nombres

Définition : Egalité EQL

EQ ou = selon l'interpréteur lisp utilisé