Techniques et Fondements de la Programmation

Responsable et cours : Robert Strandh

TD : Irène Durand
Laurent Péron

Planning : 13 semaines
(12 semaines de cours, 12 semaines de TD)

2h de cours + 2h de TD par semaine
4h de travail individuel par semaine

Web : http://dept-info.labri.fr/~strandh/Teaching/TFP/2001-2002/Dir.html

Support de cours

Robert Strandh et Irène Durand :
Traité de programmation en Common Lisp

Transparents

Bibliographie

Paul Graham : ANSI Common Lisp
Prentice Hall

Paul Graham : On Lisp
Advanced Techniques for Common Lisp
Prentice Hall

Sonya Keene : Object-Oriented Programming in Common Lisp
A programmer's guide to CLOS
Addison Wesley

David Touretzky : Common Lisp:
A Gentle introduction to
Symbolic Computation.

Autres documents

Guy Steele : Common Lisp, the Language, second edition
Digital Press
(disponible sur WWW en HTML)

David Lamkins : Successful Lisp (Tutorial en-ligne)

CMUCL user's guide

CLX reference manual

The HyperSpec (la norme ANSI complète de Common Lisp, en HTML)

Objectifs et contenu

Objectif : Maîtriser un certain nombre de méthodes et techniques de programmation symbolique, impérative et fonctionnelle.

Contenu :
Langage Common Lisp
Types : symbole, liste, ...
Fonctions de base
Techniques élémentaires de programmation
Macros
Exceptions
Programmation par objets avec CLOS

But de l'enseignement de programmation

Méthodes et techniques pour l'écriture d'applications maintenables, réutilisables, lisibles, modulaires, générales, élégantes.

Langages, mécanismes et outils adaptés à ces méthodes et techniques

Aspects de la programmation non enseignés :

- Efficacité extrême
- Temps réel
- Applications particulières (jeux, image, numériques, ...)
- Détails de langages particuliers
- Implémentations particulières

Comment faire pour apprendre à programmer?

Il faut surtout lire beaucoup de code écrit par des experts.

Il faut lire la littérature sur la programmation. Il n'y en a pas beaucoup (peut-être 10 livres).

Il faut programmer.

Il faut maintenir du code écrit par d'autres personnes.

Il faut apprendre à être bien organisé.

Pourquoi le langage Common Lisp?

- Langage très riche
- Syntaxe simple et uniforme
- Sémantique simple et uniforme
- Langage programmable (macros, reader macros)
- Représentation de programmes sous la forme de données
- Normalisé par ANSI, y compris CLOS
- Programmation par objets plus puissante qu'avec d'autres langages

Définition : langage interprété

(deux définitions possibles)

1. Un langage pouvant être implémenté par un interprète
2. Un langage forcement implémenté par un interprète

Si 1, alors tous les langages sont interprétés. Si 2, alors aucun langage n' est interprété.

Alors, "langage interprété" n'a pas de sens.

Historique de Common Lisp

Langage inventé à MIT par John McCarthy en 1959 (donc avec Fortran l'un des plus vieux langages toujours utilisés)

Dans les années 1970, deux dialectes : Interlisp et Maclisp

Aussi : Standard Lisp, NIL, Lisp Machine Lisp, Le Lisp

Travail pour uniformiser les dialectes : Common Lisp

Normalisé par ANSI en 1987

Processus de standardisation de Common Lisp

1. Introduction d'un concept nouveau
2. Distribution sous la forme de fonctions+macros
3. Utilisation par les programmeurs pendant plusieurs années
4. Proposition de normalisation du concept
5. Intégration du concept dans la norme

Tour guidé de Common Lisp

- Langage dynamique
- Sémantique par référence uniforme
- Gestion automatique de la mémoire
- Mots clés (passage de paramètres)
- Compilation
- Macros
- Programmation orientée objets

Tour guidé de Common Lisp

Types de données

- entiers à précision arbitraire
- rationnels, flottants
- complexes rationnels et complexes flottants
- la plupart des opérations arithmétiques marchent sur les complexes
- symboles
- caractères, chaînes de caractères
- séquences, listes, tableaux
- tableaux de bits, tableaux d'hachage
- fonctions
- structures, classes, métaclasses
- séries, générateurs/consommateurs
- flots

Tour guidé de Common Lisp (suite)

Structures de contrôle

- conditionnels (*if*, *when*, *unless*, *cond*, *case*)
- récursivité
- itération (*while*, *do*, *dolist*, *loop*)
- blocs (*prog1*, *progn*, *tagbody*, *let*, *let**)
- fermetures
- fonctions génériques
- valeurs multiples
- exceptions

Standards de codage

Il faut s'habituer aux standards de codage

Pour Common Lisp, suivre les exemples dans la littérature

En particulier pour l'indentation de programmes qui est très standardisée.

Il faut pouvoir comprendre le programme sans regarder les ()

Utiliser lisp-mode de GNU Emacs

L'indentation n'est donc pas une question de goût personnel

Programmation fonctionnelle

Common Lisp est un langage mixte (fonctionnel, impératif, orienté objets)

Pour un bon résultat, utiliser la programmation fonctionnelle le plus possible

Par programmation fonctionnelle, il faut comprendre la programmation sans effets de bord

Les programmes sont plus faciles à tester

Programmation ascendante (bottom-up)

Paramètres sont souvent des fonctions (fermetures)

Sémantique par référence uniforme

Manipulation d'objets par des références (pointeurs)

Le compilateur peut remplacer une référence par l'objet correspondant si cela ne change pas la sémantique

Implémentation d'objets typés par le marquage (tags)

Les petits objets (fixnums, caractères, ...) sont représentés directement dans le pointeur.

La sémantique par référence uniforme fait que les pointeurs explicites ne sont pas nécessaires ("pas de pointeurs")

Les déclarations peuvent aider le compilateur à remplacer les références par les objets correspondants

Common Lisp est interactif

Common Lisp est presque toujours implémenté sous la forme de système interactif avec une boucle d'interaction (read-eval-print loop ou REPL).

Une interaction calcule la valeur d'une expression, mais une expression peut aussi avoir des effets de bord.

En particulier un effet de bord peut être de modifier la valeur d'une variable ou de créer une fonction.

Le langage n'a pas la notion de programme principal. Il est néanmoins possible de préciser la fonction à exécuter quand l'application est lancée.

Normalement, on exécute la commande Lisp une seule fois par séance.

Common Lisp est interactif (suite)

En EMI, une séance est un TD ou une demie journée de travail. Sur un ordinateur personnel, une séance peut durer des mois.

Le langage est conçu pour le développement interactif. Les instances d'une classes sont mises à jour quand la définition d'une classe change, par exemple.

Il faut écrire les applications pour faciliter l'interactivité aussi. Souvent cela nécessite la programmation sans effets de bord.

Expressions

Une expression (ou une forme) Common Lisp peut être :

- un objet auto-évaluant,
- un symbole, ou
- une expression composée.

Expressions, analyse syntaxique

Une expression tapée à la boucle d'interaction est d'abord analysée syntaxiquement. Le résultat de cette analyse est une représentation interne de l'expression. La fonction responsable de cette analyse s'appelle *read*.
Cette fonction est disponible à l'utilisateur.

Expressions, évaluation

La représentation interne de l'expression est ensuite évaluée, c'est à dire que sa valeur est calculée. Cette évaluation peut donner des effets de bord. Le résultat de l'évaluation est un ou plusieurs objets Lisp. La fonction responsable de l'évaluation d'expression s'appelle *eval*.
Cette fonction est disponible à l'utilisateur.

Expressions, affichage

Les objets résultant de l'évaluation sont ensuite affichés (ou imprimés) en représentation externe. La fonction responsable de l'affichage s'appelle *print*.
Cette fonction est disponible à l'utilisateur.

Lancer et quitter le système Lisp

```
napperon: lisp
CMU Common Lisp release x86-linux...
...
* 1234

1234
*
```

Lancer et quitter le système Lisp

```
* hello

Error in ...:  the variable HELLO is unbound.

Restarts:
  0: [ABORT] Return to Top-Level.

Debug  (type H for help)
...
0] 0
* (quit)
napperon:
```

Objets auto-évaluants

Un objet auto-évaluant est la même chose qu'une constante. Typiquement, il s'agit de nombres, de caractères ou de chaînes de caractères.

```
* 1234

1234
* 6/8

3/4
* #\c

#\c
*
```

Objets auto-évaluants

```
* "bonjour"

"bonjour"
* #(1 2 3 4)

#(1 2 3 4)
*
```

Symboles

Si l'expression est un symbole, il sera considéré comme le nom d'une variable. la fonction eval va donc renvoyer sa valeur.

```
* *standard-output*

#<Synonym Stream to SYSTEM:*STDOUT*>
* nil

NIL
* t

T
*
```

Symboles

```
* *features*

(:MK-DEFSYSTEM :CLX-MIT-R5 ...)
* +

*FEATURES*
*
```

Expressions composées

Une expression composée est une liste de sous-expressions entourées de parenthèses. Souvent, la première sous-expression est un symbole qui est le nom d'une fonction, d'une macro ou d'un opérateur spécial. Les autres sous-expressions sont les arguments de la fonction, de la macro ou de l'opérateur spécial.

Expressions composées

```
* (+ 3 4)

7
* (length "hello")

5
* (+ (* 3 4 2) (- 5 4) (/ 5 3))

80/3
*
```

Expressions composées

```
* (if (> 5 4) "oui" "non")

"oui"
* (length *features*)

37
*
```

Ici, *if* est un opérateur spécial, alors que +, *, -, /, >, *length* sont des fonctions.

Expressions avec effets de bord

Certains expressions peuvent avoir des effets de bord. Il s'agit par exemple de l'affectation d'une variable ou de l'affichage (autre que par la boucle REPL)

```
* (setf x 3)
Warning:  Declaring X special.

3
* x

3
* (+ (print 3) 4)

3
7
*
```

Expressions avec effets de bord

Ici, *setf* est le nom d'une macro et *(setf x 3)* est une expression macro (anglais : macro form); *print* est le nom d'une fonction avec un effet de bord et *(print 3)* est une expression fonction (anglais : function form).

Définition de fonction

L'utilisateur peut définier des fonctions en utilisant la macro *defun* :

```
* (defun double (x)
    (* 2 x))

DOUBLE
* (double 10)

20
* (double 3/2)

3
* (double 1.5)

3.0
*
```

Définition de fonction

```
* (defun my-gcd (x y)
    (if (= x y)
        x
        (if (> x y)
            (my-gcd y x)
            (my-gcd x (- y x)))))

MY-GCD
*
```

Cette indentation est obligatoire, car les programmeurs Lisp ne regardent pas les parenthèses. De plus, elle doit être automatique.