



UNIVERSITÉ
CÔTE D'AZUR

Programmation impérative en C

Valisp 5. Histoire de LISP

Olivier Baldellon

Courriel : `prénom.nom@univ-cotedazur.fr`

Page professionnelle : `https://upinfo.univ-cotedazur.fr/~obaldellon/`

LICENCE 2 — FACULTÉ DES SCIENCES ET INGÉNIEURIE DE NICE — UNIVERSITÉ CÔTE D'AZUR

- 🍃 Partie I. Les débuts de LISP
- 🍃 Partie II. Scheme
- 🍃 Partie III. Des LISP machines à Common Lisp
- 🍃 Partie IV. LISP Aujourd'hui
- 🍃 Partie V. Conclusion
- 🍃 Partie VI. Table des matières

:)

Valisp

```
(defun APPLY (FONCTION PARAM ENV)
  (cond ((EQ FONCTION (QUOTE CONSP)) (CONSP (EVAL (CAR PARAM) ENV)))
        ((EQ FONCTION (QUOTE CAR)) (CAR (EVAL (CAR PARAM) ENV)))
        ((EQ FONCTION (QUOTE CDR)) (CDR (EVAL (CAR PARAM) ENV)))
        ((EQ FONCTION (QUOTE EQ)) (= (EVAL (CAR PARAM) ENV) (EVAL (CAR (CDR PARAM)) ENV)))
        ((EQ FONCTION (QUOTE CONS)) (CONS (EVAL (CAR PARAM) ENV) (EVAL (CAR (CDR PARAM)) ENV)))
        ((EQ FONCTION (QUOTE QUOTE)) (CAR PARAM))
        ((EQ FONCTION (QUOTE COND)) (EVAL-COND PARAM ENV))
        ((CONSP FONCTION) ;; (= (CAR FONCTION) 'LAMBDA)
         (EVAL (CAR (CDR (CDR FONCTION)))
                (MAKE-ENV (CAR (CDR FONCTION))
                          PARAM
                          ENV)))
        (T (APPLY (EVAL FONCTION ENV) PARAM ENV))))
```

- ▶ On conserve un tableau de tout les symboles créés
- ▶ La fonction `new_symbol`
 - ▶ regarde si le symbole existe déjà
 - ▶ Si oui, on récupère le pointeur.
 - ▶ sinon, on copie la chaîne du symbole en mémoire
- ▶ Permet d'éviter de nombreuses allocations
- ▶ Attention au ramasse-miettes !
 - ▶ Il faut mettre à jour le tableau après le ramasse-miettes

- 🍃 Partie I. Les débuts de LISP
- 🍃 Partie II. Scheme
- 🍃 Partie III. Des LISP machines à Common Lisp
- 🍃 Partie IV. LISP Aujourd'hui
- 🍃 Partie V. Conclusion
- 🍃 Partie VI. Table des matières



- ▶ *Structure and interpretation of computer programm*
- ▶ Introduction de la liaison statique (closures)

Valisp

```
(defun plus-n (n)
  (lambda (x) (+ x n)))

(defvar incr (plus-n 1)) ;; (lambda (x) (+ x 1))

(defun suivant (n)
  (incr n))
```

- ▶ Que vaut (suivant 10)
 - ▶ 11?
 - ▶ 20?

- ▶ En fait `incr` vaut `(lambda (x) (+ x n))`
 - ▶ Et que vaut `n`?

- ▶ On rajoute une forme spéciale
 - ▶ et un nouveau type `closure` (en plus des entiers, symboles, etc.)
 - ▶ codé avec un `COUPLE` (comme les `CONS`)
 - ▶ `(env-de-definition . fonction-lambda)`

Valisp

```
(defun plus-n (n)
  (closure (x) (+ x n)))

(defvar incr (plus-n 1)); (lambda (x) (+ x n))
                       ; dans l'environnement (n . 1) → env

(defun suivant (n)
  (incr n))
```

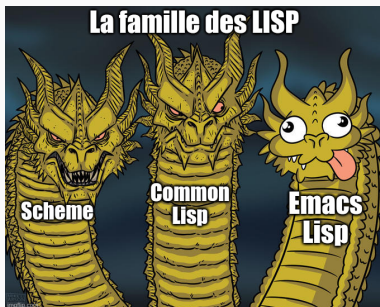
- ▶ `sexpr new_closure`(`sexpr env`, `sexpr fonction`)
 - ▶ attention à l'affichage (boucle infinie)
 - ▶ l'environnement local pointant vers l'environnement globale
- ▶ `sexpr get_closure_env`(`sexpr closure`)
- ▶ `sexpr get_closure_lambda`(`sexpr closure`)
- ▶ On modifie `apply` pour évaluer le `lambda` dans l'env. de définition.

- 🍃 Partie I. Les débuts de LISP
- 🍃 Partie II. Scheme
- 🍃 Partie III. Des LISP machines à Common Lisp
- 🍃 Partie IV. LISP Aujourd'hui
- 🍃 Partie V. Conclusion
- 🍃 Partie VI. Table des matières

:)

:)

- 🍃 Partie I. Les débuts de LISP
- 🍃 Partie II. Scheme
- 🍃 Partie III. Des LISP machines à Common Lisp
- 🍃 **Partie IV. LISP Aujourd'hui**
- 🍃 Partie V. Conclusion
- 🍃 Partie VI. Table des matières



- ▶ Emacs Lisp
 - ▶ C'est le plus récent (1984)
 - ▶ mais aussi le plus archaïque
 - ▶ et un des plus minimaliste
- ▶ Proche de Common Lisp
 - ▶ La syntaxe est quasi identique

- ▶ Emacs n'est pas un éditeur de texte
- ▶ C'est un interpréteur Lisp
 - ▶ Chaque touche appelle une fonction
 - ▶ On peut combiner ces fonctions pour en écrire de nouvelles
 - ▶ Et les associer à des touches
- ▶ C'est le dernier héritage des hackers du MIT
 - ▶ un logiciel avec un contrôle complet
 - ▶ un bac à sable
- ▶ Aujourd'hui la mode est de tout faire en javascript dans le navigateur
 - ▶ Même concept
 - ▶ La liberté en moins

- 🍃 Partie I. Les débuts de LISP
- 🍃 Partie II. Scheme
- 🍃 Partie III. Des LISP machines à Common Lisp
- 🍃 Partie IV. LISP Aujourd'hui
- 🍃 **Partie V. Conclusion**
- 🍃 Partie VI. Table des matières

- ▶ L'informatique est riche en langage divers
 - ▶ Lisp, Erlang, Prolog, Smalltalk
 - ▶ Ces langages donnent une autre définition au mot *programmer*
- ▶ Le C permet de comprendre comment fonctionne les machines
 - ▶ Je vous encourage à jeter un coup d'œil à Rust
- ▶ Vous devez maîtriser vos outils !
 - ▶ Privilégier les outils libres et hackables

Merci pour votre attention

Questions



Hello world !



Cours 5 — Histoire de LISP

Partie I. Les débuts de LISP

Les débuts de l'IA

L'interpréteur métacirculaire

La gestion des symboles

Partie II. Scheme

Scheme

Le problème du *funarg*

Les closures

Partie III. Des LISP machines à Common Lisp

Les débuts

Les LISP Machines et CL

Partie IV. LISP Aujourd'hui


Emacs Lisp

Emacs

Partie V. Conclusion

Conclusion

Partie VI. Table des matières

- ▶ © 2026 — Olivier Baldellon
- ▶ Ce document est publié sous licence **CC-BY Attribution 4.0** 
 - Vous êtes autorisé à :
 - ▶ **Partager** — copier, distribuer et communiquer le matériel par tous moyens et sous tous formats pour toute utilisation, y compris commerciale.
 - ▶ **Adapter** — remixer, transformer et créer à partir du matériel pour toute utilisation, y compris commerciale.
 - Selon les conditions suivantes :
 - ▶ **Attribution** — Vous devez créditer l'œuvre, intégrer un lien vers la licence et indiquer si des modifications ont été effectuées à l'œuvre. Vous devez indiquer ces informations par tous les moyens raisonnables, sans toutefois suggérer que l'Offrant vous soutient ou soutient la façon dont vous avez utilisé son œuvre.
- ▶ <https://creativecommons.org/licenses/by/4.0/deed.fr>