

ENS Rennes  
L3 Informatique, parcours R & I  
Programmation avancée, 2<sup>e</sup> semestre 2015–2016

---

Projet 2 : Interprète Lisp en C++

Luc Bougé

4 avril 2016 \*

## Introduction

### Présentation du projet

Le but de ce projet est de faire une synthèse de l'ensemble de votre formation en programmation cette année. Vous avez étudié dans vos divers cours une grande variété de langages : Python, Caml, Scala, C, C++, Lisp, Scheme, etc. Vous avez aussi été exposés à une grande variété d'approches de la programmation, ce qui est très bien.

Dans la plupart des cours, chaque langage et son approche associée ont été étudiées spécifiquement. L'objectif est ici de vous proposer un projet qui mixe plusieurs approches : C++ et Lisp/Scheme, en prenant appui sur Caml. Vous aurez aussi l'occasion d'utiliser les outils Lex/Flex et Yacc/Bison qui sont une application directe de votre cours de théorie des langages.

Vous connaissez déjà les références pour C++ et Lisp/Scheme. Il existe de très nombreux tutoriels pour Lex/Flex et Yacc/Bison. Nous allons ici utiliser la version C++ de ces outils. Un tutoriel rapide pour l'utilisation *conjointe* de Flex et Bison est disponible à l'adresse

[http://aquamentus.com/flex\\_bison.html](http://aquamentus.com/flex_bison.html)

Flex et Bison sont des outils GNU avec une documentation abondante à l'adresse habituelle.

### Déroulement et évaluation

Le sujet est composé d'une partie obligatoire commune et d'un choix de parties optionnelles ouvertes et graduées. Il est demandé de choisir l'une des parties optionnelles, pas plus. Par contre, un grand soin est attendu dans le travail de conception et de validation.


---

\*Id: projet2\_Lisp.tex 2052 2016–04–03 19:36:04Z bouge

Vous avez le droit de vous inspirer et même le devoir de reprendre les codes que vous avez écrits dans les séances de TP précédentes. Vous êtes encouragés à utiliser vos notes de cours, tout document et tout livre à votre convenance, et à consulter sur Internet la documentation C++ et Lisp/Scheme et tout site qui vous paraîtrait utile.

Ce projet a été conçu entièrement cette année. Je vous remercie par avance de vos commentaires et suggestions ! Malheureusement, il est difficile d'évaluer le temps nécessaire. Ce sera ajusté avec vous au cours du projet.

## 1 Présentation du code préparé

 *L'objectif est ici la robustesse du codage, pas la performance. Toute la phase de codage doit être traitée de manière défensive en utilisant des assertions à chaque fois que c'est possible. Utilisez largement les variables intermédiaires pour être sûrs de vos manipulations. Ne cherchez surtout pas à optimiser, mais assurez-vous de la correction de vos fonctions pas à pas.*

Le code C++ qui est préparé implémente une forme très rudimentaire d'un interprète à liaison dynamique, sur le modèle de l'interprète MLlisp\_dynamic que nous avons utilisé en cours.

Les principes de structuration des codes C++ ne sont pas les mêmes que ceux de Caml. Ce sera à vous de bien distribuer votre code entre différents fichiers et modules.

### 1.1 Cellules et objets Lisp

L'utilisation de C++ demande une certaine virtuosité pour la mise au point de la classe Cell qui est l'unité de base de l'allocation Lisp. En effet, une cellule peut contenir un nombre, une chaîne, un symbole ou une paire de pointeurs vers des cellules. C'est donc une **union** C++. Malheureusement, dans la version du langage que nous utilisons, les **union** ne peuvent pas contenir des types avec constructeurs, en particulier string. Il faut donc stocker les chaînes et les symboles sous la forme de tableau de caractères, comme en C.

[https://en.wikipedia.org/wiki/Union\\_type](https://en.wikipedia.org/wiki/Union_type)

C++ (since C++11) also allows for a data member to be any type that has a full-fledged constructor/destructor and/or copy constructor, or a non-trivial copy assignment operator. For example, it is possible to have the standard C++ string as a member of a union.

Nous devons donc remplacer les string C++ par des tableaux de caractères (des pointeurs) alloués explicitement dans le tas par malloc dans la fonction strdup. Ce point sera important dans l'étude fine de l'algorithme de gestion de la mémoire (*garbage collection*). Vous trouverez dans le fichier cell.cc la définition de la classe Cell et des méthodes qui permettent de créer une cellule à partir d'un objet d'un certain type (**int**, **string**, etc.) et dans le sens contraire de retrouver l'objet original d'une cellule. Tous ces traitements délicats sont entourés d'assertions sur un mode très défensif.

La classe des objets Lisp est la classe Object définie dans le fichier object.cc. L'adresse d'un objet **static** de la classe Cell est utilisé comme la valeur de nil. Les fonctions usuelles sont définies sur les objets.

## 1.2 Évaluateur

Une forme simple de l'évaluateur Lisp est programmée dans le fichier `eval.cc`. La récursivité mutuelle des fonctions `eval` et `apply` est plus simple à traiter qu'en Caml. Une trace rudimentaire est mise en place pour ces deux fonctions.

Les erreurs sont traitées par des appels à des exceptions C++ dérivées de la classe `runtime_error`.

Le traitement des *subr* est minimal : seuls `+` et `*` sont définis et leurs définitions sont incluses directement dans le code de l'évaluateur.


## 1.3 Environnement

Les environnements sont gérés par la classe `Environnement` définie dans le fichier `env.cc`. Un environnement est ici implémenté par une liste au sens C++ d'objets Lisp, c'est-à-dire de pointeurs vers des cellules. À chaque fois que l'on passe sous un  $\lambda$ , une copie de l'environnement est faite, ce qui est bien sûr peu efficace. Ce sera un point à améliorer, ô futurs chercheurs !

Les erreurs sont traitées par des appels à des exceptions C++ dérivées de la classe `runtime_error`.


## 1.4 Lecture

Les lectures sont faites par un *lexer* et un *parser* construits par les outils Flex et Bison. Toute erreur de syntaxe est fatale. Les chaînes (entre guillemets), les symboles et les entiers sont reconnus et le *quote* est expansé : `'(1 2 3)` est lu comme `(quote (1 2 3))`. Le symbole `nil` est transformé à la volée en la liste vide.

 Le parser *lisp.y* n'est pas complètement au point par manque de temps de ma part. Ce sera donc à vous de le compléter ! ☺

## 1.5 Toplevel

Il n'y a pas de *toplevel*. À vous de le faire en vous inspirant de l'interprète Caml.


 Vous aurez en particulier à gérer le rattrapage des exceptions lancées par l'évaluateur et l'environnement.

# 2 Partie obligatoire : interprète à liaison dynamique

Dans un premier temps, il vous est demandé de compléter ce code pour obtenir un interprète du niveau de l'interprète `MLisp_dynamic` que nous avons utilisé en cours. Voici quelques étapes de base, mais vous pouvez tout à fait en proposer d'autres.

**Toplevel.** Construisez un *toplevel* qui reconnaît la directive `setq` et qui lance l'évaluation. Les erreurs d'évaluation doivent être rattrapées, signalées à l'utilisateur et traitées.

**subr.** Implémentez une gestion des fonctions *subr* intégrée dans l’environnement comme dans l’interprète du cours. Ajoutez les fonctions manquantes, en particulier *read*, etc.

 Ceci va nécessiter de revoir les classes de base pour pouvoir manipuler les *subr* dans le monde *Lisp*.

**Trace.** Ajoutez un dispositif de trace et d’impression (*pretty-printing*) pour aider à la mise au point.

**Démonstration.** Montrer que toutes les fonctions habituelles fonctionnent bien. Comparez la vitesse de votre interprète avec celui de l’interprète du cours.


### 3 Partie optionnelle : extensions

#### 3.1 Méta-interprétation ★

Enrichissez votre interprète pour pouvoir exécuter l’interprète méta-circulaire Scheme

`ch4—mceval.scm`

du livre SICP à l’adresse <https://mitpress.mit.edu/sicp/code/>.

 Cet interprète est écrit en Scheme, il faudra donc le modifier légèrement pour qu’il fonctionne dans un interprète à liaison dynamique.

#### 3.2 Amélioration de l’interprète ★★


Implémentez une gestion d’environnement par liste chaînée. Adaptez votre interprète pour passer en liaison statique. Implémentez la gestion par *frames* de Scheme. Implémentez les clôtures Scheme.

#### 3.3 Continuations et futurs ★★★

Implémentez la construction *call/cc* et la construction *delay* de Scheme. Vous pouvez le faire sur l’interprète Caml ou C++ à votre gré. Ceci nécessite l’utilisation des clôtures. Utilisez le livre SICP pour bien comprendre la sémantique de ces constructions.

#### 3.4 Gestion mémoire ★★★★


Dans la version actuelle, les cellules sont créés par la construction **new**. Créez une classe *Memory* qui fonctionne comme un réservoir de cellules.

 La class *Memory* doit gérer un vecteur statique de cellules. La méthode statique *allocate* renvoie l’adresse d’une cellule non encore utilisée, comme pour la fonction *malloc*. Votre code ne doit plus comporter de **new Cell**. Voici par exemple la nouvelle version de la fonction *cons* :

```
Object cons(Object a, Object l) {  
    // Object p = new Cell();  
    Object p = Memory::allocate();  
    p -> make_cell_pair(a, l);  
    return p;  
}
```

---

Quand la mémoire est saturée, il faut libérer de la place en faisant le ménage (*house keeping*). L'idée est de repérer toutes les cellules qui ne sont plus atteignables depuis le *topelevel* et de les *recycler* en des cellules à nouveau utilisables. Vous pouvez lire la section 5.3.2 de SICP pour une introduction.

 La mise au point d'un garbage collector (GC) est délicate. Il faut adopter une discipline de programmation très défensive pour être sûr de ne pas faire d'erreur sur le statut des cellules.

La référence classique dans ce domaine est le livre *The Garbage Collection Handbook : The Art of Automatic Memory Management*, par Richard Jones, Antony Hosking, Eliot Moss, CRC Press, 2011. La première version date de 1996. Il existe de très nombreux cours sur le Web sur ce sujet. Voir par exemple la page Wikipédia *Garbage Collection*.

[https://en.wikipedia.org/wiki/Garbage\\_collection\\_%28computer\\_science%29](https://en.wikipedia.org/wiki/Garbage_collection_%28computer_science%29)