Interprète Lisp en C++

Aurèle Barrière & Jérémy Thibault

15 avril 2016

Table des matières

1 Partie obligatoire : interprète à liaison dynamique

1.1 Exceptions

La première amélioration de notre toplevel fut de rattraper toutes les exceptions lancées par le programme. En effet, il ne fallait pas que l'interprète s'arrête si l'utilisateur fait une erreur : nous voulions qu'il lui indique le type d'erreur, l'objet concerné et qu'il continue à s'éxécuter.

Ainsi, nous avons créé un module `exceptions` contenant tous les types d'exceptions (des classes différentes) : exceptions liées à l'absence d'arguments, à un problème de typage, à un manque de liaisons dans l'environnement etc...

Ensuite, dans l'exécution du toplevel, on rattrape ces exceptions et on affiche les messages d'erreurs correspondant.

1.2 Organisation du toplevel

Dans le code initial, le fichier `main` contenait le toplevel et l'appel au parseur mélangés. Nous avons séparés ça.

Nous avons ainsi créé un module `toplevel` contenant une fonction `toplevel()` qui sera appelée dans le `main`. Ce module fait appel aux fonctions de bison pour parser un fichier donné.

1.3 Subroutines

Nous avons également eu besoin de rajouter des subroutines : au moins $-$ et $=$ étaient nécessaires pour implémenter des fonctions récursives.

Nous en avons profités pour séparer toutes les subroutines de l'évaluation dans un module distinct.

1.4 La directive setq

Pour ajouter des liaisons dans l'environnement courant, nous avons du ajouter un cas particulier à l'évaluation : `setq`.

Lorsque l'utilisateur utilise cette commande, on évalue le deuxième argument, et on crée la liaison entre le premier argument et l'objet évalué.

1.5 Mode verbeux et affichage d'environnement

Nous avons également laissé à l'utilisateur la possibilité d'activer le mode verbeux (qui indique chaque appel d'évaluation ou d'application de fonction) avec un argument optionnel.

Une commande est également disponible pour afficher l'environnement courant.

2 Partie optionnelle : extensions

Nous avons choisi d'étudier la gestion mémoire de notre interpréteur.

2.1 Allocation de cellules

2.2 Garbage collector

2.3 Mise en oeuvre et tests