IFT6892 – Projet intégrateur
Spécification, conception et développement d'un interprète

Guy Lapalme
en collaboration avec Dominique Boucher et Niculina Ignat

Mars 2002

Le projet vous incitera à intégrer ce que vous avez appris dans les cours IFT6820 et IFT6825 pour développer un interprète pour un langage d'expression arithmétiques *à la Scheme*.

En suivant la méthodologie de génie logiciel apprise au cours IFT6825, vous devez donc préciser la spécification donnée ici et concevoir un interprète d'expressions arithmétiques. Le choix du langage d'implantation de l'interprète est libre mais il devra être motivé lors de la présentation. Il est possible de vous inspirer de la solution du tp2 fournie par Dominique Boucher.

En utilisant et en adaptant ce que vous avez appris dans le cours IFT6820, vous allez implanter cet interprète. Cet interprète devra ensuite être testé selon la méthodologie présentée en IFT6825.

1 Description informelle du problème

L'application est un interprète de style *read-eval-print* pour un petit langage de programmation qui est du même niveau que celui que vous avez compilé lors du TP2 de IFT6820. Vous devez donc écrire une application qui lit une expression, l'évalue et imprime sa valeur. Ce processus est répété jusqu'à ce que l'usager indique la fin de fichier. Ce langage étant un sous-ensemble strict de Scheme, vous pouvez utiliser l'interprète pour voir les résultats attendus des expressions mais pas pour vous éviter d'interpréter vous-mêmes les expressions, il est donc interdit d'utiliser directement l'interprète Scheme comme évaluateur des expressions! Comme défi supplémentaire, vous pourrez accepter qu'une expression soit une définition d'une fonction.

1.1 Spécification formelle du langage à interpréter

Toutes les valeurs sont de type entier.

Nous décrivons la syntaxe du langage à l'aide d'une grammaire EBNF. Les non-terminaux seront dénotés comme des noms entre crochets < et >. Par exemple, <sexp> est le non-terminal produisant toutes les expressions symboliques. Les terminaux sont entourés d'une boîte. x^+ indique une ou plusieurs occurrences de x.

```
<sexp>  ::=  ( <op> <sexp>+ )  |  <ident>  |  <num>
<sexp>  ::=  ( define <ident> <sexp> )
  <op>  ::=  +  |  -  |  *  |  /  |  %
<ident> ::=  symbole de la forme: <lettre>  <lettre ou chiffre>
 <num>  ::=  entier signé
```

La valeur de (op e_0 $e_1 \cdots$ e_n) est $(((e_0$ op $e_1)$ op e_2 $) \cdots$ op $e_n)$. op est l'opérateur défini sur les entiers. La valeur de define est la valeur affectée à ident. Un identificateur a une valeur initiale indéfinie, ce qui entraîne une erreur lorsqu'on l'utilise sans le définir. La valeur courante est celle du dernier define rencontré.

Il ne faut retourner la valeur que de la première expression sur une ligne. Vous n'avez pas à traiter le cas d'une expression sur plusieurs lignes mais ceci serait un petit défi supplémentaire.

Vous pouvez arrêter le traitement de l'expression courante tant au niveau syntaxique que sémantique (e.g. division par zéro) aussitôt que vous détectez une erreur.

2 Défi supplémentaire

Vous pourriez traiter la définition et l'appel de fonction en ajoutant les règles suivantes:

```
<sexp>  ::=  ( lambda <ident> ( <ident> ) <sexp>+ )
<sexp>  ::=  ( ident <sexp> )
```

c'est donc une fonction à un seul paramètre où l'expression est évaluée en remplaçant la valeur du paramètre formel par celui du paramètre actuel. Cette définition implique d'accepter des fonctions en paramètres et de retourner des fonctions en résultats.

3 Evaluation

Vous devez travailler en groupe de 3 ou 4 personnes et l'évaluation se fera en deux temps, l'avant-midi du jeudi 2 mai:

- une présentation de 15 minutes décrivant votre analyse et votre spécification du problème
- une démonstration de 5 minutes de votre interprète sur machine

Il n'est pas nécessaire d'écrire un rapport, une copie des transparents que vous présenterez sera suffisante.