CS212 Oum-El-Kheir Aktouf, Arthur Baudet, Yann Kieffer
ESISAR, 2e année A rendre pour le jeudi 4 juin 2020

Devoir final pour le cours :

Structures de données et programmation récursive

L'ensemble de ce devoir porte sur un projet de calculatrice. Beaucoup de code vous est fourni, il y a peu de choses à rajouter pour faire fonctionner une première version de la calculatrice. Puis vous modifierez petit à petit la calculatrice pour augmenter ses capacités.

Le code fourni se compose de 5 modules :

Types.hs définit un type `Oper` pour les opérateurs, et un type `Exp` pour les expressions arithmétiques ;

Parseur.hs est un module qui s'occupe de transformer une expression arithmétique sous forme de chaîne de caractères, en une valeur de type `Exp` ;

Expression.hs est le module qui s'occupe d'afficher les expressions de différentes manières ;

Exemples.hs fournit des exemples d'expression ; n'hésitez pas à le compléter, il vous facilitera la vie pour effectuer des tests à partir de ghci ;

Calc.hs est le module principal, qui contient la fonction `main`, et gère l'interaction avec l'utilisateur.

Le format d'entrée de cette calculatrice est un format dit postfixe. Il a été choisi pour sa simplicité de traitement : une de ses qualités est de ne pas nécessité de parenthèse, alors que toute expression peut être saisie.

Le principe de ce format est simple : saisir le code du premier opérande ; celui du second ; puis enfin saisir l'opérateur.

Les deux premières questions visent à vous familiariser avec ce format d'entrée.

Il n'est pas nécessaire de comprendre en détail le fonctionnement du parseur pour finir le devoir. Seule la partie 2 rentre dans le détail de certains aspects du parseur. Le parseur est légèrement enrichi en partie 5.

Partie 1

1) Compiler le programme Calc.hs. Essayez-le sur les expressions suivantes :
- `3`
- `5 2 +`
- `6 3 - 2 *`
- `1 2 3 * +`

Essayez-le également sur vos propre expressions. Que remarquez-vous ?

2) Modifier la fonction `main` dans Calc.hs, de sorte que le programme affiche également l'expression à l'aide de la fonction `montre_exp`.

3) Ecrivez une fonction d'affichage de l'expression saisie `montre_exp_postfixe`, qui la montre dans le même format que le format d'entrée. Vous pourrez la placer dans le module Expression.hs. Intégrez-la à `main`.

4) Ecrivez une fonction d'affichage de l'expression saisie `montre_exp_prefixe`, qui la montre dans la syntaxe d'Haskell, avec les opérateurs en début d'expression. Par exemple :

— `5 2 +` → `( (+) 5 2)`
— `6 3 - 2 *` → `( (*) ( (-) 6 3) 2)`

Partie 2

Nous allons maintenant nous intéresser à certaines fonctions du parseur.

Le fonctionnement global du parseur est en deux étapes. Il découpe l'expression entrée sous forme de chaîne en une suite de *lexèmes*. Les lexèmes sont ce qui se trouve entre deux blancs : des nombres, ou des opérateurs. Puis cette suite de lexèmes est transformée en une unique expression par les fonctions `token_en_exp` et `traduction`. Il n'est pas demandé d'analyser ou de comprendre le détail de cette transformation.

1) Expérimentez, et décrivez l'effet d'un appel à la fonction `lis_entier`. Que fait-elle précisément ?

2) Que fait la fonction `mange_blancs` ? Expliquez en détail son fonctionnement.

Nous allons maintenant analyser quelques fonctions du module Expression.hs.

3) Quel est l'effet exact de la fonction `separe_gauche` ?

4) Quel est l'effet exact de la fonction `ajoute_au_premier` ?

5) Comment est utilisée la fonction `ajoute_au_premier` dans ce programme ?

Partie 3

1) Ecrire un module d'évaluation, dans un nouveau fichier Evaluation.hs, contenant une unique fonction `eval :: Exp -> Int`. Cette fonction prendra une expression, et calculera sa valeur. Ce module devra importer Types.hs, puisqu'il y a besoin de connaître la définition de Exp pour le compiler.

Importez ce module d'évaluation dans Exemples.hs, et testez son bon fonctionnement depuis ghci.

2) Adaptez le module Calc pour qu'il renvoie à présent, en plus d'une représentation de l'expression, la valeur de l'évaluation de cette expression.

Vous venez de finir la première version de la calculatrice (appelée 'V1' dans la suite.)

Partie 4

1) Démontrez que la version 1 ne peut pas manipuler correctement de grands entiers.

2) Copier le répertoire de la calcultrice obtenue à la question précédente dans un nouveau répertoire, qui sera donc la version 2 ('V2').

Adaptez la calculatrice de sorte qu'elle puisse maintenant faire des calculs de somme, différence et produit sur tous les entiers. Utilisez pour cela le type Integer de Haskell.

3) Illustrez par un exemple que cette version ne souffre plus de la limite de la V1 sur la taille des entiers.

Partie 5

1) Copier le répertoire de la V2 en une V3.

Dans le module Parseur.hs, ajouter une fonction : `filtre_flottant :: Char -> Bool` qui répond `True` sur les chiffres, et les symboles `'.'`, `'e'`, `'E'` et `'-'` ; et `False` pour tous les autres caractères.

2) Toujours dans le module Parseur.hs, écrire une fonction `lis_flottant` qui récupère un flottant au début de la chaîne passée en argument, et renvoie la chaîne représentant ce flottant et le restant de la chaîne. C'est un analogue à `lis_entier`, mais pour les flottants.

3) Adaptez la V3 pour qu'elle calcule sur les flottants. Illustrez son bon fonctionnement.

4) Ajoutez à la V3 l'opérateur de division.

5) (Question avancée, pour les plus motivés !)
Copier le répertoire V3 en V4. Vous allez maintenant optimiser le code de cette calculatrice qui évalue des expressions sur les nombres flottants. Nous ne voulons garder qu'une seule fonctionnalité : l'évaluation d'une expression arithmétique sur les nombres flottants.
Retirer du code toutes les routines qui ne sont pas utilisées pour cette fonctionnalité. Comptez le nombre de lignes de code restant dans cette version.
