



Práctica 1 - Parsers

Para resolver los siguientes ejercicios, usar la biblioteca `Parsing.lhs` que se encuentra en la página de la materia. El archivo `Parsing.lhs` debe estar en el mismo directorio que el archivo que se está editando y debe ser importado usando el comando:

```
import Parsing
```

1. Estudiar todas las primitivas derivadas del archivo `Parsing.lhs`. ¿Qué hace `sepBy`? ¿Qué hace `symbol`?
2. Extender el parser de expresiones visto en clase para permitir el uso de la resta y la división, basándose en la siguiente extensión a la gramática:

$$\begin{aligned} \text{expr} &\rightarrow \text{term} \ ('+' \text{ expr} \mid '-' \text{ expr} \mid \varepsilon) \\ \text{term} &\rightarrow \text{factor} \ ('*' \text{ term} \mid '/' \text{ term} \mid \varepsilon) \end{aligned}$$

3. Escribir un transformador que al recibir un parser, devuelva un nuevo parser que se comporta como el original pero que también acepta opcionalmente que las cadenas estén entre paréntesis.
4. Modificar el parser del ejercicio 2 para que en lugar de evaluar una expresión genere un árbol de sintaxis abstracta dado por el tipo:

```
data Expr = Num Int | BinOp Op Expr Expr
data Op = Add | Mul | Min | Div
```

5. Podemos modelizar una subfamilia de los tipos de datos de Haskell mediante el siguiente tipo de datos:

```
data Basetype = DInt | DChar | DFloat
type Hasktype = [Basetype]
```

Al tipo `Int → Char → Float` podemos representarlo como `[DInt, DChar, DFloat] :: Hasktype`. Escribir un parser para esta subfamilia de los tipos de Haskell.

6. Escribir un parser para listas heterogéneas de enteros y caracteres por extensión usando el formato de Haskell. Defina un tipo de datos adecuado para representar estas listas parseadas. Por ejemplo, una cadena a parsear es la siguiente:

```
[1, 'a', 'b', 2, 3, 'c']
```

7. Podemos modelizar otra subfamilia de los tipos de datos de Haskell, más expresiva que la del ejercicio 5, mediante el siguiente tipo de datos:

```
data Hasktype = DInt | DChar | DFloat | Fun Hasktype Hasktype
```

Por ejemplo el tipo `Int → Char → Float` será representado mediante el término `Fun DInt (Fun DChar DFloat)`. Escribir un parser para esta subfamilia de los tipos de Haskell.

Ayuda: previamente a la escritura del parser, recuerde escribir la gramática correspondiente.

8. Transformar la gramática para eliminar la recursión izquierda e implementar el parser para la gramática transformada.

$$\begin{aligned} \text{expr} &\rightarrow \text{expr} \ ('+' \text{ term} \mid '-' \text{ term}) \mid \text{term} \\ \text{term} &\rightarrow \text{term} \ ('*' \text{ factor} \mid '/' \text{ factor}) \mid \text{factor} \\ \text{factor} &\rightarrow \text{digit} \mid '(' \text{ expr} ') \\ \text{digit} &\rightarrow '0' \mid '1' \mid \dots \mid '9' \end{aligned}$$

9. La siguiente gramática es una simplificación de la declaración de tipos en C:

$$\begin{aligned} \text{declaration} &\rightarrow \text{type_specifier declarator ';' } \\ \text{declarator} &\rightarrow '*' \text{ declarator} \mid \text{direct_declarator} \\ \text{direct_declarator} &\rightarrow \text{direct_declarator} \ '[' \text{ constant_expression} \ ']' \mid '(' \text{ direct_declarator} ') ' \mid \text{identifier} \\ \text{type_specifier} &\rightarrow \text{'int'} \mid \text{'char'} \mid \text{'float'} \\ \text{constant_expression} &\rightarrow \text{number} \end{aligned}$$

Construir un parser para esta gramática y dar los tipos de datos adecuados para representar estas declaraciones.