Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Departamento de Ciencia de la Computación
Análisis de Lenguajes de Programación

# 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:

*expr* → *term* (`'+'` *expr* | `'-'` *expr* | $\varepsilon$)
*term* → *factor* (`'*'` *term* | `'/'` *term* | $\varepsilon$)

**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), mientras que el tipo (Int → Char) → Float, que no pertenece a la subfamilia del ejercicio 5, será representado con Fun (Fun DInt DChar) DFloat. Escribir un parser para esta subfamilia de 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 *expr* :: Parser Expr para la gramática transformada.

*expr* → *expr* (`'+'` *term* | `'-'` *term*) | *term*
*term* → *term* (`'*'` *factor* | `'/'` *factor*) | *factor*
*factor* → *digit* | `'('` *expr* `')'`
*digit* → `'0'` | `'1'` | ... | `'9'`

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

*declaration* → *type_specifier declarator* `';'`
*declarator* → `'*'` *declarator* | *direct_declarator*
*direct_declarator* → *direct_declarator* `'['` *constant_expression* `']'` | `'('` *direct_declarator* `')'` | *identifier*
*type_specifier* → `'int'` | `'char'` | `'float'`
*constant_expression* → *number*

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