Auxiliar 3

Substitución y representación de identificadores

P1. *Operadores binarios:*

Consideremos el lenguage visto hasta ahora en clases.

```
<expr> ::= <number>
         | {+ <Expr> <Expr>}
         | {- <Expr> <Expr>}
         | <symbol>
         | {with {<symbol> <Expr>} <Expr>}
```

(a) Añada las operaciones * y /.

(b) ¿Habrá algo que se pueda mejorar del diseño?

P2. *Ofuscación:*

Implemente la función obfuscate que cambia los identificadores de un programa <expr> a nombres aleatorios (use gensym para generar los nombres). La transformación debe respetar el alcance léxico. Ejemplo:

```
> (obfuscate (parse '{with {x 1} x}))
(with 'g108 (num 1) (id 'g108))

> (obfuscate (parse '{with {x 1} {with {y x} {+ x y}}}))
(with 'g109 (num 1) (with 'g110 (id 'g109) (add (id 'g109) (id 'g110))))
```

P3. *¿Son necesarios los nombres?*

Al definir la operación de substitución, uno de los problemas principales fue especificar correctamente su semántica. En especial, especificar lo que pasa cuando se usa el mismo identificador varias veces. Resulta que es posible deshacerse completamente de los nombres. Nicolaas Govert de Bruijn propuso reemplazar nombres por números, más bien, índices, que indican la profundidad de una asociación (binding).
Por ejemplo, en vez de escribir: `(with (x 5) (+ x x))`, podemos escribir: `(with 5 (+ <0> <0>))`. Por convención el alcance (scope) corriente es 0, entonces x se reemplazó por el índice <0>. Con esta representación, basta saber que la presencia de un with indica que entramos en un nuevo alcance. Así mismo, la siguiente expresión:

```
(with (x 5)                                  (with 5
    (with (y 4)          se convierte en:        (with 4
        (+ x y)))                                    (+ <1> <0>)))
```

(a) Cree un nuevo AST Expr-d que evidencie el cambio de representación.

(b) Implemente la función `toDeBruijn :: Expr -> Expr-d`.