Entorno local: uso de `let`

1. Introducción

Muchas veces resulta necesario crear valores temporarios (variables locales) dentro de funciones. La manera correcta de hacerlo es usando la expresión `let` ... `in` ... `end`.

Una forma simplificada de esta expresión, donde sólo aparecen declaraciones de valores, se muestra a continuación:

```
let
      val <1ra_variable>= <1era_expresion>;
      val <2da_variable>= <2da_expresion>;
      ...
      val <ultima_variable>= <ultima_expresion>
in
      <expresion>
end
```

Luego de `let` se ubican las declaraciones de variables. A continuación de `in` se escribe una expresión que puede usar las variables previamente definidas. Esta expresión también puede usar variables accesibles en el entorno donde la función que usa la expresión `let` es definida, siempre y cuando dichas variables no sean redefinidas temporalmente en las declaraciones que aparecen entre `let` e `in`. La palabra `end` indica el fin de la expresión.

Algunas consideraciones importantes a tener en cuenta:

- Los `;` (puntos y comas) luego de cada declaración son opcionales (para homogeneizar el código que escribiremos, optaremos por escribir `;` luego de cada declaración, excepto la última).
- Cada declaración de variable debe ir precedida por la palabra `val`.
- No debemos olvidarnos de incluir `in` y/o `end`, que son tan esenciales como la palabra `let`.

2. Definiendo Subexpresiones Recurrentes

La expresión `let` suele usarse cuando ciertas subexpresiones son recurrentes.

El siguiente ejemplo aclara este uso:

Ejemplo: Suponga que se quiere definir una función que calcula la 100-ésima potencia de un número `x`.

Podríamos escribir la expresión `x * x * ... * x` *(que contiene 100 veces* `x`*) si tuviésemos la paciencia para ello.*

Por suerte existe una forma de escribir menos. Veamos la siguiente función:

```
- fun potencia100(x:real) =
      let
            val cuatro = x * x * x * x;
            val veinte = cuatro * cuatro * cuatro * cuatro * cuatro
      in
            veinte * veinte * veinte * veinte * veinte
      end;
```

```
val potencia100 = fn : real ->real
```

Esta definición usa dos variables locales:

- `cuatro` *que equivale a* x^4
- `veinte` *que equivale a* `cuatro`^{5}*, o sea, a* x^{20}

3. Dividiendo el valor devuelto por una función

Otro uso importante de la expresión `let` es cuando se usa para dividir el valor devuelto por una función en sus partes componentes. En particular, cuando el valor devuelto por una función es una tupla se puede usar una forma más general de declaración de variables.

Por ejemplo, si una función `f` devuelve una 3-upla, podemos escribir:

```
val (a, b, c) = f ( ...
```

y así logramos que las 3 componentes se liguen a las variables `a`, `b` y `c`, respectivamente. Esta forma de escritura es usualmente más conveniente que:

```
val x = f ( ...
```

lo cual asocia la tupla completa a la variable `x` y luego debemos extraer los componentes usando los operadores `#i`, como por ejemplo:

```
val a = #1(x);
```

4. Definiendo funciones de alto orden

Una técnica útil para definir funciones de alto orden consiste en describir el comportamiento esperado por una función mediante el uso de una expresión `let` , dándole a dicha función un nombre, por ejemplo, `f`. Luego, entre las palabras `in` y `end` ubicamos a `f`.

Por ejemplo, para la función composición `comp`, usamos una expresión `let` para definir qué hace, en términos del parámetro `x`, una función composición de `F` y `G`.

```
- fun comp F G =
      let
            fun C x = G(F(x))
      in
            C
      end;
val comp = fn : ('a ->'b) ->('b ->'c) ->'a ->'c

- fun F x = x + 3;
val F = fn : int ->int

- fun G y = y*y + 2*y;
val G = fn : int ->int

- val H = comp F G;
val H = fn : int ->int
```

```
- H 10;
val it = 195: int
```