



## Práctica 3

1. Inferir, de ser posible, los tipos de las siguientes funciones:

- a) `modulus = sqrt · sum · map square;`
- b) `vmod [] = []`  
`vmod (v:vs) = modulus v : vmod vs`
- c) `smap 0 f = map f`  
`smap (n+1) f = map (smap n f)`

2. Dar el tipo de las funciones que se definen a continuación:

- a) `hd (x:xs) = x;`
- b) `tl (x:xs) = xs;`
- c) `pred (x+1) = x;`

3. ¿A qué valor reducen las siguientes expresiones, suponiendo evaluaciones *lazy*?

- a) `pred 0;`
- b) `tl [pred 0];`
- c) `hd (tl [pred 0]);`

4. Definir un tipo de `Colores` y una función `mezclar` que permita la combinación de los mismos.

5.

- a) Definir las operaciones de suma y producto módulo 2 para el tipo

```
data DigBin = Cero |Uno
```

- b) Definir las operaciones de suma binaria, producto por dos, cociente y resto de la división por dos para el tipo:

```
type NumBin = [Digbin]
```

donde convenimos que el primer elemento de la lista de dígitos es el dígito menos significativo del número representado.

- c) Redefinir las funciones del ítem anterior, observando una convención opuesta.
- d) Definir funciones que multipliquen números binarios de acuerdo a las dos convenciones.

6. Un `bag` (o `multiset`) es una colección de elementos. Cada elemento puede ocurrir un o más veces en el `bag`. Elija una representación eficiente para `bags`. Sería deseable que cada `bag` tenga una representación única. Definir las siguientes operaciones sobre `bags`, asegurándose de que aquellas que devuelvan `bags` lo hagan mediante una representación válida.

- a) `list2bag`, que toma una lista de elementos y devuelve un `bag` que contiene esos elementos. El número de ocurrencia de un elemento en la lista y en el `bag` resultante debe ser el mismo.

- b) `bagEmpty`, que devuelve `True` exactamente cuando el bag dado está vacío.
  - c) `bagCar`, que devuelve su cardinalidad (cantidad total de ocurrencias de todos los elementos).
  - d) `bagElem`, que devuelve verdadero si un elemento dado está en el bag dado;
  - e) `bagOccur`, que devuelve `True` si dos bags tienen los mismos elementos con la misma cantidad de ocurrencias.
  - f) `bagEqual`, que devuelve `True` si dos bags tienen los mismos elementos con la misma cantidad de ocurrencias.
  - g) `bagSubbag`, que devuelve `True` si el primer bag es subbag del segundo; se dice que `X` es subbag de `Y` si cada elemento de `X` ocurre en `Y` al menos tantas veces como en `X`.
  - h) `bagInter`, que toma dos bags y calcula el bag intersección. La intersección de dos bags `X` e `Y` contiene a todos los elementos comunes a `X` e `Y` con la menor cantidad posible de ocurrencias.
  - i) `bagSum`, que calcula el bag unión de dos bags dados; el bag suma de los dos bags `X` e `Y` consiste de todos los elementos de `X` e `Y` con número de ocurrencias de suma de las ocurrencias en cada uno de estos bags.
  - j) `bagInsert`, que toma un elemento y un bag y devuelve el bag con el elemento insertado.
  - k) `bagDelete`, que toma un elemento y un bag, y devuelve el bag con el elemento borrado.
7. Definir las operaciones de unión e intersección de dos conjuntos y el predicado de pertenencia para conjuntos de elementos de tipo `a` representados.
- a) por comprensión (como predicado `a → Bool`);
  - b) por extensión (como lista de elementos de `a`).