Paradigmas de Programación

Práctica 8

Nota Importante:

Todos los ejercicios de esta práctica son opcionales. Realice las implementaciones en los ficheros `curry.ml`, `comp.ml` y `poli.ml`.

Cuando se solicite la entrega de esta práctica, cada alumno deberá enviar únicamente estos ficheros.

Sea muy cuidadoso a la hora de crear los ficheros, y **respete los nombres indicados**.

Además, **estos ficheros deben compilar sin errores** con las siguientes órdenes:

```
ocamlc -c curry.ml
ocamlc -c comp.ml
ocamlc -c poli.ml
```

Ejercicios:

1. (Ejercicio opcional) **Curry y uncurry**. Dada una función $f : X \times Y \to Z$, podemos siempre considerar una función $g : X \to (Y \to Z)$ tal que $f(x, y) = (g\, x)\, y$.

   A esta transformación se le denomina "currificación" (*currying*) y decimos que la función g es la forma "currificada" de la función f (y que la función f es la forma "descurrificada" de la función g). A la transformación inversa se le denomina "descurrificación" (*uncurrying*).

   Defina en un fichero `curry.ml` una función

   ```
   curry : (('a * 'b) -> 'c) -> ('a -> ('b -> 'c))
   ```

   de forma que para cualquier función `f` cuyo origen sea el producto cartesiano de dos tipos, `curry f` sea la forma currificada de `f`.

   Y defina también la función inversa

   ```
   uncurry : ('a -> ('b -> 'c)) -> (('a * 'b) -> 'c)
   ```

   Una vez definidas estas dos funciones, prediga y compruebe (como en la práctica 1) el resultado de compilar y ejecutar las siguientes frases en OCaml:

   ```
   uncurry (+);;

   let sum = (uncurry (+));;

   sum 1;;

   sum (2,1);;

   let g = curry (function p -> 2 * fst p + 3 * snd p);;

   g (2,5);;
   ```

```
let h = g 2;;

h 1, h 2, h 3;;
```

Escriba las respuestas como comentarios en el mismo fichero.

2. (Ejercicio opcional) **Composición**. Defina en un fichero `comp.ml` la forma currificada de la composición de funciones:

   ```
   comp : ('a -> 'b) -> ('c -> 'a) -> ('c -> 'b)
   ```

   Una vez definida esta función, prediga y compruebe (como en la práctica 1) el resultado de compilar y ejecutar las siguientes frases en OCaml:

   ```
   let f = let square x = x * x in comp square ((+) 1);;

   f 1, f 2, f 3;;
   ```

   Escriba las respuestas como comentarios en el mismo fichero.

3. (Ejercicio opcional) **Polimorfismo**. Defina en un fichero `poli.ml` funciones con los siguientes tipos:

   - $f : 'a \to 'a$
   - $h : 'a * 'b \to 'a$
   - $i : 'a * 'b \to 'b$
   - $j : 'a \to 'a\ list$

   ¿Cuántas funciones se pueden escribir para cada uno de esos tipos? Escriba las respuestas como comentarios en el mismo fichero.