Paradigmas de Programación

Práctica 2

1. Se trata de analizar la serie de expresiones y definiciones OCaml incluidas en el fichero `expresiones2.pdf`. Para ello, abriremos el compilador interactivo de OCaml y, con cada expresión del fichero, haremos lo siguiente:

   - La escribimos en el fichero de texto `expresiones2.ml` utilizando un editor (por ejemplo, gedit).
   - Debajo, y usando comentarios `(*...*)`, intentamos predecir el resultado que dará OCaml sobre su compilación y ejecución, procurando usar la misma notación.
   - Copiamos la expresión en el terminal en el que tengamos abierto el compilador interactivo de OCaml y comprobamos el resultado. Si no es el previsto, lo corregimos e intentamos razonar por qué y en qué nos hemos equivocado.
   - Para toda expresión que produzca un error:
     i. La escribiremos en el fichero de texto entre comentarios.
     ii. Indicaremos, también entre comentarios, el tipo de error (léxico, sintáctico, de tipo o de ejecución) y la causa del mismo.
     iii. Trataremos de intuir la intención original de la expresión, le cambiaremos lo que sea necesario para que no produzca ningún error, y procederemos con esta nueva expresión igual que con el resto de expresiones correctas.
   - Usaremos el manual del lenguaje para averiguar el significado de los operadores y funciones que aparecen en cada expresión.

2. Escriba en un fichero de texto `expresiones3.ml` un programa OCaml que, al ejecutarlo, defina (en este orden):

   - Un valor `u` de tipo `int` a partir de una expresión que contenga, al menos, 4 operadores infijos.
   - Un valor `v` de tipo `float` a partir de una expresión que incluya una función predefinida.
   - Un valor `w` de tipo `char` a partir de una expresión que incluya una sub-expresión de tipo `int`.
   - Un valor `x` de tipo `bool` a partir de una expresión no trivial.
   - Un valor `y` de tipo `string` a partir de una expresión que contenga una estructura `if-then-else`.
   - Un valor `z` de tipo `int * int` a partir de una expresión no trivial.

3. (Ejercicio opcional) Escriba en un fichero de texto `fi.ml` un programa OCaml que, al ejecutarlo, defina un valor `fi` de tipo `(int -> int) -> (float -> float)`. Se valorará que no se use tipado explícito, que la función no sea constante y que no devuelva funciones constantes.

4. (Ejercicio opcional) Investigue con detalle el comportamiento de las funciones `Char.lowercase` y `Char.uppercase`, y defina con OCaml funciones `lowercase: char -> char` y `uppercase: char -> char`, de modo que tengan exactamente el mismo valor que `Char.uppercase` y `Char.lowercase`, respectivamente. Naturalmente, no pueden utilizarse en la definición ni `Char.uppercase` ni `Char.lowercase`. Realice las implementaciones en el fichero de texto `letters.ml`.