Paradigmas de Programación

Práctica 1

1. Construya un programa ejecutable `nombre` que escriba en la salida estándar dos líneas de texto: la primera con su nombre completo y la segunda con su dirección de correo electrónico en la UDC. El código fuente del programa, escrito en OCaml, debe guardarse en el fichero de texto `nombre.ml`.

2. Se trata de analizar la serie de expresiones OCaml incluidas en el fichero `expresiones.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 `expresiones.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:
     - La escribiremos en el fichero de texto entre comentarios.
     - Indicaremos, también entre comentarios, el tipo de error (léxico, sintáctico, de tipo o de ejecución) y la causa del mismo.
   - Usaremos el manual del lenguaje para averiguar el significado de los operadores y funciones que aparecen en cada expresión.
   - Es importante poner entre comentarios todo aquello que se pide explícitamente que se escriba así, porque el fichero `expresiones.ml` debe compilar (aunque obviamente si se genera el correspondiente programa ejecutable, éste no tendrá ningún efecto "visible", porque el fichero `expresiones.pdf` no contiene instrucciones de entrada/salida).

3. Escriba en un fichero de texto `expresiones2.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 que incluya una o más funciones u operadores.
   - Un valor `y` de tipo `string` a partir de una expresión que contenga una frase `if-then-else`.

4. En el lenguaje OCaml, las funciones `(&&) : bool -> bool -> bool` y `(||) : bool -> bool -> bool` implementan la cojunción y la disyunción booleanas.

A diferencia del resto de funciones en OCaml, la aplicación de estas funciones sigue una estrategia *lazy* (sólo se evalúa el "segundo" argumento si es necesario).

Es por ello, que es preferible ver las expresiones de la forma `<b1> || <b2>` como una abreviatura de la expresión `if <b1> then true else <b2>` (en vez de verlas como aplicación de funciones).

De modo análogo, las expresiones de la forma `<b1> && <b2>` deben ser vistas como una abreviatura de `if <b1> then <b2> else false`.

Al igual que hizo en el ejercicio 2, prediga y compruebe el resultado de compilar y ejecutar las siguientes frases en OCaml, escribiendo todo ello en el fichero de texto `condis.ml` (es decir, tanto las frases en sí mismas, como el resultado de su compilación y ejecución entre comentarios):

```
false && (2 / 0 > 0);;

true && (2 / 0 > 0);;

true || (2 / 0 > 0);;

false || (2 / 0 > 0);;

let con = (&&);;

let dis = (||);;

(&&) (1 < 0) (2 / 0 > 0);;

con (1 < 0) (2 / 0 > 0);;

(||) (1 > 0) (2 / 0 > 0);;

dis (1 > 0) (2 / 0 > 0);;
```

NOTA: Cuando se solicite la entrega de esta práctica, deberá enviar únicamente los ficheros `nombre.ml`, `expresiones.ml`, `expresiones2.ml` y `condis.ml`. Sea muy cuidadoso a la hora de crear los ficheros, y **respete los nombres indicados**. En particular, fíjese que todos estos nombres sólo contienen letras en minúsculas, números y puntos. Además, **todos los ficheros deben compilar**.