Paradigmas de Programación

Práctica 2

Ejercicios:

1. Se trata de analizar la serie de frases (presuntamente) en OCaml incluidas en el fichero `frases.pdf` que se proporciona junto con este mismo enunciado. Para ello, abriremos el compilador interactivo de OCaml y, con cada frase del fichero, haremos lo siguiente:

   - La escribimos en el fichero de texto `frases.ml` utilizando un editor de textos.
   - Seguidamente, intentaremos predecir la información que proporcionaría el compilador interactivo de OCaml (`ocaml`) sobre su compilación y ejecución, y lo escribimos como comentario en la línea siguiente, procurando usar la misma notación que `ocaml` (que distingue claramente expresiones de definiciones). En OCaml, los comentarios comienzan con `(*` y terminan con `*)`.
   - Por último, copiamos la frase 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 frase 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 dezconozcamos. En las nuevas versiones puede ser útil el *"Index to the library"* en la parte V (*"Index of values"* en versiones antiguas).
   - Es importante poner entre comentarios todo aquello que se pide explícitamente que se escriba así, porque el fichero `frases.ml` debe compilar (aunque obviamente si se genera el correspondiente programa ejecutable, éste no tendrá ningún efecto "visible", porque el fichero `frases.pdf` no contiene instrucciones de entrada/salida).

   El fichero `frases.ml` **debe compilar sin errores** con la siguiente orden:

   ```
   ocamlc -c frases.ml
   ```

2. Escriba en un fichero de texto `def.ml` un programa OCaml que defina (en este orden) los siguientes valores:

   - Un valor `v` de tipo `int` a partir de una expresión que contenga al menos una función predefinida.
   - Un valor `w` de tipo `float` a partir de una expresión que contenga al menos 4 operadores infijos.
   - Un valor `x` de tipo `char` a partir de una expresión que contenga una frase `if-then-else`.
   - Un valor `y` de tipo `bool` a partir de una expresión que contenga una o más funciones u operadores.
   - Un valor `z` de tipo `string` a partir de una expresión que contenga una sub-expresión de tipo `int`.

El fichero `def.ml` **debe compilar sin errores** con el fichero de interfaz `def.mli` proporcionado, mediante la siguiente orden:

```
ocamlc -c def.mli def.ml
```

3. Escriba en un fichero de texto `def2.ml` un programa OCaml que defina (en este orden) los siguientes valores:

   - Una función `p : float -> float` que haga corresponder a cada número no negativo el perímetro de la circunferencia que tenga como radio ese número (no importa lo que suceda con los negativos).
   - Una función `area : float -> float` que haga corresponder a cada número no negativo el área del círculo que tenga como radio ese número (no importa lo que suceda con los negativos).
   - Una función `absf : float -> float` que haga correponder a cada número su valor absoluto.
   - Una función `even : int -> bool` de modo que, al aplicarla a cualquier entero, el resultado que devuelva indique si el entero es par. Intente que la defínción sea lo más concisa posible (pista: intente evitar el uso de expresiones `if-then-else` innecesarias).
   - Una función `next3 : int -> int` que haga corresponder a cada entero el menor múltiplo de 3 que sea mayor o igual que él.
   - Una función `is_a_letter : char -> bool` que devuelva `true` en los caracteres de la `'a'` a la `'z'` (tanto mayúsculas como minúsculas) y `false` en los demás. A estos efectos, sólo consideraremos como "letras" los caracteres del alfabeto inglés (es decir, quedan excluidas la `'ñ'`, la `'ç'`, las letras con tilde, etc.).
   - Y por último, redefina la función `string_of_bool : bool -> string`, de modo que devuelva (adecuadamente) los valores `"verdadero"` o `"falso"` (puede hacerse una definición por casos o utilizarse una expresión `if-then-else`).

   El fichero `def2.ml` **debe compilar sin errores** con el fichero de interfaz `def2.mli` proporcionado, mediante la siguiente orden:

   ```
   ocamlc -c def2.mli def2.ml
   ```

Nota Importante:

Cuando se solicite la entrega de esta práctica, cada alumno deberá enviar únicamente los ficheros `frases.ml`, `def.ml`, y `def2.ml`.

Más adelante se darán indicaciones más precisas sobre la tarea de entrega y sobre el mecanismo de envío de los ficheros.

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 sin errores** con las órdenes anteriormente citadas.