Paradigmas de Programación

Práctica 2

Nota Importante:

Cuando se solicite la entrega de esta práctica, cada alumno deberá subir a su repositorio de prácticas (del cual se indicará su ubicación más adelante) un directorio `p2` cuyo contenido debe ser únicamente los ficheros `pi.ml`, `e.ml` y `fact.ml`.

Sea muy cuidadoso a la hora de crear el directorio y 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 siguientes órdenes:

```
ocamlc -o pi pi.ml
ocamlc -o e e.ml
ocamlc -o fact fact.ml
```

Ejercicios:

1. Escriba en OCaml un programa ejecutable `pi` que calcule una buena aproximación del número *pi* y muestre el resultado por la salida estándar seguido de un salto de línea. Guarde el código fuente en un archivo con nombre `pi.ml` (puede compilarlo con la orden `ocamlc -o pi pi.ml`). En este ejercicio intente que el código fuente no contenga el doble punto y coma (`;;`) que se utiliza para terminar las frases en el compilador interactivo.

   Ejemplo de ejecución:

   ```
   $ ./pi
   3.14159265359
   $
   ```

2. Escriba en OCaml un programa ejecutable `e` que calcule una buena aproximación del número *e* y muestre el resultado por la salida estándar seguido de un salto de línea. Guarde el código fuente en un archivo con nombre `e.ml` (puede compilarlo con la orden `ocamlc -o e e.ml`). En este ejercicio intente que el código fuente no contenga el doble punto y coma (`;;`) que se utiliza para terminar las frases en el compilador interactivo.

   Ejemplo de ejecución:

   ```
   $ ./e
   2.71828182846
   $
   ```

3. Estudie la siguiente definición escrita en OCaml:

   ```
   let rec fact = function
       0 -> 1
     | n -> n * fact (n - 1)
   ```

La palabra reservada `rec` después de `let` permite (en una definición de función) hacer referencia en la expresión a la derecha del signo `=` a la función que se está definiendo (es decir, permite definiciones recursivas de funciones).

El nombre (`n`) que aparece en la parte izquierda de la segunda regla del "*pattern-matching*" de la expresión lambda (es decir, de la frase `function ...`) es un patrón que encaja con cualquier posible valor del argumento de la función y sirve para hacer referencia a este valor en la parte derecha de la regla.

Compile esta definición en el *toplevel* (compilador interactivo) `ocaml` y compruebe su funcionamiento.

Utilizando esta función construya un programa ejecutable `fact` que muestre por la salida estándar (seguido de un salto de línea) el valor del factorial del número que se pase como argumento al invocar el programa. De modo que su ejecución podría verse como a continuación:

```
$ ./fact 10
3628800
$ ./fact 20
2432902008176640000
$ ./fact
fact: número de argumentos inválido
$
```

Para acceder desde el programa OCaml al argumento de la línea de comandos, puede utilizar el array de strings `Sys.argv` (que contiene las palabras usadas en la línea de comandos al invocar el programa). El elemento 0 de ese vector (`Sys.argv.(0)`) debe contener el nombre del programa invocado (en este caso, `fact`) y el elemento 1 (`Sys.argv.(1)`) debe contener el primer argumento (`10` en el primer ejemplo y `20` en el segundo). Para comprobar que el número de argumentos empleado al invocar el programa es correcto, puede aplicar la función `Array.length` al vector `Sys.argv` (en este caso debería devolver el valor 2, pues la línea de comando debería contener exactamente 2 palabras).

Guarde el código fuente del programa en un archivo con nombre `fact.ml`. Puede compilarlo con la orden `ocamlc -o fact fact.ml`.