



- Diseñe y codifique en SBCL funciones LISP según se indica en cada uno de los siguientes ejercicios. Valide siempre el tipo de datos de los argumentos.
- En cada caso respete las restricciones, indicadas entre corchetes, sobre el nombre, argumentos y comportamiento de cada función.

- 1) [*Collect*, argumentos un *predicado* y una *lista*. Recursiva] Devuelve una lista en la cual se encuentran todos los elementos del argumento original para los cuales se cumple el predicado del primer argumento.
- 2) [*Palindromo*, argumento una *lista*, Recursiva. Predicado] Si la *lista* recibida es un palíndromo, regresa *T*; de lo contrario regresa *NIL*.
- 3) [*2Palindrome*, argumento una *cadena*, Recursiva, No destructiva] Entrega como respuesta una cadena como la original, pero convertida en palíndromo (duplicándola en orden inverso al final de sí misma).
- 4) [*IterativePalindrome*, Iterativa] Modifique la función del ejercicio anterior para que opere de forma estrictamente iterativa.
- 5) [*ListRotate*, argumentos una *cadena*, un entero *n* y una de las dos llaves **:right** o **:left**] Devuelve una lista, de la misma longitud que la original, pero rotada *n* posiciones hacia la dirección indicada por el tercer argumento.
- 6) [*Max&Pos*, argumento un arreglo bidimensional global conteniendo números reales. Iterativa] Entrega una lista de asociación, de la forma
$$((col1 . row1) (col2 . row2) \dots (coln . rown))$$
en la cual cada asociación indica el renglón del arreglo en que se encuentra el mayor valor de la columna correspondiente.
- 7) [*Combine*, argumentos una función *func* y una *lista*. Recursiva] Efecto igual al operador *reduce*; combina todos los elementos de *lista* mediante la aplicación de la función *func*.

- 8) [*Level*, argumentos una *cadena* y una *lista* anidada] Si *cadena* se encuentra en *lista*, regresa el nivel de profundidad o anidamiento (comenzando en cero) en el que se encontró; de lo contrario regresa *NIL*.
- 9) [*StrEncode*, argumento una *cadena*. Recursiva] Devuelve una lista de asociación que indica el número de veces que se repite consecutivamente cada elemento de la lista original.
- 10) [*StrCypher*, argumentos una *cadena* y otra cadena *code* de longitud 27 donde cada posición corresponde a una letra del alfabeto (incluyendo la ñ)] Devuelve una cadena en la que cada carácter del argumento original fue substituído por el indicado en la posición correspondiente en la cadena *code*.
- 11) [*MatMult*, argumentos dos arreglos *m1* y *m2* de dos dimensiones y conteniendo valores numéricos] Averigua si las matrices son compatibles y, en caso de serlo, regresa otra matriz (un arreglo de dos dimensiones) con la multiplicación de *m1* y *m2*.
- 12) [*BTree*, argumentos un número *elem* y *tree* una lista anidada posiblemente vacía] Inserta *elem* en el árbol binario ordenado *tree* y regresa el nivel de profundidad en el que fue insertado. Si *elem* ya existía en el árbol o no puede ser insertado por alguna otra razón, la función devuelve *NIL*.
- 13) [*FilterSubsets*, argumento una *lista* y una posición *pos* en dicha lista] Calcula y entrega una lista conteniendo, todos los subconjuntos de *lista* que contienen al elemento en la posición *pos*.
- 14) [*Subsets*, argumentos una *lista* y un entero *k*] Calcula y entrega una lista conteniendo, en forma de listas, todos los subconjuntos, de cardinalidad *k*, del conjunto *lista*.
- 15) [*If-positive*, Macro] Escriba una macro para una estructura algorítmica condicional que evalúe una expresión numérica y tome acciones distintas en

caso de que la expresión evalúe en un número positivo y en caso contrario. La sintaxis para uso de esta macro debe ser:

```
(if-positive <expresión>  
  :then-do <instrucciones>  
  ...  
  :else-do <instrucciones>  
  ...)
```