**8.**En tu código, parece que estás tratando de definir una función predicado llamada ordenado que determina si una lista numérica está ordenada en forma ascendente. Sin embargo, **hay un problema en la lógica de la función** **que podría generar un error cuando la lista está vacía o cuando llegas al final de la lista sin encontrar un par de elementos desordenados.** (Osea, cuando al lista esta ordenada)

**El error en tu código se produce cuando llegas al caso base en el que la lista está vacía *((endp lista))***. En ese caso, estás devolviendo T, lo cual es correcto. **Sin embargo**, cuando la lista no está vacía, no estás manejando el caso en el que los elementos no están ordenados correctamente**. Deberías devolver NIL en ese caso para indicar que la lista no está ordenada.**

Aquí tienes una versión corregida de tu función ordenado:

**(defun ordenado (lista)**

**(cond**

**((or (endp lista) (endp (cdr lista))) 'T) ;** *Caso base: cuando la lista está vacía o tiene un solo elemento, está ordenada*

**((<= (car lista) (cadr lista)) (ordenado (cdr lista))) ;** *Llamada recursiva si los elementos están ordenados correctamente*

**(t NIL)))** *; Devuelve NIL si los elementos no están ordenado*

**s**En esta versión corregida, **se agregó un nuevo caso a la construcción cond** para manejar el escenario en el que los elementos no están ordenados correctamente. Si (<= (car lista) (cadr lista)) es verdadero (lo que significa que los dos primeros elementos están en orden ascendente), entonces se hace una llamada recursiva con el resto de la lista ((ordenado (cdr lista))). Si no se cumple esa condición, se devuelve NIL para indicar que la lista no está ordenada correctamente.

**En el código original corregido,** la condición **(< (car lista) (car (cdr lista)))** se utiliza para verificar si los elementos de la lista están ordenados correctamente. **Sin embargo**, esta condición no tiene en cuenta el caso en el que la lista tiene **menos de dos elementos**.

Cuando intentas acceder al segundo elemento de la lista utilizando **(car (cdr lista)),** si la lista tiene solo un elemento, **(cdr lista)** devolverá una lista vacía (NIL). Por lo tanto, al aplicar (**car NIL**), se produce un error porque **no se puede obtener el primer elemento de una lista vacía.**

Para solucionar esto, es necesario verificar si la lista tiene al menos dos elementos antes de aplicar (car **(cdr lista)).** Esta verificación se puede realizar utilizando la condición **(or (endp lista) (endp (cdr lista))**), que devuelve T si la lista está vacía o tiene solo un elemento

En el código corregido que te proporcioné en la respuesta anterior, se agregó esta verificación adicional en el caso base de la función ordenado. **Esto asegura que cuando la lista tiene menos de dos elementos, se considera como ordenada y se devuelve T en lugar de intentar acceder a elementos que no existen.**

11. ELEMENTO REPETIDO… ESTA MAL

**(defun eliminar-repetidos (lista)**

**(cond**

**((endp lista) '()) ;** *Caso base: lista vacía, devuelve una lista vacía*

**((member (car lista) (cdr lista)) ;** *Verifica si el primer elemento está repetido en el resto de la lista*

**(cons (car lista) (eliminar-repetidos (remover-elemento (car lista) (cdr lista)))))** *; Si está repetido, agrega el elemento a la lista resultante y llama recursivamente removiendo todas las ocurrencias*

**(t ; (cons (car lista) (eliminar-repetidos (cdr lista)))))) ;***Si no está repetido, agrega el primer elemento a la lista resultante y llama recursivamente con el resto de la lista*

**(defun remover-elemento (elemento lista)**

**(cond**

**((endp lista) '()) ;** *Caso base: lista vacía, devuelve una lista vacía*

**((equal elemento (car lista)) ;** *Verifica si el elemento es igual al primer elemento de la lista*

**(remover-elemento elemento (cdr lista))) ;** *Si es igual, llama recursivamente sin incluir el primer elemento*

**(t ; (cons (car lista) (remover-elemento elemento (cdr lista)))))) ;***Si no es igual, agrega el primer elemento a la lista resultante y llama recursivamente con el resto de la lista*

En esta versión, se ha modificado la función eliminar-repetidos para llamar a remover-elemento antes de la llamada recursiva, asegurándose de que todas las ocurrencias del primer elemento repetido se eliminan antes de continuar con la recursión. Esto garantiza que solo se mantenga una ocurrencia de los números repetidos en la lista resultante.

En el ejemplo de uso, se pasa una lista con elementos repetidos (1 2 3 2 4 1 5 3). Al llamar a eliminar-repetidos sobre esa lista, se obtendrá (1 2 3 4 5), que es la nueva lista donde cada número repetido aparece solo una vez.

10. ENTEROS

**(defun ejer10 (lista)**

**(cond**

**((null lista) (list '() '())) ; Devuelve una lista con dos sublistas vacías en caso de lista vacía**

**((integerp (car lista))**

**(setq sublistas (ejer10 (cdr lista)))**

**(list (cons (car lista) (car sublistas)) (cadr sublistas)))**

**(t**

**(setq sublistas (ejer10 (cdr lista)))**

**(list (car sublistas) (cons (car lista) (cadr sublistas))))))**

La función **ejer10** toma una lista como argumento y utiliza recursión para separar los números enteros y los elementos que no son números enteros en dos sublistas distintas. Veamos el código en detalle:

1. La función **ejer10** se define utilizando **defun** y toma un parámetro **lista**.
2. Se utiliza **cond** para evaluar diferentes condiciones en orden. El **cond** tiene tres cláusulas separadas por paréntesis:
   * La primera cláusula verifica si la lista es nula. Si es así, se devuelve una lista que contiene dos sublistas vacías mediante **(list '() '())**. Esto se hace para manejar el caso base de la recursión cuando no quedan elementos en la lista original.
   * La segunda cláusula se evalúa si el primer elemento de la lista es un número entero utilizando **(integerp (car lista))**. Si es así, se ejecutan los siguientes pasos:
     + Se realiza una llamada recursiva a **ejer10** con el resto de la lista **(cdr lista)** y se almacena el resultado en la variable **sublistas** utilizando **(setq sublistas (ejer10 (cdr lista)))**.
     + Se construye una nueva lista utilizando **list**. El primer elemento de la lista original se agrega a la primera sublista con **(cons (car lista) (car sublistas))**, y la segunda sublista se toma directamente de **sublistas** utilizando **(cadr sublistas)**.
   * La tercera cláusula se evalúa si ninguna de las condiciones anteriores se cumple, lo que significa que el primer elemento de la lista no es un número entero. Se ejecutan los siguientes pasos:
     + Se realiza una llamada recursiva a **ejer10** con el resto de la lista **(cdr lista)** y se almacena el resultado en la variable **sublistas** utilizando **(setq sublistas (ejer10 (cdr lista)))**.
     + Se construye una nueva lista utilizando **list**. El primer elemento de la lista original se agrega a la segunda sublista con **(cons (car lista) (cadr sublistas))**, y la primera sublista se toma directamente de **sublistas** utilizando **(car sublistas)**.
3. Al final de cada cláusula, se utiliza **list** para combinar las sublistas y se devuelve como resultado de la función.

En resumen, la función **ejer10** procesa recursivamente la lista original y devuelve una lista que contiene dos sublistas separadas: una con los números enteros y otra con los elementos que no son números enteros.