***TP N°10: Listas***

| Salvo que se indique lo contrario, en cada ejercicio que se haga referencia a una lista se deberá usar el tipo TList según la siguiente definición:    **typedef​** struct node \* TList;  **typedef​** struct node {  int elem;  struct node \* tail;  } TNode;  y donde una lista vacía se representa con el valor ​NULL​. |
| --- |

**No usar funciones ni macros auxiliares.**

**No usar variables static**

**No usar ciclos dentro de las funciones**

***Ejercicio 1 (nivel principiante)***

1. Escribir una función recursiva sumAll que reciba una lista de enteros y retorne la suma de sus elementos
2. Escribir una función recursiva odds que reciba una lista de enteros y retorne 1 si todos sus elementos son impares, y cero si no. Escribir dos versiones:

* Considerar que la lista vacía tiene todos sus elementos impares
* Considerar que la lista vacía no tiene elementos impares

***Ejercicio 2***

Agregar a la biblioteca de funciones de listas ordenadas de enteros vista en clase las siguientes funciones recursivas, sin usar funciones auxiliares:

* TList delete(TList list): elimina el elemento de la lista, si es que el elemento está
* int get(const TList list, unsigned int i): retorna el i-ésimo elemento de la lista, donde el primer elemento está en la posición cero. Esta función sirve para darle funcionalidad de vector a una lista. Si i es mayor o igual a la cantidad de elementos la función debe abortar ejecutando exit(1).

**Ejercicios de Parcial**

***Ejercicio 3***

Escribir una **función recursiva** **order** que dada una lista de enteros que debería estar ordenada en forma ascendente, elimine de la misma los elementos que no cumplan con ese orden.

El primer elemento de la lista (si no está vacía) no cambia.

Ejemplos:

* si la lista tiene los elementos 1, 3, 4, 5, 8, 10 no cambia
* si la lista tiene los elementos 1, 1, 3, 2, 4, 5, 8, 10,1 pasa a ser 1, 3, 4, 5, 8,10
* si la lista tiene los elementos 12, 1, 3, 4, 5, 8, 10 pasa a ser la lista con el elemento 12

***Ejercicio 4***

Escribir una función recursiva **addAll** que reciba dos listas donde cada elemento es menor al siguiente (no acepta repetidos), y agregue a la primer lista los elementos de la segunda que no estén en la primera.

***Ejercicio 5***

Escribir la función recursiva **restaList** que reciba únicamente dos listas de enteros ordenadas **en forma ascendente y sin repetidos**, y retorne una nueva lista ordenada y sin repetidos con los elementos de la primer lista que no están en la segunda lista.

***Ejercicio 6***

Escribir una función recursiva **deleteDupl** que reciba como único parámetro una lista ordenada pero con repetidos y elimine de la misma las repeticiones.

Por ejemplo si la lista es {1, 3, 3, 3, 4, 5, 5, 6, 6, 6} la lista quede {1, 3, 4, 5, 6}

***Ejercicio 7***

Escribir una función recursiva **listIntersec** que reciba como únicos parámetros dos listas ordenadas ascendentemente y retorne una nueva lista con la intersección de ambas.

***Ejercicio 8***

Escribir una función recursiva **listUnion** que reciba como únicos parámetros dos listas ordenadas ascendentemente y retorne una nueva lista con la unión de ambas.

***Ejercicio 9***

Escribir la función recursiva **comprimeList** que reciba una lista ordenada **donde cada elemento es menor o igual al siguiente (puede tener repetidos)**, y retorne una nueva lista de tipo TListBrief ordenada donde en cada nodo se indica la cantidad de veces que aparece el elemento.

**typedef​** struct nodeBrief \* TListBrief;

**typedef​** struct nodeBrief {

int elem;

size\_t count;

struct nodeBrief \* tail;

} TNodeBrief;

**Ejemplo:**

Si la lista original tiene los elementos 1, 2, 2, 2, 5, 6, 6, 7, 7, 7, 7, 7

debe retornar una lista con los nodos {1,1}, {2,3}, {5,1}, {6,2}, {7,5}