



## Tema 2 - Tipos Abstractos de Datos Lineales

### Hoja de Problemas 1

#### Problemas - Listas simplemente enlazadas.

Dada la clase SList (con `_head` y `_tail`), implementación del TAD Lista basada en lista simplemente enlazada, crea un subclase SList2 e implementa las siguientes funciones:

- `remove(e)`: método que recibe un elemento, `e`, y borra la primera ocurrencia de `e` en la lista (es decir, elimina el primer nodo que contiene el elemento `e`). El método modifica la lista y no devuelve nada. Si el elemento no existe en la lista, el método debe mostrar un mensaje indicando que el elemento no existe.
- `remove_all(e)`: función que recibe un elemento, `e`, y borra todas las ocurrencias de `e` en la lista (es decir, elimina todos los nodos que contienen a `e`). El método modifica la lista y no devuelve nada. Si el elemento no existe en la lista, el método debe mostrar un mensaje indicando que el elemento no existe.
- `getAtRev(index)`: método que recibe un índice, `index`, y devuelve el elemento en la posición `index` empezando por el final. Por ejemplo:
  - l: 0->1->2->3->4, `l.getAtRev(0)=4`, `l.getAtRev(1)=3`, `l.getAtRev(2)=2`, `l.getAtRev(3)=1`, `l.getAtRev(4)=0`,  
El método debe comprobar que el índice es correcto (es una posición de la lista). en caso de no ser un índice correcto, imprime un mensaje y devuelve `None`.
- `get_middle()`: método que devuelve el elemento que está en la mitad de la lista. Si la lista tiene un número par de elementos, la función devolverá el elemento en la posición `len(l)//2`. Ejemplo: 1->2->3->4->5->6, `l.getMiddle()=3`
- `count(e)`: método que recibe un elemento, `e`, y devuelve el número de veces que ocurre en la lista. Si el elemento no existe en la lista, la función devuelve 0.
- `is_sorted()`: método que comprueba si la lista está ordenada de forma ascendente (en este caso devuelve `True`). En caso contrario, debe devolver `False`.
- `remove_duplicates_sorted()`: método que borra los elementos duplicados en una lista ordenada. El método modifica la lista, no devuelve nada.  
Ejemplo: l: 1->1->2->3->3->4->5->5, l: 1->2->3->4->5.
- `remove_duplicates()`: método que borra los elementos duplicados en una lista (no tiene que estar ordenada). La función modifica la lista, no devuelve nada.

Ejemplo: l: 1->2->1->0->2->6->6->4->5->5, l: 1->2->0->6->4->5.

- `swap_pairwise()`: método que intercambia los elementos que ocupan posiciones contiguas. La función modifica la lista, no devuelve nada. Ejemplos:  
l: 1->2->3->4->5, l: 2->1->4->3->5  
l: 1->2->3->4->5->6, l: 2->1->4->3->6->5
- `move_last()`: método que mueve el último elemento al principio de la lista, sin usar ninguna del resto de métodos de la clase. El método modifica la lista, no devuelve nada. Ejemplo: l: 1->2->3->4->5->6, l: 6->1->2->3->4->5
- `intersection(l2)`: método que recibe una lista l2 y devuelve una nueva lista que contenga la intersección de ambas listas, la invocante y l2. Como precondition, se exige que ambas listas están ordenadas de forma ascendente. Ejemplo: l: 1->2->3->4->5->6, l2: 0->1->2->3, output: 1->2->3.
- `segregate_odd_even()`: método que modifica la lista invocante para que todos los elementos pares aparezcan antes que los elementos impares. El método debe respetar el orden de los elementos pares y el orden de los elementos impares. El método modifica la lista, y no devuelve nada.  
Ejemplo: l: 17->15->8->12->10->5->4->1->7->6  
l: 8->12->10->4->6->17->15->5->1->7

Nota. Muchos de estos métodos pueden implementarse de forma sencilla si usas otros métodos de la clase `SList`. Sin embargo, en algunas ocasiones el uso de estos métodos aumenta la complejidad temporal del método final. Recuerda que tus soluciones siempre deberían ser lo más eficientes posibles en términos de complejidad temporal.