# Ejercitación TAD PILA (Stack) y COLA (Queue)

## Ejercicio 1

Crear un módulo de nombre **mystack.py** que **implemente** las siguientes especificaciones de las operaciones elementales para un **TAD Pila** utilizando el TAD Lista. Recordar que una Pila puede implementarse también sobre una estructura **LinkedList** donde, el último elemento en ingresar a la lista es el primero en salir (LIFO).

**push(S,element)**

**Descripción:** Agrega un elemento al comienzo de S, siendo S una estructura de tipo LinkedList
**Entrada:** La pila S sobre la cual se quiere agregar el elemento (LinkedList) y el valor del elemento (element) a agregar.
**Salida:** No hay salida definida

**pop(S)**

**Descripción:** extrae el primer elemento de la pila S, siendo S una estructura de tipo LinkedList
**Poscondición:** Se debe desvincular el **Node** a eliminar.
**Entrada:** la pila S (Linkedlist) sobre el cual se quiere realizar la eliminación
**Salida**: Devuelve el elemento eliminado. Devuelve **None** si la pila está vacía.

## Ejercicio 2

Crear un módulo de nombre **myqueue.py** que **implemente** las siguientes especificaciones de las operaciones elementales para un **TAD Cola** utilizando el TAD Lista. Recordar que una Cola puede implementarse también sobre una estructura **LinkedList** donde, el primer elemento en ingresar a la lista es el primero en salir (FIFO).

**enqueue(Q,element)**

**Descripción:** Agrega un elemento al comienzo de Q, siendo Q una estructura de tipo LinkedList.
**Entrada:** La cola Q (LinkedList) sobre la cual se quiere agregar el elemento y el valor del elemento (element) a agregar.
**Salida:** No hay salida definida.

**dequeue(Q)**

**Descripción:** extrae el último elemento de la cola Q, siendo Q una estructura de tipo LinkedList.
**Poscondición:** Se debe desvincular el **Node** a eliminar.
**Entrada:** la cola Q (Linkedlist) sobre el cual se quiere realizar

la eliminación.
**Salida:** Devuelve el elemento de la cola. Devuelve **None** si la cola está vacía.

## Ejercicio 3

A partir de las estructuras definidas como:

```
class PriorityQueue:
    head=None

class PriorityNode:
    value=None
    nextNode=None
    priority=None
```

Crear un módulo de nombre **mypriorityqueue.py** que implemente una cola con prioridad. Una cola con prioridad es un **TAD** similar a una cola en la que los elementos tienen adicionalmente, una prioridad asignada. En una cola de prioridades un elemento con mayor prioridad será encolado antes que un elemento de menor prioridad. Si dos elementos tienen la misma prioridad, se desencolarán siguiendo el orden de cola.

**enqueue_priority(Q,element,priority)**
**Descripción:** Agrega un elemento a Q con la prioridad **priority** (entero), siendo Q una estructura de tipo PriorityQueue
**Entrada:** La cola Q sobre la cual se quiere agregar el elemento (PriorityQueue), el valor del elemento (element) a agregar y un número que indica la prioridad.
**Salida:** Devuelve la posición donde se inserto el elemento.

**dequeue_priority(Q)**
**Descripción:** extrae el primer elemento de la cola Q con la mayor prioridad (un valor mayor del campo **priority,** indica una mayor prioridad), siendo Q una estructura de tipo PriorityQueue
**Poscondición:** Se debe desvincular el **Node** a eliminar.
**Entrada:** la cola sobre el cual se quiere realizar la eliminación (PriorityQueue)
**Salida:** Devuelve el elemento con mayor prioridad. Devuelve **None** si la cola está vacía.

**A tener en cuenta:**

1. Cada operación básica debe ser implementada como una función.
   Ejemplo:
   ```
   def enqueue(LinkedList,element,priority):
       <...Código que implementa la operación enqueue...>
   ```
2. Las operaciones deben respetar la especificación propuesta. Es decir solo incluir los parámetros mencionados en la definición de la operación.

3. Se sugiere usar lápiz y papel primero
4. **No se puede utilizar otra Biblioteca más allá de:**
    a. **algo1.py**
    b. **myarray.py**
    c. **mylinkedlist.py**