**Antes de pasar a la página siguiente...**
**¿Has iniciado la grabación del screencast?**

# Partición de una lista

Simulacro de examen - Estructuras de Datos - Grupo F
Facultad de Informática - UCM

Este ejercicio debe entregarse al problema *DOMjudge* on identificador `SIM1`. Los datos de acceso son los siguientes:

- URL: `http://ed.fdi.ucm.es/`
- Concurso: `ED-F-EV` (puedes seleccionarlo en la esquina superior derecha de la página de *DOMJudge*, al lado del botón *Logout*)
- Identificador de problema: `SIM1`

Debes entregar:

1. El fichero `.cpp` con el código fuente de tu solución. Para ello utiliza la plantilla que se proporciona con este enunciado. El código fuente se entrega en *DOMjudge*. Para ello dispones de **1 hora** desde el comienzo del examen. Se evaluará la última entrega realizada antes de la hora límite.
2. Un breve video explicativo (3-5 minutos) de la solución realizada. Este video se entrega a través de la carpeta de *Google Drive* que se ha compartido contigo. Para ello dispones de **30 minutos** una vez finalizado el plazo de entrega del código fuente.
3. El vídeo con el *screencast* de la realización de la prueba. Este video se entrega a través de la carpeta de *Google Drive* que se ha compartido contigo. Para ello dispones de **1 hora** una vez finalizado el plazo de entrega del código fuente.

Recuerda que **no se permite copiar** en la entrega **código fuente externo**, salvo que provenga de la plantilla proporcionada.

Enlace a las instrucciones:
`https://drive.google.com/open?id=1y50P6GtroTeru8-f3TDjn6d8fasIW_9sHNLvMaI5aKs`.

Este ejercicio consiste en extender (mediante herencia) la clase `deque<T>`, la cual implementa el TAD doble cola mediante una lista doblemente enlazada circular con nodo fantasma. Ha de añadirse un nuevo método `partition()` a la clase derivada:

```
template<typename T>
class deque_partition : public deque<T> {
  ...

  private:
    void partition(int pivot);
};
```

El método `partition()` recibe un número entero `pivot`, y debe separar a un lado de la lista aquellos elementos que sean menores (o iguales) que `pivot`, y al otro lado de la lista aquellos elementos que sean estrictamente mayores que `pivot`.

Por ejemplo, supongamos la lista `xs = [5, 10, 9, 7, 4, 6]`. Tras la llamada a `xs.pivot(8)` la lista debe quedar del siguiente modo: `[5, 7, 4, 6, 10, 9]`. Es decir, al principio están todos los elementos

menores o iguales a 8 seguidos de todos los elementos de la lista mayores que 8. Observa que se ha preservado el orden relativo entre los elementos que son menores o iguales a 8. Esto es, si el 5 estaba antes a la izquierda del 7 en la lista original, también debe ser así en la lista ordenada. Lo mismo ocurre con los números que son mayores que 8.

**Importante:** Para la implementación del método no pueden crearse nuevos nodos mediante `new`, ni liberar nodos mediante `delete`; deben reutilizarse los nodos de la lista enlazada. Tampoco se permite copiar valores de un nodo a otro. El coste de la operación debe ser lineal con respecto al número de elementos de la lista de entrada.

## Entrada

La entrada comienza con un número que indica el número de casos de prueba que vienen a continuación. Cada caso de prueba consiste en dos lineas. La primera línea contiene una lista, representada mediante una secuencia de números comprendidos entre `1` y `5000`, finalizando con un `0` que no se considera parte de la lista. La segunda línea contiene un número entero, que es el valor del parámetro `pivot` que debe utilizarse para esa lista.

## Salida

Para cada caso se imprimirá una línea con el contenido de la lista tras aplicar el método `partition()`. Los elementos de la lista deben de estar separados por un carácter de espacio.

## Entrada de ejemplo

```
7
5 10 9 7 4 6 0
8
5 10 9 7 4 6 0
6
5 10 9 7 4 6 0
3
5 10 9 7 4 6 0
20
0
9
4 0
4
4 0
6
```

## Salida de ejemplo

```
5 7 4 6 10 9
5 4 6 10 9 7
5 10 9 7 4 6
5 10 9 7 4 6

4
4
```