Córdoba 2 de Junio del 2022

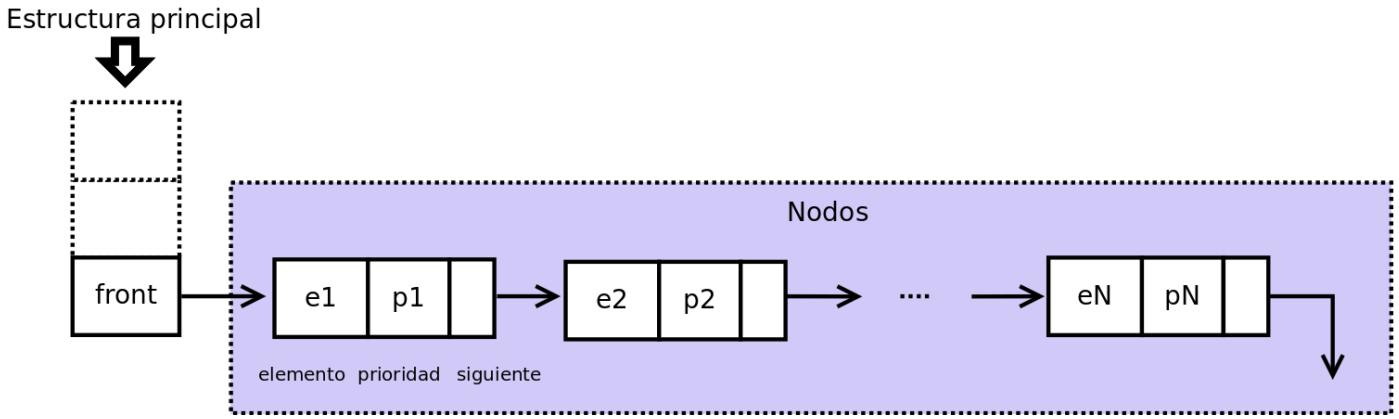
# Segundo Parcial de Laboratorio

Algoritmos y Estructura de Datos II

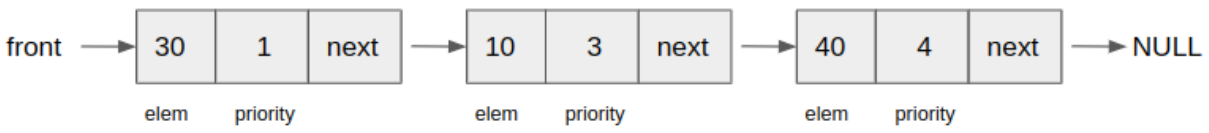
## TEMA A

### Ejercicio

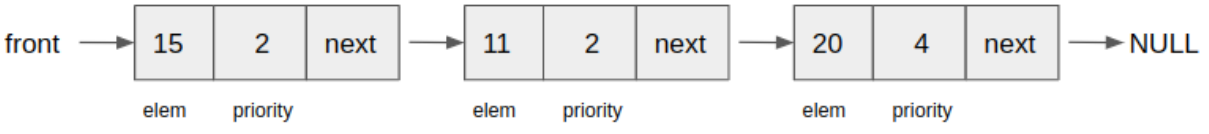
Implementar el TAD ***pqueue*** que representa una cola de prioridades. Una cola de prioridades (**pqueue**) es un tipo especial de cola en la que cada elemento está asociado con una prioridad asignada. En una cola de prioridades un elemento con mayor prioridad será desencolado antes que un elemento de menor prioridad. Sin embargo, si dos elementos tienen la misma prioridad, se desencolarán siguiendo el orden de la cola. En este examen vamos a implementar **pqueue** usando lista enlazadas y una estructura principal:




En la representación, **`e1`** es el primer elemento de la cola y **`p1`** tiene la mayor prioridad. En realidad la propiedad fundamental es que **`p1`** ≤ **`p2`** ≤ … ≤ **`pN`** (notar que aquí la prioridad representada con 0 es la mayor prioridad, y números más grandes establecen prioridades más bajas). Además, si siempre se usa la misma prioridad, el TAD ***pqueue*** debe comportarse exactamente igual a una cola común (FIFO: first input, first output). Algunos ejemplos:




En esta ***pqueue***, el próximo elemento a desencolar es 30 ya que tiene prioridad 1 (la más prioritaria).



En este ejemplo el próximo elemento a desencolar es 15 ya que aunque tiene la misma prioridad que el elemento 11, fue agregado primero el 15 a la cola. **Lo importante es mantener la estructura de nodos bien ordenada para desencolar el elemento correcto**.

El TAD ***pqueue*** tiene la siguiente interfaz

| Función | Descripción |
|---|---|
| `pqueue pqueue_empty(void)` | Crea una cola de prioridades vacía |
| `pqueue pqueue_enqueue(pqueue q, pqueue_elem e, unsigned int priority);` | Inserta un elemento a la cola con su correspondiente prioridad. |
| `bool pqueue_is_empty(pqueue q);` | Indica si la cola de prioridades está vacía |
| `unsigned int pqueue_size(pqueue q)` | Obtiene el tamaño de la cola de prioridades |
| `pqueue_elem pqueue_peek(pqueue q)` | Obtiene el elemento con mayor prioridad |
| `unsigned int pqueue_peek_priority(pqueue q)` | Obtiene el valor de la prioridad del elemento con mayor prioridad. |
| `pqueue pqueue_dequeue(pqueue q)` | Quita un elemento con mayor prioridad más antiguo de la cola |
| `pqueue pqueue_destroy(pqueue q)` | Destruye una instancia del TAD Cola de prioridades |

En `pqueue.c` se da una implementación incompleta del TAD ***pqueue*** que deben completar **siguiendo la representación explicada anteriormente.** Además deben asegurar que la función `pqueue_size()` sea de orden constante ( *O(1)* ). Por las dudas se aclara que no es necesario que la función `pqueue_enqueue()` sea de orden constante ya que puede ser muy complicado lograrlo para la representación que se utiliza y no recomendamos intentarlo.

Para verificar que la implementación del TAD funciona correctamente, se provee el programa (**main.c**) que toma como argumento de entrada el nombre del archivo cuyo contenido sigue el siguiente formato:

| <patient_id> | <priority> |
|---|---|

El archivo de entrada representa una lista de pacientes con prioridades de atención médica. Entonces el programa lee el archivo, carga los datos en el TAD ***pqueue*** y finalmente muestra por pantalla la cola de prioridades de los pacientes.

**El programa resultante no debe dejar *memory leaks ni lecturas/escrituras inválidas*.**

Una vez compilado el programa puede probarse ejecutando:

```
$ ./dispatch_patients inputs/hospital_a.in
```

Obteniendo como resultado:

```
length: 7
[ (454, 1), (456, 2), (345, 3), (686, 4), (234, 5), (234, 6), (789, 8)]
```

Otro ejemplo de ejecución:

```
$ ./dispatch_patients inputs/hospital_b.in
length: 7
[ (100, 0), (200, 0), (300, 0), (400, 0), (500, 0), (600, 0), (700, 0)]
```

Otro ejemplo:

```
$ ./dispatch_patients inputs/hospital_c.in
length: 7
[ (234, 4), (456, 4), (789, 4), (686, 4), (454, 4), (234, 5), (345, 6)]
```

Consideraciones:

- Se provee el archivo **`Makefile`** para facilitar la compilación.
- Se recomienda usar las herramientas **`valgrind`** y **`gdb`**.
- Si el programa no compila, no se aprueba el parcial.
- Los *memory leaks* bajan puntos
- Entregar código muy improlijo puede restar puntos
- Si **`pqueue_size()`** no es de orden constante baja muchísimos puntos
- Para promocionar **se debe** hacer una invariante que chequee la propiedad fundamental de la representación de la cola de prioridades.