Córdoba 2 de Junio del 2022

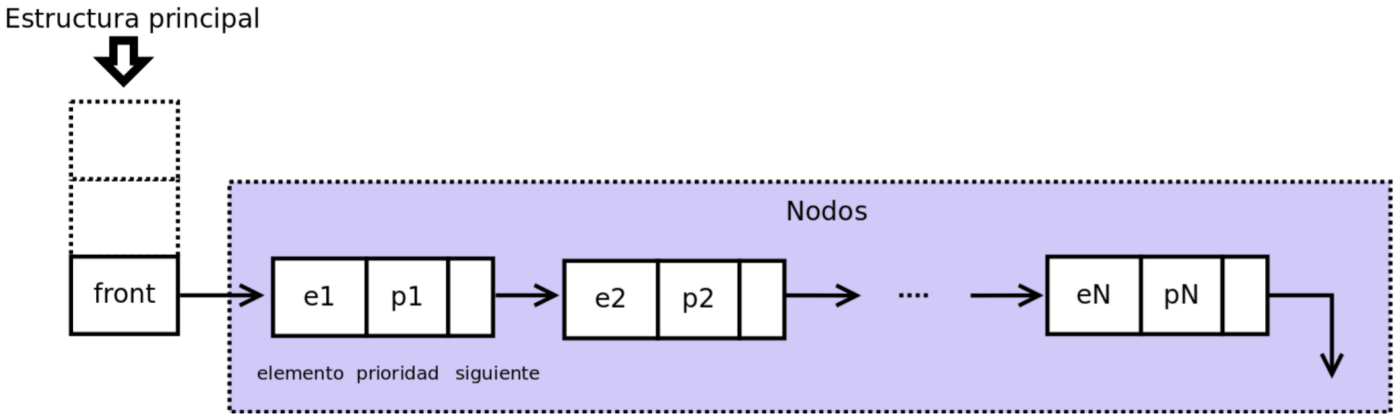
Segundo Parcial de Laboratorio

Algoritmos y Estructura de Datos II

TEMA B

Ejercicio

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



En la representación, **e1** es el primer elemento de la pila (el tope) 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 menor prioridad, y números más grandes establecen prioridades más altas). Además, si siempre se usa la misma prioridad, el TAD ***pstack*** debe comportarse exactamente igual a una pila común (LIFO: last input, first output). Algunos ejemplos:

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

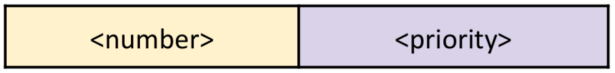
En este ejemplo el próximo elemento a desencolar es 15 ya que aunque tiene la misma prioridad que el elemento 11, fue agregado primero el 15 a la cola. **Lo importante es mantener la estructura de nodos bien ordenada para desapilar el elemento correcto**.

El TAD ***pstack*** tiene la siguiente interfaz

Función	Descripción
`pstack pstack_empty(void)`	Crea una pila de prioridades vacía
`pstack pstack_push(pstack s, pstack_elem e, priority_t priority);`	Inserta un elemento a la pila con su correspondiente prioridad.
`bool pstack_is_empty(pstack s);`	Indica si la pila de prioridades está vacía
`unsigned int pqueue_size(pstack s)`	Obtiene el tamaño de la pila de prioridades
`pstack_elem pstack_top(pstack s)`	Obtiene el elemento con mayor prioridad
`priority_t pstack_top_priority(pqueue q)`	Obtiene el valor de la prioridad del elemento con mayor prioridad.
`pstack pstack_pop(pstack s)`	Quita un elemento con mayor prioridad más recientemente apilado.
`pstack pstack_destroy(pstack s)`	Destruye una instancia del TAD ***pstack***

En `pstack.c` se da una implementación incompleta del TAD ***pstack*** que deben completar **siguiendo la representación explicada anteriormente.** Además deben asegurar que la función `pstack_size()` sea de orden constante (*O(1)*). Por las dudas se aclara que no es necesario que la función `pstack_push()` 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:



El archivo de entrada es una lista de elementos con prioridades de atención. Entonces el programa lee el archivo, carga los datos en el TAD ***pstack*** y finalmente muestra por pantalla la pila 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:

```
$ ./pstack_test inputs/example_a.in
```

Obteniendo como resultado:

```
length: 4
[ (686, best priority), (345, high priority), (456, normal priority), (454, low priority)]
```

Otro ejemplo de ejecución:

```
$ ./pstack_test inputs/example_b.in
length: 7
[ (700, normal priority), (600, normal priority), (500, normal priority),
  (400, normal priority), (300, normal priority), (200, normal priority),
  (100, normal priority)]
```

Otro ejemplo:

```
$ ./pstack_test inputs/example_c.in
length: 7
[ (234, best priority), (345, high priority), (454, low priority),
  (686, low priority), (789, low priority), (456, low priority),
  (234, low priority)]
```

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 **`pstack_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 pila de prioridades.