Córdoba 1 de Junio del 2023

Segundo Parcial de Laboratorio

Algoritmos y Estructura de Datos II

TEMA B

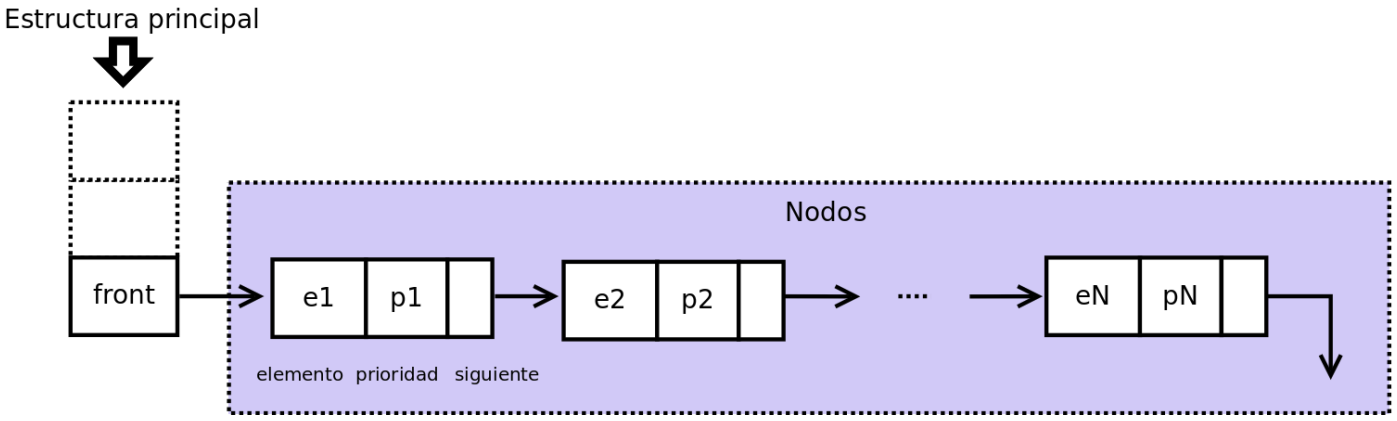
Ejercicio

La FaMAF está pensando en implementar un incentivo a alumnos que tengan buen rendimiento académico y para ellos cuenta con un fondo a invertir en alumnos según una **prioridad** (`priority`), calculada en base a su promedio de notas actual (`average_grade`) y la cantidad de materias aprobadas (`approved_courses`) utilizando la fórmula:

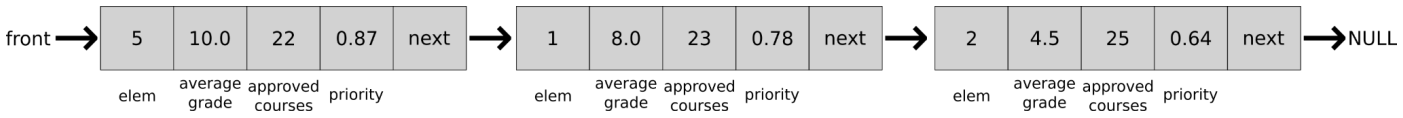
```
priority = 0.3 * (average_grade/MAX_GRADE) + 0.7 * (approved_courses/TOTAL_COURSES)
```

Donde `priority` es un valor entre `[0, 1]`, `MAX_GRADE` es el máximo valor de un promedio de notas `(10 en argentina)` `TOTAL_COURSES` es la cantidad total de materias de la carrera a la que pertenecen los estudiantes.

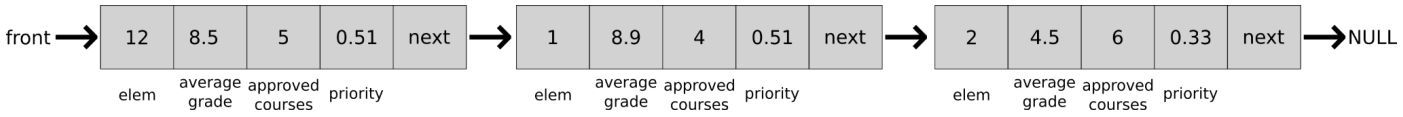
Con esta información se pide Implementar el TAD ***pstack*** que representa una pila de prioridades. Una pila de prioridades (**pstack**) es un tipo especial de pila en la que cada elemento está asociado con una prioridad asignada. En una pila de prioridades un elemento con mayor prioridad será desapilado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



En la representación, **e1** es el primer elemento de la pila y **p1** tiene la mayor prioridad. En realidad la propiedad fundamental es que **p1** ≥ **p2** ≥ … ≥ **pN** (notar que aquí la prioridad representada con 1 es la mayor prioridad, y números más bajos establecen prioridades más bajas). 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 desencolar es el **5** ya que tiene prioridad **.87** (la más prioritaria).



En este ejemplo el próximo elemento a desapilar es 12 ya que aunque tiene la misma prioridad que el elemento 1, fue agregado último en la pila. **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, float average_grade, unsigned int approved_courses);`	Inserta un elemento a la pila calculando su correspondiente prioridad.
`bool pstack_is_empty(pstack s);`	Indica si la pila de prioridades está vacía
`unsigned int pstack_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.
`float pstack_top_average_grade(pqueue q)`	Obtiene el valor del promedio de notas del elemento con mayor prioridad.
`unsigned int pstack_top_approved_courses(pqueue q)`	Obtiene la cantidad de cursos aprobados 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:

<student_id>	<average_grade>	<approved_courses>

El archivo de entrada representa una lista de estudiantes con sus notas-promedio y cantidad de cursos aprobados. Entonces el programa lee el archivo, carga los datos en el TAD ***pstack*** y finalmente muestra por pantalla la pila de prioridades de los estudiantes agregando además el valor de la prioridad calculada.

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

Una vez compilado el programa puede probarse ejecutando:

```
$ ./dispatch_students input/different.in
```

Obteniendo como resultado:

```
length: 7
5: 10.000000, 22 -- 0.685000
1: 8.000000, 23 -- 0.642500
2: 4.500000, 25 -- 0.572500
3: 4.000000, 22 -- 0.505000
4: 8.000000, 11 -- 0.432500
6: 10.000000, 6 -- 0.405000
7: 1.000000, 10 -- 0.205000
```

Otro ejemplo de ejecución:

```
$ ./dispatch_students ./input/same_grade_and_courses.in
3: 0.500000, 10 -- 0.190000
4: 0.500000, 10 -- 0.190000
1: 0.500000, 10 -- 0.190000
2: 0.500000, 10 -- 0.190000
```

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.
- **No modificar los .h**