Córdoba, 8 de Julio del 2021

Examen Final

Algoritmos y Estructuras de Datos II - Taller

El ejercicio consiste en implementar el TAD *Fence* que representa un cerco o corral. Un cerco puede estar formado por cuatro tipo de elementos:

- Poste (P)
- Alambre (A)
- Hueco (H)
- Tranquera (T))

Vamos a implementar al cerco utilizando una **lista enlazada de elementos** donde los nodos alojarán valores del tipo enumerado `type_t`:

```
typedef enum {P, A, H, T} type_t;
```

Las funciones a implementar en **`fence.c`** son las siguientes:

Función	Descripción
`fence_t fence_P();`	Cerco formado por un poste (únicamente)
`fence_t fence_add_P(fence_t fence);`	Agregar un poste al cerco
`fence_t fence_add_A(fence_t fence);`	Agregar alambre al cerco
`fence_t fence_add_H(fence_t fence);`	Agregar un hueco al cerco
`fence_t fence_add_T(fence_t fence);`	Agregar una tranquera al cerco
`bool fence_is_correct(fence_t fence);`	¿Es correcto el cerco?
`unsigned int fence_perimeter(fence_t fence);`	Perímetro del cerco
`unsigned int fence_size(fence_t fence);`	Cantidad de elementos del cerco
`type_t *fence_to_array(fence_t fence);`	Arreglo dinámico con los elementos del cerco
`fence_t fence_destroy(fence_t fence);`	Liberar la memoria

GURI LaBisagra
CEIMAF CENTRO DE ESTUDIANTES FAMAF

Perímetro

La función `fence_perimeter()` da valor 3 a todos los elementos excepto al poste que tiene valor 0. El resultado entonces debe ser la suma de los valores de cada elemento.

Cercos bien construidos

En `fence_is_correct()` se debe chequear que exista al menos una tranquera y que además todos los elementos tengan un poste de cada lado (excepto los postes mismos). Se dan algunos ejemplos en la siguiente tabla:

Cerco	*Retorno*	*Razón*
[P A P]	false	es un cerco **incorrecto** porque no tiene tranquera
[P T P]	true	**correcto**, tiene tranquera y está entre dos postes
[P T P P P A P]	true	**correcto**, tiene tranquera y **T** y **A** están entre postes
[T P P A P]	false	**incorrecto**, la primera tranquera no tiene poste izquierdo.
[P T P A P T T P]	false	**incorrecto**, la última tranquera no tiene poste izquierdo
[P T P A P H P P]	true	**correcto**

Arreglos

La función `fence_to_array()` debe devolver un arreglo dinámico con los elementos del cerco (del tipo `type_t`) en el orden apropiado (ver los ejemplos!). La cantidad de elementos contenidos en el arreglo se debe corresponder con el valor devuleto por `fence_size()`.

Ejemplos

```
f = fence_add_P(fence_add_A(fence_add_H(fence_P()))):
fence_to_array(f) == [P A H P]
fence_size(f) == 4
fence_perimeter(f) == 6
fence_is_correct(f) == false
```

```
f = fence_add_H(fence_add_P(fence_add_A(fence_add_H(fence_P())))):
fence_to_array(f) == [H P A H P]
fence_size(f) == 5
fence_perimeter(f) == 9
fence_is_correct(f) == false
```

```
f = fence_add_P(fence_add_T(fence_add_P(fence_add_A(fence_P())))):
fence_to_array(F) == [P T P A P]
fence_size(F) == 5
fence_perimeter(F) == 6
fence_is_correct(F) == true
```

GURI LaBisagra
CEIMAF CENTRO DE ESTUDIANTES FAMAF

Tests

Se provee un **`Makefile`** para compilar y correr una serie de *tests*, para ello deben ejecutar:

```
$ make test
```

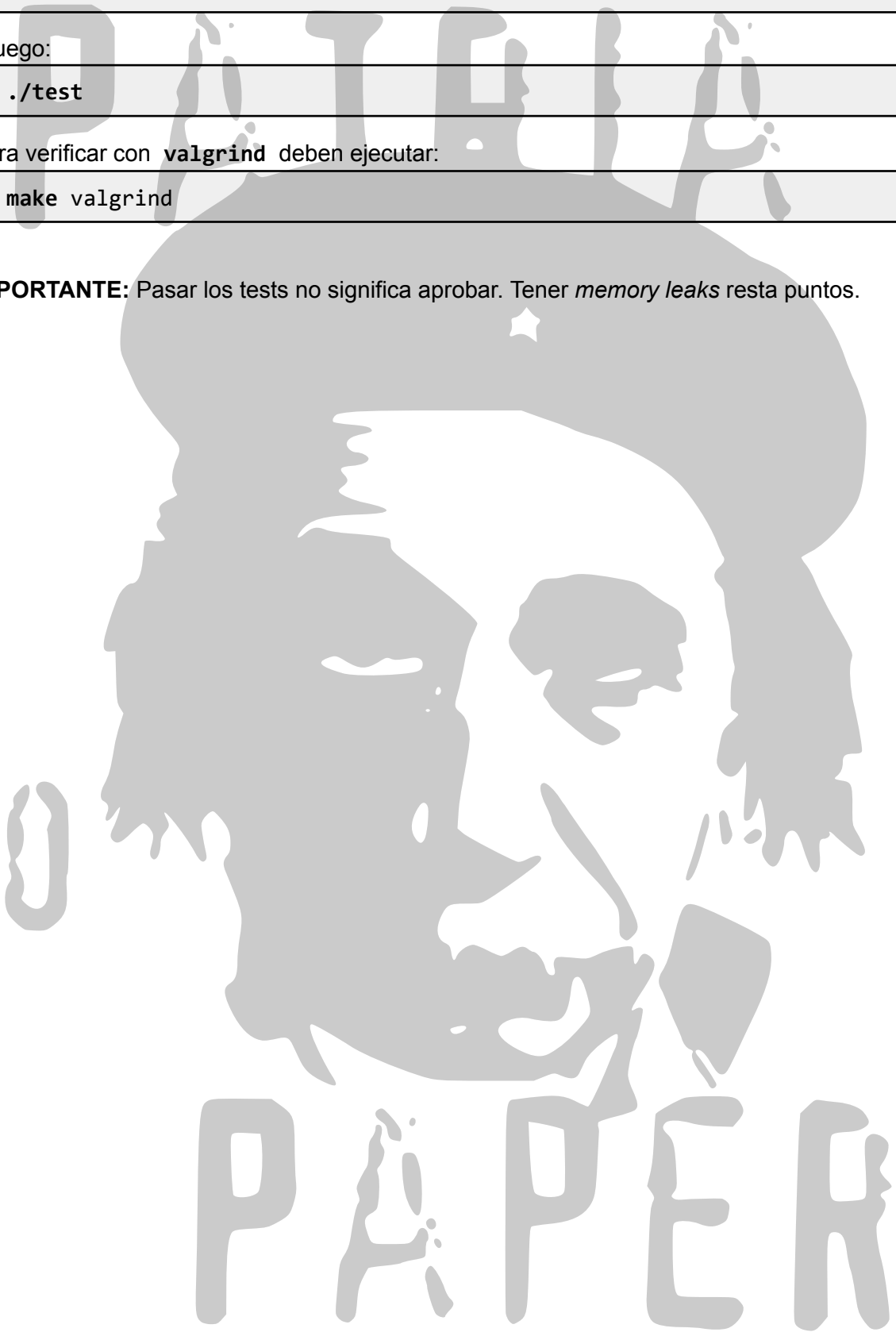
y luego:

```
$ ./test
```

Para verificar con **`valgrind`** deben ejecutar:

```
$ make valgrind
```

IMPORTANTE: Pasar los tests no significa aprobar. Tener *memory leaks* resta puntos.



GURI LaBisagra
CEIMAF CENTRO DE ESTUDIANTES FAMAF