

Algoritmos y Estructuras de Datos II

Laboratorio - 29/05/2025

Laboratorio 5 - Parte 2: TADs: Tipos Abstractos de Datos

- Revisión 2025: Franco Luque

Código

<lab05-2-kickstart.tar.gz>

Recursos

Recursos generales:

- [Videos del Laboratorio en el aula virtual](#)
- [Documentación en el aula virtual](#)
- Estilo de codificación:
 - [Guía de estilo para la programación en C](#)
 - [Consejos de Estilo de Programación en C](#)

Recursos específicos:

- Teórico/práctico:
 - [TADs](#)
 - [Práctico 2.2](#)
 - [Pilas y colas](#)
 - [Práctico 2.3](#)
- Para el ejercicio 6:
 - Arreglos circulares
 - Aritmética modular

Ejercicio 4: Tests para el TAD lista

Dentro de la carpeta `ej4` se encuentran los siguientes archivos:

Archivo	Descripción
<code>list.h</code>	Especificación del TAD lista
<code>tests.c</code>	Tests para algunas funciones de consulta: <code>head</code> , <code>index</code> y <code>length</code>

tests2.c	Tests para más funciones: tail, addr y take
-----------------	---

a) Leer y entender todo el código de testing.

b) Leer la especificación `list.h` dada. Adaptar la **implementación con listas enlazadas** realizada en el ejercicio 3 para que funcione con esta nueva especificación.

Funciones a implementar:

- empty
- add1
- is_empty
- head
- tail
- addr
- length
- concat
- index
- take
- drop
- copy_list
- destroy_list

Compilar y testear con:

```
$ gcc -Wall -Wextra -pedantic -std=c99 list.c tests.c -o tests
$ ./tests
$
$ ./tests2
```

Asegurarse de que todos los tests pasan exitosamente.

Aclaración: Al compilar Ignorar el siguiente warning:

```
aviso: ISO C forbids empty initializer braces before C23 [-Wpedantic]
```

c) Agregar tests nuevos para todas las funciones testeadas (head, index, length, tail, addr y take). Por lo menos dos tests por función.

d) En `tests2.c` agregar nuevos tests para las funciones drop y concat. Como mínimo cuatro tests por función.

Ejercicio 5: Listas con arreglos versión 1

Hacer una nueva implementación del TAD lista usando un arreglo y un número que indica el tamaño de la lista (como en los ejercicios [práctico 2.2 ejercicio 2](#) y [práctico 2.3 ejercicio 3a](#)).

No modificar `list.h` de ninguna manera. Compilar y testear usando todos los tests realizados en el ejercicio anterior.

Ayuda: El tipo se puede definir de la siguiente manera:

```
#define MAX_LENGTH 100

struct _list {
    elem elems[MAX_LENGTH];
    int size;
};
```

Ejercicio 6: Listas con arreglos versión 2

Hacer una nueva implementación del TAD lista usando la idea de **arreglos circulares** (como en el ejercicio [práctico 2.3 ejercicio 3b](#)). Las funciones `addr` y `add1` deben tener ambas **complejidad constante**.

En un **arreglo circular** se usa un índice `start` que indica la posición del arreglo donde comienzan los elementos de la lista y un número `size` que indica la cantidad de elementos. La lista puede dar “la vuelta” del arreglo, empezando al final del arreglo y terminando al principio.

Por ejemplo la lista: [8, 1, -2, 34, 4] se puede representar en un arreglo de 10 elementos de la siguiente manera:

índice:	0	1	2	3	4	5	6	7	8	9
valor:	34	4						8	1	-2

En este ejemplo, `start = 7` y `size = 5`.

Ayuda: Para las operaciones será necesario usar **aritmética modular**. Por ejemplo para la función `index`, la posición del arreglo a devolver es $(start + n) \% MAX_ELEMS$.