



Práctica 1 - Arreglos

1. La mediana de un arreglo de números es el elemento m para el cual la mitad del resto de los elementos del arreglo es mayor o igual que m y la otra mitad es menor o igual que m , si el número de elementos es impar. Si es par, la mediana se define como el promedio del par de elementos m_1 y m_2 para el cual una mitad de los elementos restantes del arreglo es mayor o igual que m_1 y m_2 y la otra mitad es menor o igual que m_1 y m_2 . Escriba una función que reciba un arreglo de números y calcule su mediana:

```
float mediana(float *arreglo, size_t longitud);
```

Puede utilizar la siguiente rutina para ordenar un arreglo de menor a mayor

```
void bubble_sort(float arreglo[], size_t longitud) {  
    for (int iter = 0 ; iter < longitud - 1 ; iter++) {  
        for (int i = 0 ; i < longitud - iter - 1; i++) {  
            if (arreglo[i] > arreglo[i + 1]) {  
                float aux = arreglo[i];  
                arreglo[i] = arreglo[i + 1];  
                arreglo[i + 1] = aux;  
            }  
        }  
    }  
}
```

Nota: La función `mediana` no debe modificar el arreglo original.

2. Trabajando con cadenas de caracteres:

a) Implemente la siguientes funciones:

- `int string_len(char* str)`, que retorne la longitud de la cadena.
- `void string_reverse(char* str)`, que invierta la cadena dada.
- `size_t string_concat(char* str1, char* str2, size_t max)`, que copie la cadena `str2` al final de la cadena `str1`, hasta un máximo de `max` caracteres en `str1`. Retorna el número de caracteres copiados.
- `int string_compare(char* str1, char* str2)`, que compare en orden lexicográfico las dos cadenas dadas y retorne -1, si `str1` es menor a `str2`, 0 si son iguales, y 1 si `str1` es mayor a `str2`.
- `int string_subcadena(char* str1, char* str2)`, que retorne el índice de la primera ocurrencia de la cadena `str2` en la cadena `str1`. En caso de no ocurrir nunca, retorna -1.
- `void string_unir(char* arregloStrings[], size_t capacidad, char* sep, char* res)`, que dada un arreglo de cadenas, las concatene (guardando el resultado en `res`), separándolas con la cadena `sep`.

b) Con el objetivo de medir el rendimiento de las funciones recién definidas, agregue un *contador de operaciones*. Es decir, una variable que se incremente cada vez que realizamos una operación elemental. En cada caso, ¿depende este número de la longitud del arreglo (o de los arreglos) que pasamos como argumento?

3. Considere arreglos definidos a través de una estructura que lleve registro de la capacidad:

```
typedef struct {
    int*   direccion;
    size_t capacidad;
} ArregloEnteros;
```

a) Implemente las operaciones básicas:

- `ArregloEnteros* arreglo_enteros_crear(size_t capacidad);`
- `void arreglo_enteros_destruir(ArregloEnteros* arreglo);`
- `int arreglo_enteros_leer(ArregloEnteros* arreglo, size_t pos);`
- `void arreglo_enteros_escribir(ArregloEnteros* arreglo, size_t pos, int dato);`
- `size_t arreglo_enteros_capacidad(ArregloEnteros* arreglo);`
- `void arreglo_enteros_imprimir_en_pantalla(ArregloEnteros arreglo);`

b) Si midiéramos el tiempo contando la cantidad de operaciones, como hicimos en el ejercicio 1.b), ¿cuál de ellas *tardaría* más? ¿Para cuál de ellas su tiempo depende de la longitud del arreglo?

4. Implemente las funciones:

- `void arreglo_enteros_ajustar(ArregloEnteros* arreglo, size_t capacidad)`, que ajuste el tamaño del arreglo. Si la nueva capacidad es menor, el contenido debe ser truncado.
- `void arreglo_enteros_insertar(ArregloEnteros* arreglo, size_t pos, int dato)`, que inserte el dato en la posición dada, moviendo todos los elementos desde esa posición un lugar a la derecha (tendrá que incrementar el tamaño del arreglo).
- `void arreglo_enteros_eliminar(ArregloEnteros* arreglo, size_t pos)`, que elimine el dato en la posición dada, reduciendo el tamaño del arreglo.

5. De manera similar al ejercicio anterior, implemente **Matriz**, un arreglo de números flotantes en dos dimensiones, junto a las operaciones básicas asociadas (crear, destruir, escribir una posición, leer de una posición).

a) Impleméntelo de dos maneras distintas:

- Utilizando un único arreglo unidimensional y trabajando con los índices.
- Utilizando un arreglo de punteros a arreglos que representen cada fila.

A pesar de tener dos implementaciones distintas, las operaciones básicas sobre una matriz son las mismas. Por dicha razón queremos tener un único archivo de cabecera `matriz.h`, y un archivo `.c` por cada implementación: `matriz.uni.c` y `matriz.bi.c`.

De esta forma, si tuviéramos un programa `main.c` que utiliza matrices (incluye `matriz.h`), podremos optar por la implementación preferida en el momento de compilarlo.

```
$ gcc main.c matriz.uni.c
```

b) ¿Cuáles son las ventajas y desventajas de cada implementación?

c) Defina una función `matriz_intercambiar_filas`, que intercambie dos filas dadas. ¿En cuál de las dos implementaciones anteriores resulta más simple de definir? ¿Y para la operación `matriz_insertar_fila`, que agrega una nueva fila en una posición dada?

- d) Utilizando las operaciones básicas, implementar `matriz_sumar` y `matriz_multiplicar`.
- e) Una Matriz Triangular inferior A es una matriz de tamaño n por n en la cual $A[i][j] == 0$, si $i < j$. ¿Cuál es el numero máximo de elementos distintos de cero en la misma? ¿Cómo pueden almacenarse en forma secuencial estos elementos en la memoria?
Implemente `MatrizTriangularInf` y sus operaciones básicas.