

# Algoritmos y Estructuras de Datos II

TALLER - 21 de marzo 2022

## Laboratorio 1: Arreglos, Archivos, Módulos

- Revisión 2019: Gonzalo Peralta
- Revisión 2021: Marco Rocchietti
- Revisión 2022: Marco Rocchietti

### Objetivos

1. Familiarizarse con vocabulario informático
2. Saber cómo compilar programas con `gcc`
3. Tener manejo de las instrucciones básicas del lenguaje de programación C
4. Comenzar a manejar archivos como fuente de datos
5. Tener manejo de **standard input** y **standard output**.
6. Tener nociones del manejo de parámetros a través de la función principal `main()`
7. Trabajar con módulos en C

### Ejercicio 1 - Lectura de archivos

En la carpeta `ej1` se encuentra el archivo principal `main.c` y un directorio `input` que contiene varios archivos con *extensión* `.in`. Cada archivo de la carpeta `input` contiene los datos de un arreglo que ha sido guardado dentro de él. El arreglo (o *array*) se representa con su tamaño (*size*) y luego se enumera cada uno de los elementos, separándolos con espacios. Por ejemplo, un *array* cuyos elementos son `[1,2,3,4,5]` es representado en el archivo como:

```
5
1 2 3 4 5
```

El archivo principal es `main.c`, donde se va a programar el ejercicio. Para compilarlo:

```
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -o reader main.c
```

Notar que el símbolo `$` no es parte del comando para compilar, sino que es el símbolo de *prompt* que indica que el comando `gcc` se debe ejecutar desde la consola. Se pide que el programa principal sea capaz de leer *arrays* de cualquiera de los archivos dentro del directorio `input` y que luego imprima por pantalla su contenido. Entonces por ejemplo al ejecutar el programa:

```
$ ./reader input/example-easy.in
```

se debe obtener la siguiente salida por pantalla:

```
[ 1, 2, 3, 4, 5]
```

Para ello se deben completar las definiciones de las funciones `array_from_file()` y la función `array_dump()`.

Se sugiere no esperar hasta el final para compilar, se puede ir compilando a medida que se completan las funcionalidades del programa.



*Pueden ser de utilidad las funciones `fopen()`, `fscanf()`, `fclose()` ... se pueden consultar las páginas del manual de referencia de linux, por ejemplo:*  
`$ man fopen`

## Ejercicio 2 - Entrada Estándar

Modificar `main.c` (no borrar el original!) para que el programa en lugar de leer un archivo de la carpeta `input`, lea el tamaño y cada uno de los miembros del `array` por teclado y luego los muestre por la pantalla. Se puede (y se sugiere fuertemente) reutilizar la función `array_from_file()`.



*Investigar sobre standard input: `$ man stdin`*

## Ejercicio 3 - Módulos

En este ejercicio se va a modularizar el programa del ejercicio 1. Primero se debe copiar el archivo `main.c` del ejercicio 1 al directorio `ej3` y luego completar los archivos:

- **`array_helpers.h`:** Se deben escribir aquí los *prototipos* de las funciones
  - `array_from_file()`
  - `array_dump()`
- **`array_helpers.c`:** Se deben colocar las definiciones de las funciones declaradas en `array_helpers.h` e incluir los prototipos al comienzo del archivo usando la directiva del preprocesador: `#include "array_helpers.h"`

En `main.c` también se debe incluir la librería `array_helpers.h` y borrar las funciones definidas en `array_helpers.c`. El programa se compila realizando los siguientes pasos:

```
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -c array_helpers.c  
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -c main.c
```

y finalmente:

```
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 array_helpers.o main.o -o reader
```

## Ejercicio 4 - Orden de elementos

Agregar la función `array_is_sorted()` a la librería `array_helpers`. La función tiene prototipo

```
bool array_is_sorted(int a[], unsigned int length);
```

y dado un *array* `a[]` y su tamaño `length` debe devolver `true` si y sólo si los elementos del arreglo `a[]` están ordenados de manera ascendente, es decir si:

$$a[0] \leq a[1] \leq \dots \leq a[\text{length} - 1]$$

Como se utiliza el tipo `bool`, el cual no es nativo del lenguaje C, **no olvidar incluir** en `array_helpers.h` y `array_helpers.c` la librería `stdbool.h`. Modificar `main.c` para que además de mostrar el contenido del *array* del archivo especificado, también informe si está ordenado o no. Un ejemplo sería:

```
./reader input/example-easy.in  
[1, 2, 3, 4, 5]  
El arreglo está ordenado
```

Otro ejemplo:

```
./reader input/example-unsorted.in  
[2, -1, 3, 8, 0]  
El arreglo no está ordenado
```

## Ejercicio 5 - Problemática de librerías: mybool

Aquí se utilizará el mismo programa construido en el ejercicio anterior, pero en vez de usar el tipo `bool` de `stdbool.h` se va a usar una definición casera de los *booleanos*. Como se vio en Algoritmos I, en C los enteros y los *booleanos* son muy parecidos. Se puede definir entonces en el archivo `mybool.h`:

```
typedef int mybool;
```

Recordar que `typedef` define sinónimos de tipos (como `type` en *Haskell*), por lo cual estamos diciendo que `mybool` es un sinónimo de `int` (son el mismo tipo). Además se definen las constantes `true` y `false`:

```
#define true 1  
#define false 0  
  
typedef int mybool;
```

En el archivo `test_mybool.c` se muestran ejemplos del uso de este tipo, donde se puede apreciar que se trabaja prácticamente igual que con el tipo `bool`. Para comparar además se puede ver el archivo `test_bool.c` que utiliza `stdbool.h`.

El objetivo entonces es reemplazar el uso de la librería `stdbool` por `mybool`, para ello se debe modificar `main.c`, `array_helpers.h` y `array_helpers.c` reemplazando al tipo `bool` por el tipo `mybool` y además reemplazando `#include <stdbool.h>` por `#include "mybool.h"`.

Una vez realizados los reemplazos, compilar el programa como indica el ejercicio 3 y responder:

### **-¿Por qué falla la compilación?**

Tener en cuenta que cuando se compila `test_mybool.c` todo funciona correctamente.

### **-¿Cómo se resuelve el problema?**

Para solucionar el problema sólo se debe modificar el archivo `mybool.h`.



*Investigar la directiva del preprocesador `#ifndef`*

Una vez resuelto el problema, hacer algo similar con `array_helpers.h` para evitar que pudiera generar el mismo problema.

## **Ejercicio 6\* - Bonus Track**

Agregar a la librería `array_helpers` la función `array_swap()` con prototipo

```
void array_swap(int a[], unsigned int i, unsigned int j);
```

que dado un *array* `a[]` y dos índices `i, j` debe intercambiar los valores de dichas posiciones.

Modificar `main.c` e invertir el *array* antes de mostrarlo por pantalla. Para ello pensar un algoritmo que utilice sucesivas llamadas a `array_swap()` para ir intercambiando los elementos del *array* hasta lograr invertirlo. El programa resultante debería comportarse de la siguiente manera:

```
./reader input/example-easy.in
[5, 4, 3, 2, 1]
El arreglo no está ordenado
```

Otro ejemplo:

```
./reader input/example-unsorted.in
[0, 8, 3, -1, 2]
El arreglo no está ordenado
```