Guía de Ejercicios 9 - Memoria Dinámica

```
Advertencia

La resolución conjunta o grupal de los ejercicios aquí presentes no está permitida, excepto en
la medida en que puedas pedir ayuda a tus compañeros de clase y a otras personas, y siempre que
esa ayuda no se reduzca a que otro haga el trabajo por vos.

El código fuente entregado por un estudiante debe ser escrito en su totalidad por dicha persona.
```

Condiciones de entrega:

¿Qué se entrega?	¿Qué no se entrega?
Archivos fuente/source (.c)	Archivos objeto (.o)
Archivos encabezado/header (.h)	Archivos ejecutables (programa, app, a.out, etc.)
Bibliotecas específicas (.a)	

Se deben entregar los ejercicios en un archivo zip (usar template como ayuda para el formato).

Importante: Recordá validar **siempre** que no se reciben punteros `NULL`. En dicho caso, la función deberá retornar sin efectuar operación alguna y en caso de tener que retornar algún valor devolverá el valor `-1`.

Recordá también que se evaluarán las buenas prácticas de programación con respecto al pedido de memoria. Por lo tanto, ¡no te olvides de **liberar** la memoria pedida! (podés ayudarte del programa `valgrind` para verificarlo).

Tené en cuenta también que para **todos** los ejercicios, se pide también implementar (y entregar) una función main que demuestre el correcto funcionamiento del módulo.

Ejercicio 9.1

Utilizando las funciones `malloc`, `realloc` y `free`, desarrollá un programa en el que se ingrese cíclicamente por teclado la cantidad en bytes de memoria que se desea reservar o liberar. El programa deberá reservar o liberar de un solo bloque de memoria y preguntar por un nuevo valor. Si el valor es positivo, se reserva ese valor en bytes **más** de memoria para ese mismo bloque. Si es negativo, se **libera** ese valor en bytes de memoria. El programa debe finalizar (indicando lo sucedido) cuando:

- El valor ingresado es `cero`, indicando **cuánta** memoria había quedado reservada antes de finalizar el programa. No te olvides de liberar dicha memoria antes de finalizar.
- El resultado de reservar y liberar memoria equivale o es menor a **cero**.
- No hay **suficiente** espacio de memoria disponible para reservar. Al finalizar el programa, no te olvides de liberar cualquier espacio de memoria que haya quedado reservado.

Ejercicio 9.2

Desarrollá un programa donde el usuario ingrese letras mayúsculas y minúsculas de a una a la vez. El ingreso finaliza cuando el usuario ingresa `FFF` (tres `'F'` seguidas). Tené en cuenta que no se conoce la cantidad de caracteres a ingresar y se deben validar que los caracteres ingresados correspondan solamente a letras mayúsculas y minúsculas, caso contrario son descartados y no por lo tanto **no** se almacenan. Al finalizar, se deben imprimir por pantalla los caracteres ingresados.

Ejercicio 9.3

Desarrollá un programa que solicite el ingreso de las calificaciones (como número enteros) del primer parcial de los estudiantes de Informática 1. El ingreso de calificaciones finalizará cuando se ingrese un once (11). Tené en cuenta que las calificaciones deberán ser números comprendidos entre 0 y 10 (0 significa ausente), por lo que se debe validar la carga de datos. Una vez finalizada la carga de datos, se pide computar e imprimir en pantalla las calificaciones ingresadas, la cantidad total de calificaciones y la cantidad de aprobados, desaprobados y ausentes. Utilizá los siguientes prototipos:

```c
// Agrega una nueva calificacion en el arreglo dinámico.
// Retorna ERROR (-1) si ocurrió algún error, y EXITO (0) en caso contrario.
int agregar_calificacion(int** calificaciones, int* cantidad, int calificacion);

// Clasifica las calificaciones en aprobados, desaprobados y ausentes.
void clasificar_calificaciones(const int* calificaciones, int cantidad, int* aprobados, int*
desaprobados, int* ausentes);
```

Ejercicio 9.4

Implementá una función que reciba un string y retorne una copia del mismo (utilizando memoria dinámica) por valor. En caso de error, se deberá retornar `NULL`. Utilizá el siguiente prototipo:

```c
char* copiar_string(const char* string);
```

Ejercicio 9.5

Implementá una función que reciba un string y retorne una copia del mismo (utilizando memoria dinámica) por referencia. La función deberá retornar `ERROR` (-1) si ocurrió algún error, y `EXITO` (0) en caso contrario. Utilizá el siguiente prototipo:

```c
int copiar_string(const char* string_original, char** string_copia);
```

Ejercicio 9.6

Implementá una función que reciba un arreglo numérico y retorne una copia del mismo (utilizando memoria dinámica) por valor. Si por algún motivo no se puede, se deberá retornar `NULL`. Utilizá el siguiente prototipo:

```c
int* copiar_arreglo(const int* arreglo, int largo);
```

Ejercicio 9.7

Implementá una función que reciba un arreglo numérico y retorne una copia del mismo (utilizando memoria dinámica) por referencia. La función deberá retornar `ERROR` (-1) si ocurrió algún error, y `EXITO` (0) en caso contrario. Utilizá el siguiente prototipo:

```c
int copiar_arreglo(const int* arreglo_original, int largo, int** arreglo_copia);
```

Ejercicio 9.8

Implementá una función que reciba un string y un caracter. La función retorna una copia del string (utilizando memoria dinámica) por referencia, pero elimina todo lo que esté después del caracter recibido. La función deberá retornar `ERROR` (-1) si ocurrió algún error, y `EXITO` (0) en caso contrario. Utilizá el siguiente prototipo:

```c
int strip(const char* string_in, char caracter, char** string_out);
```

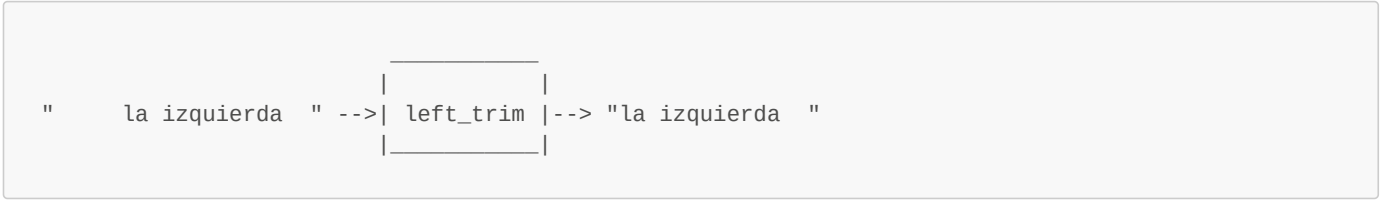
Ejemplo:

Ejercicio 9.9

Implementá una función que reciba un string y retorne una copia del string (utilizando memoria dinámica) por referencia, eliminando los espacios al comienzo (si es que hay alguno). La función deberá retornar `ERROR` (-1) si ocurrió algún error, y `EXITO` (0) en caso contrario. Utilizá el siguiente prototipo:

```
int left_trim(const char* string_in, char** string_out);
```

Ejemplo:

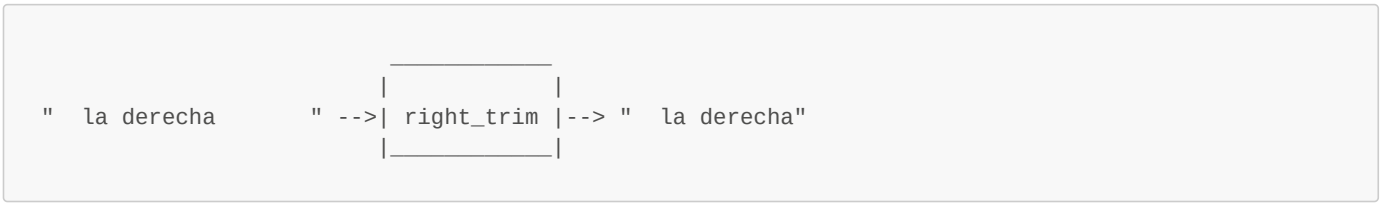


Ejercicio 9.10

Implementá una función que reciba un string y retorne una copia del string (utilizando memoria dinámica) por referencia, eliminando los espacios al final (si es que hay alguno). La función deberá retornar `ERROR` (-1) si ocurrió algún error, y `EXITO` (0) en caso contrario. Utilizá el siguiente prototipo:

```
int right_trim(const char* string_in, char** string_out);
```

Ejemplo:

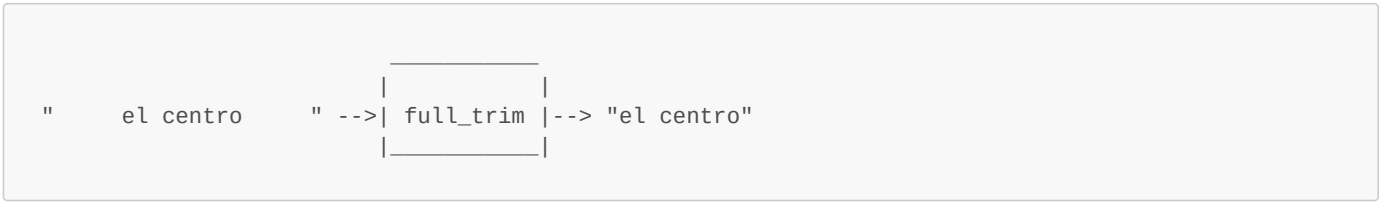


Ejercicio 9.11

Implementá una función que reciba un string y retorne una copia del string (utilizando memoria dinámica) por referencia, eliminando los espacios al comienzo y al final (si es que hay alguno). La función deberá retornar `ERROR` (-1) si ocurrió algún error, y `EXITO` (0) en caso contrario. Utilizá el siguiente prototipo:

```
int full_trim(const char* string_in, char** string_out);
```

Ejemplo:



Ejercicio 9.12

Implementá una función que reciba un string, un caracter y una longitud. La función retorna una copia del string original (utilizando memoria dinámica), centrado en la longitud especificada y rellenando los espacios restantes con el caracter pedido. La función deberá retornar `ERROR` (-1) si ocurrió algún error, y `EXITO` (0) en caso contrario. Utilizá el siguiente prototipo:

```
int centrar(const char* string_in, char caracter, int largo, char** string_out);
```

Ejemplo:

```
                          _________
                         |         |
(" cadena ", '*', 12) -->| centrar |--> "** cadena **"
                         |_________|
```

Ejercicio 9.13

Implementá una función que reciba un número entero y retorne un string que lo represente. La función deberá retornar `ERROR` (-1) si ocurrió algún error, y `EXITO` (0) en caso contrario. Utilizá el siguiente prototipo:

```
int itostr(int numero, char** string);
```

Ejemplo:

```
        ________
       |        |
9511 -->| itostr |--> "9511"
       |________|
```

Ejercicio 9.14

Implementá una función que reciba un string que contenga varios campos, en formato de campos de ancho fijo, y un arreglo con los anchos de cada campo. La función retorna **por valor** un arreglo de strings con cada campo extraído (copiado, utilizando memoria dinámica). La función deberá retornar `NULL` si ocurrió algún error. Utilizá el siguiente prototipo:

```
char** split(const char* string_in, const int* anchos, int cantidad_campos);
```

Ejemplo:

```
                          _______
                         |       |
("unodostres",{3,3,4}) -->| split |-->{"uno", "dos", "tres"}
                         |_______|
```

Ejercicio 9.15

Implementá una función que reciba un string que contenga varios campos, en formato de campos de ancho fijo, y un arreglo con los anchos de cada campo. La función retorna **por referencia** un arreglo de strings con cada campo extraído (copiado, utilizando memoria dinámica). La función deberá retornar `ERROR` (-1) si ocurrió algún error, y `EXITO` (0) en caso contrario. Utilizá el siguiente prototipo:

```
int split(const char* string_in, const int* anchos, int cantidad_campos, char*** campos);
```

Ejemplo:

```
                          _______
                         |       |
```

```
("unodostres",{3,3,4}) -->| split |-->{"uno", "dos", "tres"}
                          |_______|
```

Ejercicio 9.16

Implementá una función que reciba un string que contenga varios campos, en formato CSV (ver https://es.wikipedia.org/wiki/Valores_separados_por_comas). La función retorna **por valor** un arreglo de strings con cada campo extraído y **por referencia** la cantidad de campos. La función deberá retornar `NULL` si ocurrió algún error. Utilizá el siguiente prototipo:

```
char** split_csv(const char* string_in, int* cantidad_campos);
```

Ejemplo:

```
                    ___________
                   |           |
("uno,dos,tres") -->| split_csv |-->({"uno", "dos", "tres"}, 3)
                   |___________|
```

Ejercicio 9.17

Implementá una función que reciba un string que contenga varios campos, en formato CSV (ver https://es.wikipedia.org/wiki/Valores_separados_por_comas). La función retorna **por referencia** un arreglo de strings con cada campo extraído y la cantidad de campos. La función deberá retornar `ERROR` (-1) si ocurrió algún error, y `EXITO` (0) en caso contrario. Utilizá el siguiente prototipo:

```
int split_csv(const char* string_in, char*** campos, int* cantidad_campos);
```

Ejemplo:

```
                    ___________
                   |           |
("uno,dos,tres") -->| split_csv |-->({"uno", "dos", "tres"}, 3)
                   |___________|
```

Ejercicio 9.18

Implementá una función que reciba un arreglo de strings y su longitud y retorne **por valor** un string en formato CSV (ver https://es.wikipedia.org/wiki/Valores_separados_por_comas) uniendo todos los strings del vector, agregando el caracter delimitador entre ellos (`','`). La función deberá retornar `NULL` si ocurrió algún error. Utilizá el siguiente prototipo:

```
char* join_csv(const char** campos, int cantidad_campos);
```

Ejemplo:

```
                          _________
                         |         |
({"uno", "dos", "tres"})-->| join_csv |-->"uno,dos,tres"
                         |__________|
```

Ejercicio 9.19

Implementá una función que reciba un arreglo de strings y su longitud y retorne **por referencia** un string en formato CSV (ver https://es.wikipedia.org/wiki/Valores_separados_por_comas) uniendo todas los strings del vector, agregando el caracter delimitador entre ellos (`','`). La función deberá retornar `ERROR` (-1) si ocurrió algún error, y `EXITO` (0) en caso contrario. Utilizá el siguiente prototipo:

```
int join_csv(const char** campos, int cantidad_campos, char** string_out);
```

Ejemplo:

```
                          _________
                         |         |
({"uno", "dos", "tres"})-->| join_csv |-->"uno,dos,tres"
                         |_________|
```

Ejercicio 9.20

Implementá una función que lea, de stdin, un string de largo indefinido y lo retorne **por valor**. Sugerimos hacer uso de las funciones estándar `fgetc` o `getchar` para obtener los datos desde stdin. La función deberá retornar `NULL` si ocurrió algún error. Utilizá el siguiente prototipo:

```
char* obtener_string(void);
```

Ejercicio 9.21

Implementá una función que lea, de stdin, un string de largo indefinido y lo retorne **por referencia**. Sugerimos hacer uso de las funciones estándar `fgetc` o `getchar` para obtener los datos desde stdin. La función deberá retornar `ERROR` (-1) si ocurrió algún error, y `EXITO` (0) en caso contrario. Utilizá el siguiente prototipo:

```
int obtener_string(char** string);
```

Ejercicio 9.22

Implementá una función que reciba el inicio de un intervalo, el final y la cantidad de puntos y retorne **por valor** un arreglo de números linealmente espaciados entre el inicio y el final. La función deberá retornar `NULL` si ocurrió algún error. Utilizá el siguiente prototipo:

```
double* linspace(double inicio, double fin, int cantidad_puntos);
```

Ejercicio 9.23

Implementá una función que reciba el inicio de un intervalo, el final y la cantidad de puntos y retorne **por referencia** un arreglo de números linealmente espaciados entre el inicio y el final. La función deberá retornar `ERROR` (-1) si ocurrió algún error, y `EXITO` (0) en caso contrario. Utilizá el siguiente prototipo:

```
int linspace(double inicio, double fin, int cantidad_puntos, double** valores);
```

Ejercicio 9.24

Implementá una función que reciba dos arreglos de números enteros y retorne **por valor** un nuevo arreglo con los números de ambos arreglos concatenados. La función deberá retornar `NULL` si ocurrió algún error. Utilizá el siguiente prototipo:

```
int* arraycat(const int* arreglo_1, int largo_1, const int* arreglo_2, int largo_2);
```

Ejercicio 9.25

Implementá una función que reciba dos arreglos de números enteros y retorne **por referencia** un nuevo arreglo con los números de ambos arreglos concatenados. La función deberá retornar `ERROR` (-1) si ocurrió algún error, y `EXITO` (0) en caso contrario. Utilizá el siguiente prototipo:

```
int arraycat(const int* arreglo_1, int largo_1, const int* arreglo_2, int largo_2, int**
arreglo_out);
```

Ejercicio 9.26

Implementá una función que reciba la cantidad de filas y columnas de una matriz y un valor de inicialización. La función retorna **por valor** una matriz (utilizando memoria dinámica) del tamaño indicado e inicializada con el valor solicitado. La función deberá retornar `NULL` si ocurrió algún error. Utilizá el siguiente prototipo:

```
int* generar_matriz(int filas, int columnas, int valor);
```

Ejercicio 9.27

Implementá una función que reciba la cantidad de filas y columnas de una matriz y un valor de inicialización. La función retorna **por referencia** una matriz (utilizando memoria dinámica) del tamaño indicado e inicializada con el valor solicitado. La función deberá retornar `ERROR` (-1) si ocurrió algún error, y `EXITO` (0) en caso contrario. Utilizá el siguiente prototipo:

```
int generar_matriz(int filas, int columnas, int valor, int** matriz);
```

Referencias

Algunos ejercicios fueron obtenidos y adaptados de:

- Guía de Trabajos Prácticos 2011 - Informática I - Departamento de Electrónica - UTN FRBA
- Guía de Ejercicios - Algoritmos y Programación I - UBA FIUBA