Guía de Ejercicios 4 - Punteros

```
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 tres ejercicios en un zip (usar template como ayuda para el formato).

Ejercicio 4.1

Escribir una función que incremente en uno (1) el valor de una variable entera pasada por referencia. Utilizar el siguiente prototipo:

```c
void incrementar(int* numero);
```

Importante: Validar que no se reciben punteros `NULL`. En dicho caso, la función retornará sin efectuar operación alguna.

Ejercicio 4.2

Escribir una función llamada swap que reciba dos datos llamados `a` y `b` por referencia e intercambie su contenido, de forma tal que `b` pase a tener el contenido que originalmente tenía `a` y viceversa. Utilizar el siguiente prototipo:

```c
void swap(int* a, int* b);
```

Importante: Validar que no se reciben punteros `NULL`. En dicho caso, la función retornará sin efectuar operación alguna.

Ejercicio 4.3

Escribir una función que compute el promedio de dos números enteros y almacene el resultado de la operación en una variable pasada por referencia. Utilizar el siguiente prototipo:

```c
void promedio(int x, int y, float* resultado);
```

Importante: Validar que no se reciben punteros `NULL`. En dicho caso, la función retornará sin efectuar operación alguna.

Ejercicio 4.4

Escribir una función que reciba dos números enteros y una operación a realizar entre ellos. Las operaciones soportadas son:

- Suma (`'+'`)
- Resta (`'-'`)
- Multiplicación (`'*'`)
- División (`'/'`)
- Módulo (`'%'`)

Finalmente, deberá almacenar el resultado de la operación en una variable pasada por referencia. En caso de error por operación inválida, la función deberá retornar `ERROR_OPERACION_INVALIDA` (-2). En caso de error por división por cero, la función deberá retornar `ERROR_DIVISION_POR_CERO` (-1). En caso de éxito, la función deberá retornar `EXITO` (0). Utilizar el siguiente prototipo:

```
int calculadora(int x, int y, char operacion, int* resultado);
```

Importante: Validar que no se reciben punteros `NULL`. En dicho caso, la función retornará sin efectuar operación alguna.

Ejercicio 4.5

Escribir una función que convierta un número que representa una cantidad total de segundos, a su equivalente en horas, minutos y segundos, retornando dichos valores en variables pasadas por referencia. Utilizar el siguiente prototipo:

```
void a_sexagesimal(int total_segundos, int* horas, int* minutos, int* segundos);
```

Importante: Validar que no se reciben punteros `NULL`. En dicho caso, la función retornará sin efectuar operación alguna.

Ejercicio 4.6

Implementar la siguiente función:

```
int ordenar(int* min, int* max);
```

Dicha función recibirá dos números enteros por referencia (`min` y `max`). La función deberá verificar si cada uno está en la posición que corresponde y si así no fuera, ordenar su ubicación (si `min` no es menor a `max` se los debe intercambiar).

La función debe devolver `CORRECTO` (0) si los números estaban bien ubicados o `INCORRECTO` (-1) en caso contrario.

Importante: Validar que no se reciben punteros `NULL`. En dicho caso, la función retornará sin efectuar operación alguna.

Ejercicio 4.7

Escribir una función que reciba un número entero pasado por referencia y muestre cómo se almacena en memoria. Primero lo mostrará como un entero en formato decimal y en formato hexadecimal y luego mostrará las posiciones de memoria byte a byte y el valor que hay almacenado en formato hexadecimal. Utilizar el siguiente prototipo:

```
void mostrar_entero_en_memoria(int* numero);
```

Ejemplo de cómo se debería ver la salida:

```
Ingrese un número entero para visualizar cómo se almacena en memoria: 78456
Valor ingresado visualizado como entero:
Dirección de memoria      Valor decimal      Valor Hexadecimal
0x7ffe4dd88dc4:           78456              00013278
Valor ingresado visualizado byte a byte:
Dirección                 Valor Hexadecimal
0x7ffe4dd88dc4:           78
0x7ffe4dd88dc5:           32
0x7ffe4dd88dc6:           01
0x7ffe4dd88dc7:           00
```

Importante: Validar que no se reciben punteros `NULL`. En dicho caso, la función retornará sin efectuar operación alguna.