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

Importante: Recordar 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`.

Ejercicio 4.1

Implementar una función que incremente en `1` el valor de una variable entera pasada por referencia. Utilizar el siguiente prototipo:

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

Ejercicio 4.2

Implementar 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);
```

Ejercicio 4.3

Implementar 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);
```

Ejercicio 4.4

Implementar 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` (`1`). En caso de error por división por cero, la función deberá retornar `ERROR_DIVISION_POR_CERO` (`2`). En caso de éxito, la función deberá retornar `EXITO` (`0`). Utilizar el siguiente prototipo:

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

Ejercicio 4.5

Implementar una función que convierta una letra minúscula a su equivalente mayúscula. Utilizar el siguiente prototipo:

```
int a_mayuscula(char* letra);
```

La función debe devolver `EXITO` (`0`) si se pudo realizar la conversión o `ERROR` (`1`) en caso contrario.

Ejercicio 4.6

Implementar una función que eleve un determinado número al cuadrado. Utilizar el siguiente prototipo:

```
void elevar_al_cuadrado(float* numero);
```

Ejercicio 4.7

Implementar 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);
```

Ejercicio 4.8

Implementar una función que convierta un número en grados decimales, a su equivalente en grados, minutos y segundos, retornando dichos valores en variables pasadas por referencia. Utilizar el siguiente prototipo:

```
void a_grados_sexagesimales(float grados_decimales, int* grados, int* minutos, int* segundos);
```

Ejercicio 4.9

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 `ORDENADOS` (`0`) si los números estaban bien ubicados o `DESORDENADOS` (`1`) en caso contrario.

Ejercicio 4.10

Implementar una función que dadas dos rectas definidas por su pendiente y su ordenada al origen devuelva el punto de intersección. Validar lo que considere necesario. Utilizar el siguiente prototipo:

```
int interseccion_rectas(float m1, float b1, float m2, float b2, float* x, float* y);
```

La función debe devolver `EXITO` (`0`) si existe punto de intersección o `ERROR` (`1`) en caso contrario.

Ejercicio 4.11

Implementar una función tal que ingresando los coeficientes `a`, `b` y `c` de una ecuación cuadrática $a^2x + bx + c = 0$, compute sus raíces, a partir de la ecuación resolvente:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Utilizar el siguiente prototipo:

```
int raices_cuadratica(float a, float b, float c, float* x1, float* x2);
```

La función deberá devolver `RAICES_REALES_Y_DISTINTAS` (`0`), `RAICES_REALES_E_IGUALES` (`1`), `RAICES_COMPLEJAS` (`2`) o `ERROR` (`3`), según corresponda.

Ejercicio 4.12

Implementar una función que calcule la mínima cantidad de billetes (moneda peso argentino) con la que se puede alcanzar un determinado monto. Por ejemplo, `2320 ARS = 2*1000 ARS + 3*100 ARS + 1*20 ARS`. Utilizar el siguiente prototipo:

```
int cantidad_billetes(int monto, int* billetes_1000, int* billetes_500, int* billetes_200, int*
billetes_100, int* billetes_50, int* billetes_20, int* billetes_10);
```

La función deberá devolver `MONTO_JUSTO` (`0`) o `MONTO_REDONDEADO` (`1`), según corresponda.

Ejercicio 4.13

Implementar una función que reciba tres números que corresponden a los lados de un triángulo y lo clasifique según sus lados y según sus ángulos. Utilizar el siguiente prototipo:

```
int clasificar_triangulo(float lado1, float lado2, float lado3, int* segun_lados, int*
segun_angulos);
```

La función deberá retornar por valor `ES_TRIANGULO` (`0`) o `NO_ES_TRIANGULO` (`1`), según corresponda.

En caso de conformar un triángulo, la función deberá retornar por referencia a través de la variable `segun_lados` el valor `EQUILATERO` (`0`), `ISOSCELES` (`1`) o `ESCALENO` (`2`), según corresponda.

Por otro lado, la función deberá retornar por referencia a través de la variable `segun_angulos` el valor `ACUTANGULO` (`0`), `RECTANGULO` (`1`) o `OBTUSANGULO` (`2`), según corresponda.

Ejercicio 4.14

Implementar 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
```