🎯 Ejercicios Progresivos de Recursión en C

🟢 NIVEL BÁSICO - Números

Ejercicio 1: Contar dígitos

Problema: Escribir una función recursiva que cuente la cantidad de dígitos de un número.

c

```c
int contar_digitos(int numero);
// Ejemplos: contar_digitos(1234) → 4, contar_digitos(0) → 1
```

Ejercicio 2: Suma de dígitos

Problema: Calcular la suma de todos los dígitos de un número.

c

```c
int suma_digitos(int numero);
// Ejemplos: suma_digitos(123) → 6, suma_digitos(456) → 15
```

Ejercicio 3: Número invertido

Problema: Invertir los dígitos de un número.

c

```c
int invertir_numero(int numero);
// Ejemplos: invertir_numero(123) → 321, invertir_numero(1000) → 1
```

Ejercicio 4: Verificar palíndromo numérico

Problema: Verificar si un número es palíndromo (se lee igual al revés).

c

```c
int es_palindromo(int numero);
// Ejemplos: es_palindromo(121) → 1, es_palindromo(123) → 0
```

🟡 NIVEL INTERMEDIO - Arrays Básicos

Ejercicio 5: Encontrar máximo

Problema: Encontrar el elemento máximo en un array.

c

```c
int encontrar_maximo(int arr[], int n);
// Ejemplo: [3,1,4,1,5] → 5
```

Ejercicio 6: Contar ocurrencias

Problema: Contar cuántas veces aparece un valor en un array.

c

```c
int contar_ocurrencias(int arr[], int n, int valor);
// Ejemplo: [1,2,1,3,1] valor=1 → 3
```

Ejercicio 7: Array ordenado

Problema: Verificar si un array está ordenado ascendentemente.

c

```c
int esta_ordenado(int arr[], int n);
// Ejemplo: [1,2,3,4] → 1, [1,3,2,4] → 0
```

Ejercicio 8: Suma alternada

Problema: Calcular suma alternada: +arr[0] -arr[1] +arr[2] -arr[3]...

c

```c
int suma_alternada(int arr[], int n);
// Ejemplo: [1,2,3,4] → 1-2+3-4 = -2
```

🟠 NIVEL INTERMEDIO-AVANZADO - Arrays Complejos

Ejercicio 9: Producto de pares

Problema: Calcular el producto de todos los números pares en un array.

c

```c
int producto_pares(int arr[], int n);
// Ejemplo: [1,2,3,4,5,6] → 2*4*6 = 48
```

Ejercicio 10: Segundo máximo

Problema: Encontrar el segundo elemento más grande.

c

```c
int segundo_maximo(int arr[], int n);
// Ejemplo: [3,1,4,1,5,2] → 4
```

Ejercicio 11: Intercalar arrays

Problema: Verificar si un array puede formarse intercalando dos arrays ordenados.

c

```c
int puede_intercalar(int arr1[], int n1, int arr2[], int n2, int resultado[], int nr);
// Verificar si resultado[] es intercalación válida de arr1[] y arr2[]
```

🔴 NIVEL AVANZADO - Algoritmos Complejos

Ejercicio 12: Subsecuencia común más larga

Problema: Encontrar la longitud de la subsecuencia común más larga entre dos arrays.

c

```c
int lcs_longitud(int arr1[], int n1, int arr2[], int n2);
```

Ejercicio 13: Partición de suma igual

Problema: Verificar si un array puede dividirse en dos subconjuntos con suma igual.

c

```c
int puede_particionar(int arr[], int n);
// Ejemplo: [1,5,11,5] → 1 (se puede: {1,5,5} y {11})
```

Ejercicio 14: Camino con suma mínima (matriz)

Problema: Encontrar la suma mínima desde esquina superior izquierda a inferior derecha.

c

```c
int suma_minima_camino(int matriz[][MAX_COL], int fila, int col, int filas, int cols);
```

🎯 EJERCICIOS DE PRÁCTICA INMEDIATA

Ejercicio A: Múltiplos del siguiente

Completar el ejercicio de tu guía:

c

```c
int contar_multiplos_siguiente(int arr[], int n);
// [16,8,2,4,5,15] → 3 (16 múltiplo de 8, 8 de 2, 5 de 15)
```

Ejercicio B: Elementos únicos

Problema: Contar elementos que aparecen exactamente una vez.

c

```c
int elementos_unicos(int arr[], int n);
// [1,2,3,2,4] → 3 (elementos 1,3,4 aparecen una vez)
```

Ejercicio C: Suma de productos consecutivos

Problema: Calcular arr[0]*arr[1] + arr[1]*arr[2] + ... + arr[n-2]*arr[n-1]

c

```c
int suma_productos_consecutivos(int arr[], int n);
// [1,2,3,4] → 1*2 + 2*3 + 3*4 = 2+6+12 = 20
```

📚 PLANTILLAS PARA RESOLVER

Para números:

c

```c
int funcion_numero(int num) {
    if (num < 10) {
        // caso base: un dígito
        return ...;
    }
    // trabajar con ultimo dígito: num % 10
    // recurrir con resto: num / 10
    return ... + funcion_numero(num / 10);
}
```

Para arrays:

c

```c
int funcion_array(int arr[], int n) {
    if (n == 0) {
        // array vacío
        return ...;
    }
    if (n == 1) {
        // un elemento
        return ...;
    }
    // trabajar con arr[0]
    // recurrir con arr+1, n-1
    return ... + funcion_array(arr + 1, n - 1);
}
```

Para búsqueda/verificación:

c

```c
int buscar_verificar(int arr[], int n, int valor) {
    if (n == 0) {
        return 0; // no encontrado/no cumple
    }
    if (condicion_con_arr[0]) {
        return 1; // encontrado/cumple
    }
    return buscar_verificar(arr + 1, n - 1, valor);
}
```

🏆 Desafío Final

Implementar las 4 funciones de tu guía usando las plantillas y metodología explicada.

¿Con cuál te gustaría empezar? ¡Te ayudo paso a paso!