===============================================

✅ CHECKLIST DE C — PREPARACIÓN PARA EMBEDDED SYSTEMS

===============================================

📦 1. Fundamentos Básicos

--------------------------

[ ] Estructura de un programa en C

[ ] Declaración y uso de variables (int, char, float, etc.)

[ ] Operadores aritméticos (+, -, \*, /, %)

[ ] Operadores lógicos (&&, ||, !)

[ ] Condicionales: if, else, switch

[ ] Bucles: for, while, do-while

[ ] Funciones: declaración, definición y llamada

**🗃 2. Punteros y Memoria**

**----------------------------------------------------------------------------------**

**[ ] Declaración y uso de punteros básicos**

**[ ] Punteros a punteros**

**[ ] Arrays y relación con punteros**

**[ ] Manejo de strings (cadenas de caracteres)**

**[x] Memoria dinámica: malloc, free (opcional, pero bueno saberlo)**

Memoria dinamica es la parte de la memoria RAM que el programa puede solicitar y liberar en tiempo de ejecución, es decir mientras el programa esta corriendo. En muchos microcontroladores la memoria dinamica esta muy limitada o incluso no se recomienda usarla, en aplicaciones criticas (aviones, dispositivos medicos) suele evitarse totalmente.

**Tipos de memoria en un microcontrolador**

* Flash (ROM o memoria de programa) Donde se almacena el codigo del programa, es no volatil (no se borra al apagar), tamaño fijo según el modelo.
* SRAM (RAM estática) Usada para variables en tiempo de ejecucion, pila (stack) y heap. Es volatil, se divide entre stack, heap, y variables globales/staticas.
* EEPROM (opcional, no todos lo tienen) Memoria no volatil para guardar configuraciones o datos persistentes. Escribible durante la ejecucion, pero mucho mas lenta que la RAM.

**Organización de la SRAM:**

+-------------------------------+

| Variables Globales y |

| Estáticas |

+-------------------------------+

| Heap | <- malloc toma memoria de aquí

| (crece hacia arriba) |

+-------------------------------+

| Espacio Libre |

+-------------------------------+

| Stack | <- funciones locales, parámetros, return address

| (crece hacia abajo) |

+-------------------------------+

Heap: Espacio dinamico de la SRAM, lo gestionamos con malloc calloc, realloc.

Si abusas del uso del heap o stack puedes tener problemas de fragmentacion o quedarte sin memoria

Stack (pila): Area usada por el compilador, guarda variables locales de funciones, guarda direcciones de retorno y parametros en llamadas de funciones. Crece al llamar funciones y se libera al salir de ellas.

Variables globales y estaticas: Estan al principio de la SRAM, siempre existen durante toda la ejecución del programa

**Funciones comunes:**

malloc Reserva un bloque de memoria

free Libera ese bloque de memoria

calloc Reserva un bloque de memoria pero se inicializa la memoria en 0

realloc Cambia el tamaño de un bloque de memoria previamente reservado

# [ ] Diferencias entre stack y heap

Tanto el stack como el heap son particiones logicas de la SRAM. Estas tienen algunas diferencias principales.

Stack: Memoria automática, asignación automática, velocidad rapida, tamaño limitado, uso tipico para variables locales y funciones, mas segura.

Heap: Memoria dinámica, asignacion manual con funciones (malloc()), liberación manual, velocidad lenta, tamaño mayor que el stack pero limitado aun, uso para datos que deben persistir o ser grandes, mas propensa a errores.

El heap, sirve para guardar variables que necesitan persistir y no pueden ser implementadas con el stack o variables globales. Por ejemplo el buffer, si uso una variable local (stack) el buffer se destruye al salir de la funcion, y si uso variables globales tendre problemas si uso 2 o mas buffers. Ahí entra el heap para solucionar el problema.

# ⚙️ 3. Manipulación de Bits y Bajo Nivel

# ---------------------------------------------------------------------------------------------------

# [ ] Operadores bit a bit: &, |, ^, ~, <<, >>

Estos operadores trabajan directamente sobre los bits individuales de los datos. Son fundamentales en programación de sistemas embebidos, donde necesitas controlar registros, puertos, banderas, etc.

| **Operador** | **Nombre** | **Descripción** |
| --- | --- | --- |
| & | AND | Bit a bit: ambos bits deben ser 1 |
| ` | ` | OR |
| ^ | XOR | Bit a bit: uno u otro, pero no ambos |
| ~ | NOT | Invierte cada bit |
| << | Shift left | Desplaza bits a la izquierda |
| >> | Shift right | Desplaza bits a la derecha |

# [ ] Creación y uso de máscaras de bits (bitmask)

# [ ] Uso de bitfields en struct

# [ ] Uso de const

# [ ] Uso de volatile

📚 4. Estructuras de Datos

---------------------------

[ ] Estructuras (struct)

[ ] Uniones (union)

[ ] Enumeraciones (enum)

🏗 5. Preprocesador y Modularidad

----------------------------------

[ ] Uso de macros simples: #define

[ ] Inclusión de archivos: #include

[ ] Separación de código en .c y .h

[ ] Ámbito de variables: static y extern

⚡ 6. Extras para Embedded

---------------------------

[ ] Entender el concepto de interrupciones (solo teoría en C)

[ ] Diferencias entre memoria RAM, Flash y registros en un microcontrolador

[ ] Buenas prácticas de programación (comentarios, legibilidad)

[ ] Técnicas básicas de depuración (imprimir valores, ver en simulador)

⭐ Opcional pero Útil

----------------------

[ ] Conceptos básicos de Makefiles

[ ] Compilación y enlace (.c, .h, .o, .elf)

[ ] Introducción a lectura de datasheets

===============================================