Video4 (1)

Introducción a C – Textos

Conceptos Principales

1. a
2. b
3. c

Notas

- **Propósito del video**: Enseñar el manejo de **textos en C** y cómo se relaciona con la **memoria**, incluyendo la memoria dinámica y el uso de punteros.
- **Manejo de memoria en C**:
  - Cuando se ejecuta un programa, el **sistema operativo** asigna un **pedazo de memoria** al programa.
  - La memoria se distribuye en secciones principales:
    1. **Código**: instrucciones compiladas que se envían al CPU.
    2. **Constantes**: literales definidas en el código.
    3. **Globales**: variables globales con tamaño fijo durante toda la ejecución.
    4. **Heap (memoria dinámica)**: para variables cuyo tamaño no se conoce en tiempo de compilación.
    5. **Stack (pila)**: para variables locales y parámetros de funciones; temporal y automática.
  - Diferencia clave: **stack** es temporal y se limpia automáticamente; **heap** requiere manejo manual (`malloc`, `calloc`, `realloc`, `free`).
- **Textos en C**:
  - No existen strings como en Python; se usan **arreglos de caracteres**.
  - Cada posición del arreglo almacena un **carácter** (`char`).
  - Los strings deben terminar en **carácter nulo (`\0`)** para indicar su final.
  - Para crear un string dinámicamente se usan punteros y funciones de asignación de memoria:
    - `malloc`: asigna memoria sin inicializar.
    - `calloc`: asigna memoria y la inicializa a cero.
    - `realloc`: cambia el tamaño de un bloque ya asignado.
    - `free`: libera memoria para evitar **errores de segmentación** (`segmentation fault`).
- **Uso de punteros**:
  - Un puntero () almacena la **dirección de memoria** donde se guarda la información.
  - Permite manipular directamente la memoria del heap, esencial para textos y estructuras dinámicas.
- **Matrices en C**:
  - Una matriz se representa como un **arreglo de arreglos** (punteros a punteros).

    - Se asigna primero un arreglo de punteros para las filas y luego cada fila con un `malloc` o `calloc`.
    - Las matrices rectangulares requieren cuidado para no acceder a memoria no asignada.
- **Notas importantes**:
    - El tamaño de los tipos de datos puede variar según arquitectura (32 o 64 bits); se recomienda usar `sizeof`.
    - Siempre liberar la memoria asignada dinámicamente para evitar errores.
    - Este video prepara el terreno para manejar cadenas y arreglos complejos con control total de memoria.
- **Resumen conceptual**:
    - **Stack**: temporal, automática, para variables locales.
    - **Heap**: dinámico, manual, para arreglos y textos de tamaño variable.
    - Strings = arreglos de `char` con terminación `\0`.
    - Memoria dinámica = `malloc`, `calloc`, `realloc`, `free`.
    - Matrices = arreglos de punteros a arreglos.