Análisis de Métricas - Parcial Sistemas Operativos

Grupo conformado por:

- *Simon Mazo Gomez*
- *Juan Simon Ospina Martinez*
- *Samuel Samper Cardona*
- *Andres Felipe Velez Alvarez*
- *Sebastian Salazar Henao*

En este documento se presenta el análisis comparativo de rendimiento y uso de memoria en un programa en C/C++ que procesa 10,000,000 de registros de personas. Se evaluaron dos dimensiones principales: Structs vs Clases y Valores vs Apuntadores, siguiendo las métricas solicitadas en el parcial.

Métricas de Ejecución

Enfoque	Generación (ms)	Procesamiento (ms)	Memoria Inicial (KB)	Memoria Final (KB)
Clases + Valores	11824.7	86356.7	2286976	2336676
Clases + Apuntadores	13363.3	91401.2	2521344	2583200
Structs + Valores	11543.0	84649.0	2334720	2384296
Structs + Apuntadores	12300.0	95973.0	2521216	2587248

Análisis de Resultados

Tiempo de Ejecución

El tiempo más rápido se obtuvo con Structs usando valores (11,543 ms en generación y 84,649 ms en procesamiento). El enfoque más lento fue Structs con apuntadores (12,300 ms en generación y 95,973 ms en procesamiento). Se concluye que usar valores es más eficiente que usar apuntadores debido a que se evita el acceso indirecto y la sobrecarga de reservas dinámicas de memoria. Asimismo, los Structs presentaron un mejor rendimiento en tiempo que las Clases.

Consumo de Memoria

El consumo de memoria más bajo se observó en Clases con valores (2,286,976 KB iniciales y 2,336,676 KB finales). En contraste, Structs con apuntadores fueron los de mayor consumo (2,521,216 KB iniciales y 2,587,248 KB finales). Esto demuestra que los apuntadores, en lugar de ahorrar memoria, generan mayor uso debido a la sobrecarga del heap y la fragmentación, además de los bytes extra por cada puntero.

Eficiencia

1. Valores vs. Apuntadores

- **Métricas observadas**:
  - **Valores**: menor tiempo y menor memoria (ej. Structs con valores → 84,649 ms / 2,384,296 KB).
  - **Apuntadores:** mayor tiempo y mayor memoria (ej. Structs con apuntadores → 95,973 ms / 2,587,248 KB).

- **Razones técnicas**:
  1. **Localidad de memoria (cache-friendly):**
     - En arreglos de valores, los datos quedan contiguos → la CPU aprovecha mejor la **caché**.
     - Con apuntadores, cada objeto queda en posiciones distintas del heap → acceso disperso y penalización en rendimiento.
  2. **Sobrecarga de gestión dinámica:**
     - Cada new o malloc en el heap añade metadatos de control → memoria extra.
     - Además, la gestión dinámica introduce **fragmentación**.
  3. **Accesos indirectos:**
     - Con valores → acceso directo al dato.
     - Con punteros → primero se lee la dirección, luego el dato → 1 salto adicional.
- **Conclusión**:
  Aunque intuitivamente se pensaría que guardar un puntero (8 bytes) es más liviano que guardar un objeto, en escenarios masivos con datos completos y únicos **los valores son más eficientes**. Los punteros **solo serían más útiles** si se comparten estructuras repetidas o si no conocemos el tamaño de los datos de antemano.

2. Structs vs. Clases

- **Métricas observadas**:
  - Structs con valores: mejor tiempo (84,649 ms).
  - Clases con valores: ligeramente más lentas (86,356 ms) pero con menos memoria inicial (2,286,976 KB vs. 2,334,720 KB).
- **Razones técnicas**:
  1. **Overhead de Clases:**

        - Las clases pueden tener constructores, métodos y alineación distinta, lo que introduce un poco más de trabajo al procesar.
    2. **Structs más "planos":**
        - En C, los structs son más simples, se manejan como bloques de memoria sin lógica asociada.
        - Esto se refleja en tiempos más bajos de ejecución.
    3. **Gestión de memoria:**
        - Las clases optimizan mejor la disposición interna de atributos → lo que explica el **menor consumo de memoria**.
- **Conclusión**:
  Los **Structs son más rápidos** en operaciones masivas de datos crudos, mientras que las **Clases consumen menos memoria** y ofrecen ventajas de diseño orientado a objetos (encapsulación, reutilización, abstracción).

3. Relación entre tiempo y memoria

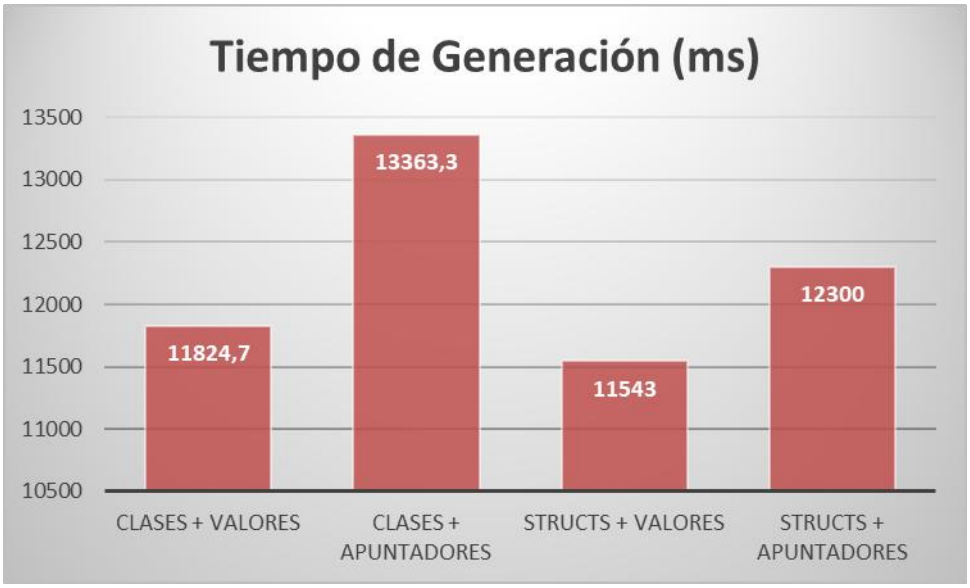
- **Valores** = mejor **eficiencia global** (tiempo + memoria).
- **Apuntadores** = peores resultados en ambos indicadores.
- **Structs** = favorecen tiempo.
- **Clases** = favorecen memoria.

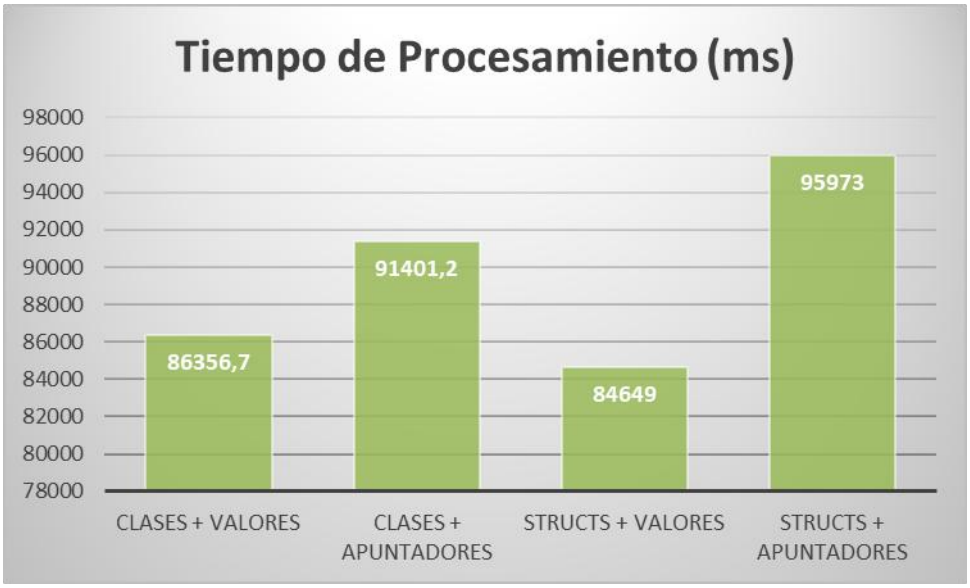
Esto confirma que **no hay un único "mejor" enfoque**, sino que depende de si el criterio prioritario es **velocidad** o **compactación de datos**.

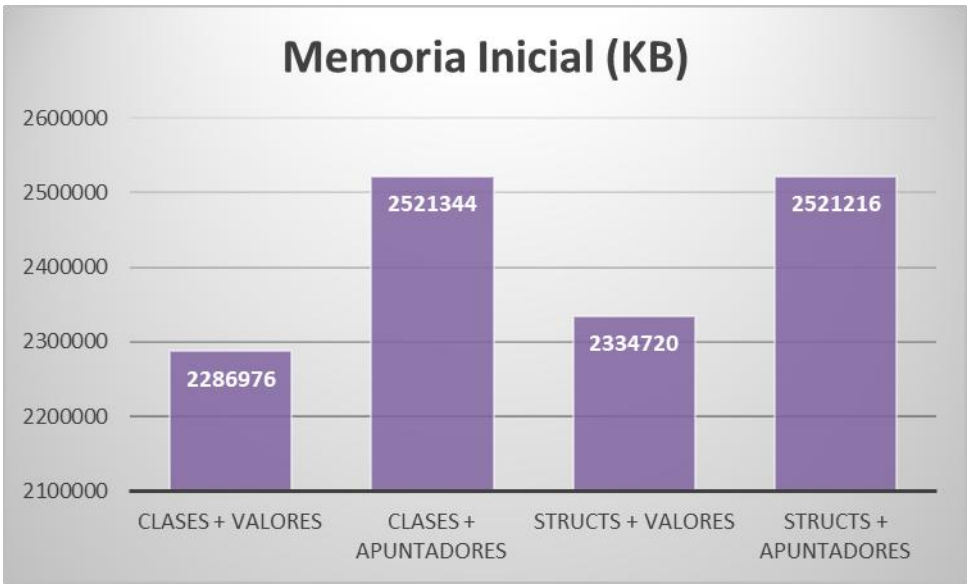
4. Implicaciones prácticas

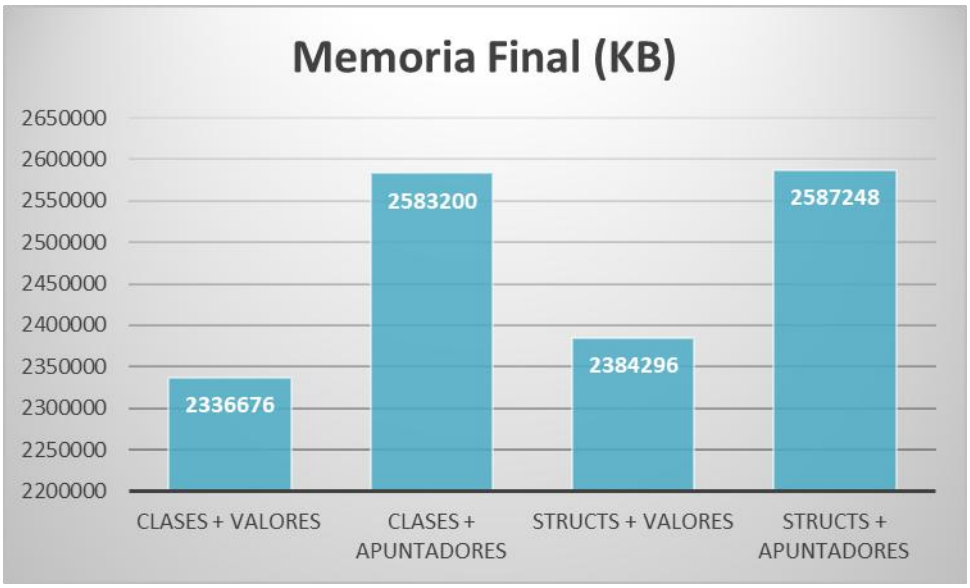
- En **aplicaciones científicas o de simulación** → mejor usar *Structs por valor* (maximizar velocidad).
- En **sistemas con memoria limitada o que requieran modularidad OO** → mejor usar *Clases por valor*.
- El uso de *apuntadores* solo es justificable cuando se busca flexibilidad (estructuras dinámicas, listas enlazadas, árboles, compartir memoria).

Gráficas Comparativas




Tiempo de Procesamiento (ms)
98000
96000
94000
92000
90000
88000
86000
84000
82000
80000
78000
86356,7
91401,2
84649
95973
CLASES + VALORES
CLASES + APUNTADORES
STRUCTS + VALORES
STRUCTS + APUNTADORES


Memoria Inicial (KB)
2600000
2500000
2400000
2300000
2200000
2100000
2286976
2521344
2334720
2521216
CLASES + VALORES
CLASES + APUNTADORES
STRUCTS + VALORES
STRUCTS + APUNTADORES


Memoria Final (KB)
2650000
2600000
2550000
2500000
2450000
2400000
2350000
2300000
2250000
2200000
2336676
2583200
2384296
2587248
CLASES + VALORES
CLASES + APUNTADORES
STRUCTS + VALORES
STRUCTS + APUNTADORES