Parcial 1.
Paradigmas de programación.

1. Para el siguiente enunciado: [Tienes una lista de nombres de estudiantes junto con sus respectivas calificaciones en un examen final. El objetivo es ordenar esta lista en *orden descendente según las calificaciones*, de modo que sea posible identificar fácilmente quién obtuvo las mejores notas. En caso de que *dos estudiantes tengan la misma calificación*, estos deben ordenarse *alfabéticamente por su nombre*. **Resolver el problema de ordenamiento utilizando dos enfoques diferentes, cada uno representando un paradigma de programación distinto (Declarativo, imperativo).** Elabore un análisis comparativo entre los dos enfoques implementados, destacando las diferencias clave entre la programación imperativa** y la **programación declarativa/funcional. El análisis debe incluir:

   - Comparación de claridad y legibilidad del código.
   - Nivel de expresividad y abstracción.
   - Manejo de estructuras de datos (mutabilidad vs inmutabilidad).
   - Manejo de estado en cada paradigma.
   - Facilidad de mantenimiento y extensión de cada enfoque.
   - Eficiencia de cada solución, considerando el algoritmo y el lenguaje utilizado.

2. Estás desarrollando un sistema de gestión de estudiantes para una universidad. Cada estudiante tiene un registro que incluye su nombre, apellido, edad, número de identificación y un conjunto de calificaciones correspondientes a sus materias. Debido a las limitaciones de memoria en el sistema, es fundamental optimizar el uso de memoria al almacenar estos registros. ***Desarrollar un programa en C que gestione de forma dinámica y eficiente la memoria utilizada para almacenar registros de estudiantes, asegurando que se optimice el espacio ocupado y se evite el desperdicio innecesario***.

   El programa debe utilizar `*malloc*` y `*free*` para asignar y liberar memoria dinámicamente a medida que los registros de estudiantes se crean y eliminan. Consideraciones:

   - Cada registro debe ocupar únicamente la memoria necesaria para almacenar la información del estudiante, ajustándose dinámicamente a la longitud de nombres, apellidos y calificaciones.

- Implemente un mecanismo de compactación de memoria, optimizando el espacio usado por los registros de estudiantes. Para ello, utilice:
  - Estructuras (`struct`) optimizadas.
  - Manejo de cadenas dinámicas (`char`) para nombres y apellidos, reservando solo la memoria estrictamente necesaria.
  - Arrays dinámicos para almacenar calificaciones, ajustados al número real de materias.
  - Técnicas de optimización de memoria, como el uso de bitfields si es necesario, para campos pequeños como la edad o el número de identificación.

3. El cálculo lambda es una herramienta matemática para describir el funcionamiento de la computación de forma declarativa. Si se tiene una lista de n números, la forma de calcular el promedio de la lista en Haskell es el siguiente:

```haskell
promedio xs = realToFrac (sum xs) / genericLength xs
main :: IO ()
main = do
    putStrLn "Ingrese una lista de números separados por espacios:"
    entrada <- getLine
    let numeros = map read (words entrada) :: [Double]
    if null numeros
        then putStrLn "La lista está vacía, no se puede calcular el promedio."
        else putStrLn $ "El promedio es: " ++ show (promedio numeros)
```

Escriba en notación de cálculo lambda la implementación de este método para calcular el promedio de números de una lista de tamaño n