Nota:

- **Todos retos deben resolverse exclusivamente usando reasignación de punteros, sin intercambiar valores de nodos**
- **Estos retos no son calificables, es solo para preparación del 2do parcial de ED**

Reto: Eliminar cada N-ésimo nodo de una lista circular (Problema de Josephus)

Descripción:
Dada una **lista circular** con `n` nodos, elimina cada `m`-ésimo nodo hasta que solo quede uno.

Este problema es una versión del famoso **Problema de Josephus**, que tiene aplicaciones en teoría de juegos y programación competitiva.

Ejemplo:

Entrada: `n = 7, m = 3`
Lista inicial: `1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> (ciclo)`
Eliminaciones:

1. Se elimina el nodo `3`: `1 -> 2 -> 4 -> 5 -> 6 -> 7 -> (ciclo)`
2. Se elimina el nodo `6`: `1 -> 2 -> 4 -> 5 -> 7 -> (ciclo)`
3. Se elimina el nodo `2`: `1 -> 4 -> 5 -> 7 -> (ciclo)`
4. Se elimina el nodo `7`: `1 -> 4 -> 5 -> (ciclo)`
5. Se elimina el nodo `5`: `1 -> 4 -> (ciclo)`
6. Se elimina el nodo `1`: `4` (Sobreviviente)

Salida esperada: `El nodo sobreviviente es 4`

💡 Reglas:

1. Implementa una **lista circular** con nodos numerados del `1` al `n`.
2. Implementa un algoritmo para eliminar cada `m`-ésimo nodo.
3. El programa debe imprimir el nodo que **sobrevive al final**.

Reto: Rotar una lista circular N posiciones

Descripción:

Dada una **lista circular simplemente enlazada**, rota la lista `k` posiciones a la derecha.

Ejemplo 1:

Entrada:

Lista inicial:
`1 -> 2 -> 3 -> 4 -> 5 -> (ciclo)`
Rotar `k = 2`

Salida esperada:

Lista rotada:
`4 -> 5 -> 1 -> 2 -> 3 -> (ciclo)`

Ejemplo 2:

Entrada:

Lista inicial:
`10 -> 20 -> 30 -> 40 -> 50 -> (ciclo)`
Rotar `k = 3`

Salida esperada:

Lista rotada:
`30 -> 40 -> 50 -> 10 -> 20 -> (ciclo)`

📌 Reglas:

1. Implementa una **lista circular simplemente enlazada**.
2. Implementa una función que **rote la lista `k` posiciones** a la derecha.
3. Asegúrate de que el ciclo **se mantenga intacto**.
4. Maneja el caso donde `k` es mayor que el tamaño de la lista (usa `k % tamaño`).

Reto: Invertir un grupo de nodos en una lista enlazada

Descripción:

Dada una **lista simplemente enlazada**, invierte los nodos en grupos de tamaño `k`.

Ejemplo 1:

Entrada:

Lista inicial:
`1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> None`
`k = 3`

Salida esperada:

Lista modificada:
`3 -> 2 -> 1 -> 6 -> 5 -> 4 -> 8 -> 7 -> None`

Ejemplo 2:

Entrada:

Lista inicial:
`10 -> 20 -> 30 -> 40 -> 50 -> 60 -> None`
`k = 2`

Salida esperada:

Lista modificada:
`20 -> 10 -> 40 -> 30 -> 60 -> 50 -> None`

📌 Reglas:

1. Implementa una **lista simplemente enlazada**.
2. Implementa una función que **invierta los nodos en grupos de tamaño `k`**.
3. Si el número de nodos no es múltiplo de `k`, deja el grupo final sin invertir.

Reto: Fusionar dos listas enlazadas ordenadas

Descripción:

Dadas dos **listas simplemente enlazadas ordenadas**, fusiónalas en una sola lista también ordenada.

Ejemplo 1:

Entrada:

Lista 1: `1 -> 3 -> 5 -> 7 -> None`
Lista 2: `2 -> 4 -> 6 -> 8 -> None`

Salida esperada:

Lista fusionada: `1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> None`

Ejemplo 2:

Entrada:

Lista 1: `10 -> 20 -> 30 -> None`
Lista 2: `5 -> 15 -> 25 -> 35 -> None`

Salida esperada:

Lista fusionada: `5 -> 10 -> 15 -> 20 -> 25 -> 30 -> 35 -> None`

📌 Reglas:

1. Implementa una **lista simplemente enlazada**.
2. Implementa una función que **fusione las dos listas en orden ascendente**.
3. No puedes usar estructuras auxiliares como listas o arreglos (hazlo con punteros).

Reto: Encontrar el nodo donde dos listas enlazadas se cruzan

Descripción:

Dadas dos **listas simplemente enlazadas**, encuentra el primer nodo donde ambas listas se cruzan. Si no hay intersección, devuelve `None`.

Ejemplo 1:

Entrada:

Lista 1: `1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> None`
Lista 2: `9 -> 8 -> 5 -> 6 -> 7 -> None`

Salida esperada:

Nodo de intersección: `5`

Ejemplo 2:

Entrada:

Lista 1: `10 -> 20 -> 30 -> 40 -> 50 -> None`
Lista 2: `5 -> 15 -> 25 -> None`

Salida esperada:

`None` (No hay intersección)

📌 Reglas:

1. Implementa una **lista simplemente enlazada**.
2. Implementa una función que encuentre el **primer nodo común** entre dos listas.
3. Si no hay intersección, la función debe devolver `None`.
4. No puedes modificar las listas originales.

Reto: Reorganizar una lista enlazada en forma de zigzag

Descripción:

Dada una **lista simplemente enlazada**, reorganízala de tal forma que los nodos sigan un **patrón de zigzag**, es decir:

Si la lista es `a -> b -> c -> d -> e -> f -> None`, entonces la salida debe ser:
`a -> c -> b -> e -> d -> f -> None`

Es decir, el segundo nodo se mueve después del tercer nodo, el cuarto nodo después del quinto, y así sucesivamente.

Ejemplo 1:

Entrada:

Lista inicial:
```
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> None
```

Salida esperada:

Lista reorganizada:
```
1 -> 3 -> 2 -> 5 -> 4 -> 6 -> None
```

Ejemplo 2:

Entrada:

Lista inicial:
```
10 -> 20 -> 30 -> 40 -> 50 -> None
```

Salida esperada:

Lista reorganizada:
```
10 -> 30 -> 20 -> 50 -> 40 -> None
```

📌 Reglas:

1. Implementa una **lista simplemente enlazada**.
2. Implementa una función que **reorganice la lista en zigzag** sin cambiar los valores de los nodos, solo los enlaces.
3. No puedes usar estructuras auxiliares como listas o arreglos.

Reto: Invertir grupos de K nodos en una lista enlazada

Descripción:

Dada una **lista simplemente enlazada** y un número `K`, invierte la lista en **grupos de K nodos**. Si al final quedan menos de `K` nodos, déjalos en el mismo orden.

Ejemplo 1:

Entrada:

Lista inicial:
```
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> None
K = 3
```

Salida esperada:

Lista reorganizada:
```
3 -> 2 -> 1 -> 6 -> 5 -> 4 -> 7 -> 8 -> None
```

Ejemplo 2:

Entrada:

Lista inicial:
```
10 -> 20 -> 30 -> 40 -> 50 -> None
K = 2
```

Salida esperada:

Lista reorganizada:
```
20 -> 10 -> 40 -> 30 -> 50 -> None
```

📌 Reglas:

1. Implementa una **lista simplemente enlazada**.
2. Implementa una función que **invierta la lista en grupos de K nodos**.
3. Si el número de nodos no es múltiplo de `K`, los últimos nodos deben permanecer en el mismo orden.
4. No puedes usar estructuras auxiliares como listas o arreglos.

Reto: Mover todos los nodos pares al final de la lista

Descripción:

Dada una **lista simplemente enlazada**, reordena los nodos de tal forma que todos los nodos en **posiciones pares** sean movidos al final de la lista, **manteniendo su orden relativo**.

Ejemplo 1:

Entrada:

Lista inicial:
```
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> None
```

Salida esperada:

Lista reorganizada:
`1 -> 3 -> 5 -> 2 -> 4 -> 6 -> None`

Ejemplo 2:

Entrada:

Lista inicial:
`10 -> 20 -> 30 -> 40 -> 50 -> None`

Salida esperada:

Lista reorganizada:
`10 -> 30 -> 50 -> 20 -> 40 -> None`

📌 Reglas:

1. Implementa una **lista simplemente enlazada**.
2. Implementa una función que **mueva los nodos en posiciones pares al final de la lista**.
3. Mantén el orden relativo de los nodos impares y pares.
4. No puedes usar estructuras auxiliares como listas o arreglos

Reto: Dividir la lista en dos mitades y revertir la segunda mitad

Descripción:

Dada una **lista simplemente enlazada**, realiza las siguientes acciones:

1. Divide la lista en **dos mitades**.
2. **Invierte la segunda mitad** usando solo reasignación de punteros.
3. Devuelve la lista como dos partes separadas.

Ejemplo 1:

Entrada:

`1 -> 2 -> 3 -> 4 -> 5 -> 6 -> None`

Salida esperada:

- Primera mitad: `1 -> 2 -> 3 -> None`

- Segunda mitad (invertida): `6 -> 5 -> 4 -> None`

Ejemplo 2:

Entrada:

```
10 -> 20 -> 30 -> 40 -> 50 -> None
```

Salida esperada:

- Primera mitad: `10 -> 20 -> None`
- Segunda mitad (invertida): `50 -> 40 -> 30 -> None`

Reglas:

- Solo puedes usar **punteros**, sin arreglos ni listas auxiliares.
- **No puedes intercambiar valores entre nodos**.
- Si la lista tiene un número impar de elementos, deja el nodo del medio en la primera mitad.
- Implementa una función que reciba la lista y devuelva los **dos punteros al inicio de cada mitad** (la primera normal, la segunda invertida).