

Nombre : Lilibeth Tatiana Gomez Mantilla
Jimena Sofia Valiente Blandon

TALLER – LISTAS EN C#

Ejercicio 1: Contar Nodos Escribe un método int ContarNodos() que recorra una lista enlazada y devuelva el número total de nodos.

Script :

```
using System;
```

```
// Clase Nodo que representa un elemento de la lista
```

```
public class Nodo<T>
```

```
{
```

```
    public T Valor; // Valor almacenado en el nodo
```

```
    public Nodo<T> Siguiente; // Referencia al siguiente nodo en la lista
```

```
    // Constructor que inicializa el nodo con un valor
```

```
    public Nodo(T valor)
```

```
    {
```

```
        Valor = valor;
```

```
        Siguiente = null; // Al principio, no hay siguiente nodo
```

```
    }
```

```
}
```

```
// Clase ListaEnlazada que gestiona los nodos
```

```
public class ListaEnlazada<T>
```

```
{
```

```
    public Nodo<T> Cabeza; // Primer nodo de la lista
```

```
    // Constructor que inicializa la lista vacía
```

```
    public ListaEnlazada()
```

```
    {
```

```
        Cabeza = null;
```

```
    }
```

```
    // Método para agregar un nuevo valor al final de la lista
```

```
    public void Agregar(T valor)
```

```
    {
```

```
        Nodo<T> nuevo = new Nodo<T>(valor); // Crear un nuevo nodo con el valor dado
```

```
        if (Cabeza == null) // Si la lista está vacía
```

```
        {
```

```
            Cabeza = nuevo; // El nuevo nodo se convierte en la cabeza
```

```
        }
```

```
        else
```

```
        {
```

```

        Nodo<T> actual = Cabeza; // Comenzar desde la cabeza

        // Recorrer la lista hasta el último nodo
        while (actual.Siguiente != null)
        {
            actual = actual.Siguiente;
        }

        actual.Siguiente = nuevo; // Agregar el nuevo nodo al final
    }
}

// Método para contar cuántos nodos hay en la lista
public int ContarNodos()
{
    int contador = 0; // Inicializar contador
    Nodo<T> actual = Cabeza; // Comenzar desde la cabeza

    // Recorrer la lista mientras haya nodos
    while (actual != null)
    {
        contador++; // Incrementar el contador
        actual = actual.Siguiente; // Pasar al siguiente nodo
    }

    return contador; // Devolver el total de nodos
}

// Método para mostrar todos los elementos de la lista
public void MostrarElementos()
{
    Nodo<T> actual = Cabeza; // Comenzar desde la cabeza
    Console.WriteLine("---Elementos en la lista:---");

    // Recorrer la lista y mostrar cada valor
    while (actual != null)
    {
        Console.WriteLine("* " + actual.Valor); // Mostrar el valor del nodo
        actual = actual.Siguiente; // Pasar al siguiente nodo
    }
}

// Clase principal con el punto de entrada del programa
class Program
{
    static void Main()
    {

```

```

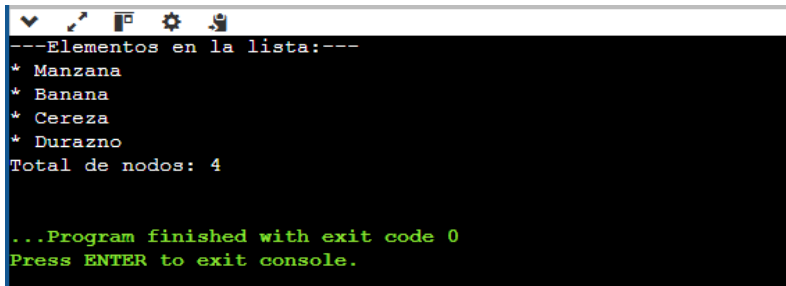
// Crear una lista enlazada de tipo string
ListaEnlazada<string> lista = new ListaEnlazada<string>();

// Agregar elementos a la lista
lista.Agregar("Manzana");
lista.Agregar("Banana");
lista.Agregar("Cereza");
lista.Agregar("Durazno");

// Mostrar los elementos en la terminal
lista.MostrarElementos();

// Mostrar el número total de nodos
Console.WriteLine("Total de nodos: " + lista.ContarNodos());
}
}

```



```

---Elementos en la lista:---
* Manzana
* Banana
* Cereza
* Durazno
Total de nodos: 4

...Program finished with exit code 0
Press ENTER to exit console.

```

Ejercicio 2: Buscar un Valor Crea un método bool Buscar(int valor) que devuelva true si un valor específico se encuentra en la lista, y false en caso contrario.

Script

```
using System;
```

```
// Clase Nodo genérica
```

```
public class Nodo<T>
```

```
{
```

```
    public T Valor;           // Valor almacenado en el nodo
```

```
    public Nodo<T> Siguiente; // Referencia al siguiente nodo
```

```
    public Nodo(T valor)
```

```
{
```

```
        Valor = valor;
```

```
        Siguiente = null;
```

```
}
```

```
}
```

```
// Clase ListaEnlazada genérica
```

```
public class ListaEnlazada<T>
```

```
{
```

```
    public Nodo<T> Cabeza;    // Primer nodo de la lista
```

```
    public ListaEnlazada()
```

```
    {
```

```
        Cabeza = null;
```

```
    }
```

```
// Agrega un nuevo nodo al final de la lista
```

```
public void Agregar(T valor)
```

```
{
```

```
    Nodo<T> nuevo = new Nodo<T>(valor);
```

```
    if (Cabeza == null)
```

```
    {
```

```
        Cabeza = nuevo;
```

```
    }
```

```
    else
```

```
    {
```

```
        Nodo<T> actual = Cabeza;
```

```
        while (actual.Siguiente != null)
```

```
        {
```

```
            actual = actual.Siguiente;
```

```
        }
```

```
        actual.Siguiente = nuevo;
```

```
    }
```

```
}
```

```
// Cuenta cuántos nodos hay en la lista
```

```
public int ContarNodos()
{
    int contador = 0;
    Nodo<T> actual = Cabeza;

    while (actual != null)
    {
        contador++;
        actual = actual.Siguiente;
    }

    return contador;
}
```

// Muestra todos los elementos de la lista

```
public void MostrarElementos()
{
    Nodo<T> actual = Cabeza;
    Console.WriteLine("Elementos en la lista:");

    while (actual != null)
    {
        Console.WriteLine("- " + actual.Valor);
        actual = actual.Siguiente;
    }
}
```

// Busca un valor en la lista y devuelve true si lo encuentra

```
public bool Buscar(T valor)
{
    Nodo<T> actual = Cabeza;
```

```

while (actual != null)
{
    if (actual.Valor.Equals(valor))
    {
        return true;
    }
    actual = actual.Siguiente;
}

return false;
}
}

```

// Programa principal

class Program

```

{
    static void Main()
    {
        // Crear una lista enlazada de enteros
        ListaEnlazada<int> lista = new ListaEnlazada<int>();

        // Agregar elementos a la lista
        lista.Agregar(200);
        lista.Agregar(215);
        lista.Agregar(47);
        lista.Agregar(22);

        // Mostrar elementos
        lista.MostrarElementos();

        // Mostrar cantidad de nodos
        Console.WriteLine("Total de nodos: " + lista.ContarNodos());
    }
}

```

```

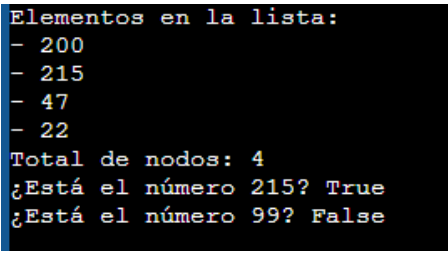
        // Buscar valores

        Console.WriteLine("¿Está el número 215? " + lista.Buscar(215)); // true

        Console.WriteLine("¿Está el número 99? " + lista.Buscar(99)); // false

    }
}

```



```

Elementos en la lista:
- 200
- 215
- 47
- 22
Total de nodos: 4
¿Está el número 215? True
¿Está el número 99? False

```

Ejercicio 3: Encontrar el Último Nodo Implementa un método `Nodo ObtenerUltimoNodo()` que recorra la lista y devuelva una referencia al último nodo sin eliminarlo.

Script:

```
using System;
```

```
// Nodo de la lista enlazada
```

```
public class Nodo<T>
```

```
{
```

```
    public T Valor;           // Dato almacenado
```

```
    public Nodo<T> Siguiente; // Referencia al siguiente nodo
```

```
    public Nodo(T valor)
```

```
    {
```

```
        Valor = valor;
```

```
        Siguiente = null;
```

```
    }
```

```
}
```

```
// Lista enlazada
```

```

public class ListaEnlazada<T>
{
    public Nodo<T> Cabeza;    // Primer nodo de la lista

    public ListaEnlazada()
    {
        Cabeza = null;    // Lista vacía al inicio
    }

    // Agrega un elemento al final de la lista
    public void Agregar(T valor)
    {
        var nuevo = new Nodo<T>(valor); // Crear nuevo nodo

        if (Cabeza == null)
        {
            Cabeza = nuevo;    // Si lista está vacía, nuevo nodo es la cabeza
        }
        else
        {
            var actual = Cabeza;

            // Recorrer hasta el último nodo
            while (actual.Siguiente != null) actual = actual.Siguiente;
            actual.Siguiente = nuevo; // Enlazar nuevo nodo al final
        }
    }

    // Muestra todos los elementos de la lista
    public void MostrarElementos()
    {
        Console.WriteLine("Elementos en la lista:");
    }
}

```



```

var actual = Cabeza;

// Recorrer y mostrar cada nodo
while (actual != null)
{
    Console.WriteLine("- " + actual.Valor);
    actual = actual.Siguiente;
}

}

// Obtiene el último nodo sin eliminarlo
public Nodo<T> ObtenerUltimoNodo()
{
    if (Cabeza == null) return null; // Si lista está vacía

    var actual = Cabeza;

    // Recorrer hasta el último nodo
    while (actual.Siguiente != null) actual = actual.Siguiente;

    return actual; // Retornar último nodo encontrado
}

}

// Programa principal
class Program
{
    static void Main()
    {
        // Crear lista de días de la semana
        var lista = new ListaEnlazada<string>();
    }
}

```

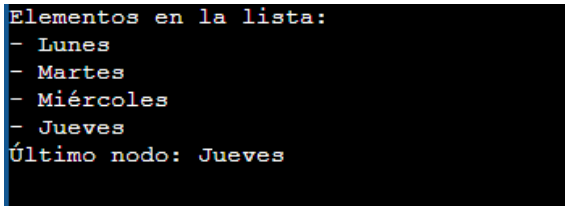
```

// Agregar elementos a la lista
lista.Agregar("Lunes");
lista.Agregar("Martes");
lista.Agregar("Miércoles");
lista.Agregar("Jueves");

// Mostrar todos los elementos
lista.MostrarElementos();

// Obtener y mostrar el último elemento
var ultimo = lista.ObtenerUltimoNodo();
if (ultimo != null)
{
    Console.WriteLine("Último nodo: " + ultimo.Valor);
}
else
{
    Console.WriteLine("La lista está vacía.");
}
}
}

```



```

Elementos en la lista:
- Lunes
- Martes
- Miércoles
- Jueves
Último nodo: Jueves

```

Ejercicio 4: Sumar Todos los Valores Escribe un método `int SumarValores()` que sume los datos de todos los nodos en una lista enlazada de enteros.

Script:

```
using System;
```

```
// Clase Nodo
```

```
public class Nodo<T>
{
    public T Valor;
    public Nodo<T> Siguiente;

    public Nodo(T valor)
    {
        Valor = valor;
        Siguiente = null;
    }
}
```

// Clase ListaEnlazada para enteros

```
public class ListaEnlazadaEnteros
```

```
{
    public Nodo<int> Cabeza;
```

```
    public ListaEnlazadaEnteros()
```

```
{
    Cabeza = null;
}
```

// Agrega un nuevo nodo al final de la lista

```
public void Agregar(int valor)
```

```
{
    Nodo<int> nuevo = new Nodo<int>(valor);
    if (Cabeza == null)
    {
        Cabeza = nuevo;
    }
    else
    {
```

```

        Nodo<int> actual = Cabeza;
        while (actual.Siguiente != null)
        {
            actual = actual.Siguiente;
        }
        actual.Siguiente = nuevo;
    }
}

```

// Suma todos los valores de los nodos

```

public int SumarValores()
{
    int suma = 0;
    Nodo<int> actual = Cabeza;

    while (actual != null)
    {
        suma += actual.Valor;
        actual = actual.Siguiente;
    }

    return suma;
}

```

// Muestra los elementos de la lista

```

public void MostrarElementos()
{
    Nodo<int> actual = Cabeza;
    Console.WriteLine("Elementos en la lista:");

    while (actual != null)
    {

```

```

        Console.WriteLine("- " + actual.Valor);
        actual = actual.Siguiente;
    }
}
}

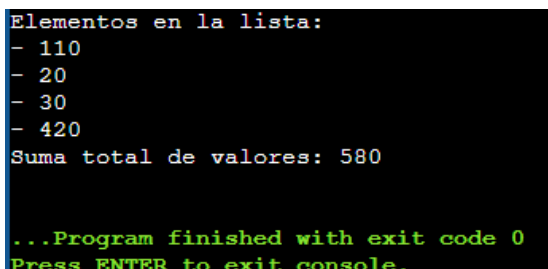
// Programa principal
class Program
{
    static void Main()
    {
        ListaEnlazadaEnteros lista = new ListaEnlazadaEnteros();

        // Agregar valores
        lista.Agregar(110);
        lista.Agregar(20);
        lista.Agregar(30);
        lista.Agregar(420);

        // Mostrar elementos
        lista.MostrarElementos();

        // Mostrar suma total
        Console.WriteLine("Suma total de valores: " + lista.SumarValores());
    }
}

```



```

Programa principal
Elementos en la lista:
- 110
- 20
- 30
- 420
Suma total de valores: 580

...Program finished with exit code 0
Press ENTER to exit console.

```

Ejercicio 5: Encontrar el Valor Máximo Crea un método `int EncontrarMaximo()` que encuentre y devuelva el valor más alto almacenado en la lista. Debe manejar el caso de una lista vacía.

Script

```
using System;
```

```
// Clase Nodo que representa un elemento de la lista
```

```
public class Nodo<T>
```

```
{
```

```
    public T Valor;          // Valor almacenado en el nodo
```

```
    public Nodo<T> Siguiente; // Referencia al siguiente nodo
```

```
// Constructor que inicializa el nodo con un valor
```

```
public Nodo(T valor)
```

```
{
```

```
    Valor = valor;
```

```
    Siguiente = null;      // Al principio, no hay siguiente nodo
```

```
}
```

```
}
```

```
// Clase ListaEnlazada especializada para enteros
```

```
public class ListaEnlazadaEnteros
```

```
{
```

```
    public Nodo<int> Cabeza; // Primer nodo de la lista
```

```
// Constructor que inicializa la lista vacía
```

```
public ListaEnlazadaEnteros()
```

```
{
```

```
    Cabeza = null;
```

```
}
```

```
// Método para agregar un nuevo valor al final de la lista
```

```

public void Agregar(int valor)
{
    Nodo<int> nuevo = new Nodo<int>(valor); // Crear nuevo nodo

    if (Cabeza == null) // Si la lista está vacía
    {
        Cabeza = nuevo; // El nuevo nodo se convierte en la cabeza
    }
    else
    {
        Nodo<int> actual = Cabeza; // Comenzar desde la cabeza

        // Recorrer hasta el último nodo
        while (actual.Siguiente != null)
        {
            actual = actual.Siguiente;
        }

        actual.Siguiente = nuevo; // Agregar el nuevo nodo al final
    }
}

// Método para mostrar todos los elementos de la lista
public void MostrarElementos()
{
    Nodo<int> actual = Cabeza;
    Console.WriteLine("Elementos en la lista:");

    // Recorrer la lista y mostrar cada valor
    while (actual != null)
    {

```

```

        Console.WriteLine("- " + actual.Valor);
        actual = actual.Siguiente;
    }
}

// Método para encontrar el valor máximo en la lista
public int EncontrarMaximo()
{
    if (Cabeza == null)
    {
        Console.WriteLine("La lista está vacía.");
        return int.MinValue; // Indicador de lista vacía
    }

    int maximo = Cabeza.Valor; // Inicializar con el primer valor
    Nodo<int> actual = Cabeza.Siguiente; // Comenzar desde el segundo nodo

    // Recorrer la lista comparando valores
    while (actual != null)
    {
        if (actual.Valor > maximo)
        {
            maximo = actual.Valor; // Actualizar si se encuentra un valor mayor
        }
        actual = actual.Siguiente;
    }

    return maximo; // Devolver el valor máximo encontrado
}
}

```

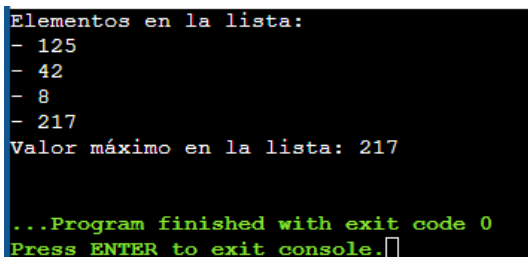


```
// Clase principal con el punto de entrada del programa
class Program
{
    static void Main()
    {
        ListaEnlazadaEnteros lista = new ListaEnlazadaEnteros(); // Crear lista de enteros

        // Agregar valores a la lista
        lista.Agregar(125);
        lista.Agregar(42);
        lista.Agregar(8);
        lista.Agregar(217);

        // Mostrar los elementos en consola
        lista.MostrarElementos();

        // Obtener y mostrar el valor máximo
        int maximo = lista.EncontrarMaximo();
        if (maximo != int.MinValue)
        {
            Console.WriteLine("Valor máximo en la lista: " + maximo);
        }
    }
}
```



```
Elementos en la lista:
- 125
- 42
- 8
- 217
Valor máximo en la lista: 217

...Program finished with exit code 0
Press ENTER to exit console.
```

Ejercicio 6: Convertir a Arreglo Implementa un método `int[] ConvertirAArreglo()` que cree y devuelva un arreglo de enteros con todos los elementos de la lista enlazada en el mismo

orden.

Script

```
using System;
```

```
public class Nodo<T>
{
    public T Valor;          // Valor almacenado en el nodo
    public Nodo<T> Siguiente; // Referencia al siguiente nodo

    public Nodo(T valor)
    {
        Valor = valor;
        Siguiente = null;
    }
}
```

```
public class ListaEnlazadaEnteros
{
    public Nodo<int> Cabeza; // Primer nodo de la lista

    public ListaEnlazadaEnteros()
    {
        Cabeza = null;
    }

    // Agrega un nuevo nodo al final de la lista
    public void Agregar(int valor)
    {
        Nodo<int> nuevo = new Nodo<int>(valor);
        if (Cabeza == null)
        {
            Cabeza = nuevo;
        }
    }
}
```

```

    }
    else
    {
        Nodo<int> actual = Cabeza;
        while (actual.Siguiente != null)
        {
            actual = actual.Siguiente;
        }
        actual.Siguiente = nuevo;
    }
}

```

// Cuenta cuántos nodos hay en la lista

```

public int ContarNodos()
{
    int contador = 0;
    Nodo<int> actual = Cabeza;
    while (actual != null)
    {
        contador++;
        actual = actual.Siguiente;
    }
    return contador;
}

```

// Convierte la lista enlazada en un arreglo de enteros

```

public int[] ConvertirAArreglo()
{
    int tamaño = ContarNodos();    // Obtener el número de elementos
    int[] arreglo = new int[tamaño];    // Crear arreglo del tamaño adecuado

    Nodo<int> actual = Cabeza;

```

```

int indice = 0;

// Recorrer la lista y copiar los valores al arreglo
while (actual != null)
{
    arreglo[indice] = actual.Valor;
    actual = actual.Siguiente;
    indice++;
}

return arreglo; // Devolver el arreglo completo
}

// Muestra los elementos de la lista
public void MostrarElementos()
{
    Nodo<int> actual = Cabeza;
    Console.WriteLine("Elementos en la lista:");
    while (actual != null)
    {
        Console.WriteLine("- " + actual.Valor);
        actual = actual.Siguiente;
    }
}

}

class Program
{
    static void Main()
    {
        ListaEnlazadaEnteros lista = new ListaEnlazadaEnteros();
    }
}

```

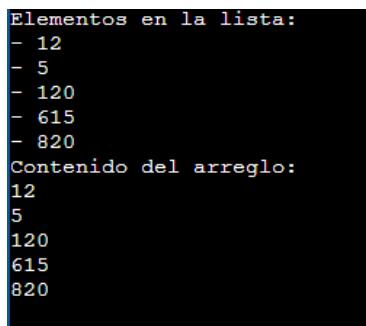
```

// Agregar valores a la lista
lista.Agregar(12);
lista.Agregar(5);
lista.Agregar(120);
lista.Agregar(615);
lista.Agregar(820);

// Mostrar elementos
lista.MostrarElementos();

// Convertir a arreglo y mostrarlo
int[] arreglo = lista.ConvertirAArreglo();
Console.WriteLine("Contenido del arreglo:");
foreach (int valor in arreglo)
{
    Console.WriteLine(valor);
}
}
}

```



```

Elementos en la lista:
- 12
- 5
- 120
- 615
- 820
Contenido del arreglo:
12
5
120
615
820

```

Ejercicio 7: Insertar Producto Ordenado por Precio Crea un método void InsertarProductoOrdenado(Producto nuevoProducto) que inserte un nuevo producto en la lista de manera que la lista se mantenga ordenada de menor a mayor precio.

Script

```
using System;
```

```
public class Producto
{
    public string Nombre;
    public decimal Precio;

    public Producto(string nombre, decimal precio)
    {
        Nombre = nombre;
        Precio = precio;
    }
}
```

```
public class NodoProducto
{
    public Producto Valor;
    public NodoProducto Siguiente;

    public NodoProducto(Producto valor)
    {
        Valor = valor;
        Siguiente = null;
    }
}
```

```
public class ListaProductos
{
    public NodoProducto Cabeza;

    public ListaProductos()
    {
        Cabeza = null;
    }
}
```

```
}
```

```
// Inserta un producto manteniendo el orden por precio (menor a mayor)
```

```
public void InsertarProductoOrdenado(Producto nuevoProducto)
```

```
{
```

```
    NodoProducto nuevoNodo = new NodoProducto(nuevoProducto);
```

```
    // Si la lista está vacía o el nuevo producto tiene menor precio que la cabeza
```

```
    if (Cabeza == null || nuevoProducto.Precio < Cabeza.Valor.Precio)
```

```
    {
```

```
        nuevoNodo.Siguiente = Cabeza;
```

```
        Cabeza = nuevoNodo;
```

```
    }
```

```
    else
```

```
    {
```

```
        NodoProducto actual = Cabeza;
```

```
        // Buscar la posición correcta para insertar
```

```
        while (actual.Siguiente != null && actual.Siguiente.Valor.Precio <
nuevoProducto.Precio)
```

```
        {
```

```
            actual = actual.Siguiente;
```

```
        }
```

```
        // Insertar el nuevo nodo en la posición encontrada
```

```
        nuevoNodo.Siguiente = actual.Siguiente;
```

```
        actual.Siguiente = nuevoNodo;
```

```
    }
```

```
}
```

```
public void MostrarProductos()
```

```
{
```

```

    NodoProducto actual = Cabeza;

    Console.WriteLine("Productos ordenados por precio:");

    while (actual != null)
    {
        Console.WriteLine($"- {actual.Valor.Nombre}: ${actual.Valor.Precio:N0} COP");
        actual = actual.Siguiente;
    }
}

```

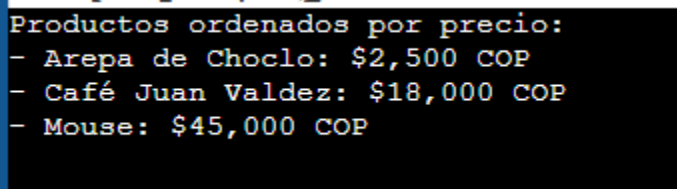
```

class Program
{
    static void Main()
    {
        ListaProductos lista = new ListaProductos();

        // Solo 3 productos
        lista.InsertarProductoOrdenado(new Producto("Café Juan Valdez", 18000m));
        lista.InsertarProductoOrdenado(new Producto("Mouse", 45000m));
        lista.InsertarProductoOrdenado(new Producto("Arepa de Choclo", 2500m));

        lista.MostrarProductos();
    }
}

```



```

Productos ordenados por precio:
- Arepa de Choclo: $2,500 COP
- Café Juan Valdez: $18,000 COP
- Mouse: $45,000 COP

```

Ejercicio 8: Buscar Producto por ID Escribe un método Producto BuscarPorId(int id) que busque un producto por su Id y devuelva el objeto Producto. Si no lo encuentra, debe

devolver null.

Script

```
using System;
```

```
// Clase Producto con ID, nombre y precio
```

```
public class Producto
```

```
{
```

```
    public int Id;
```

```
    public string Nombre;
```

```
    public decimal Precio;
```

```
    public Producto(int id, string nombre, decimal precio)
```

```
    {
```

```
        Id = id;
```

```
        Nombre = nombre;
```

```
        Precio = precio;
```

```
    }
```

```
}
```

```
// Nodo que almacena un Producto
```

```
public class NodoProducto
```

```
{
```

```
    public Producto Valor;
```

```
    public NodoProducto Siguiente;
```

```
    public NodoProducto(Producto valor)
```

```
    {
```

```
        Valor = valor;
```

```
        Siguiente = null;
```

```
    }
```

```
}
```

// Lista enlazada de productos

public class ListaProductos

{

public NodoProducto Cabeza;

public ListaProductos()

{

Cabeza = null;

}

// Agrega un producto al final de la lista

public void AgregarProducto(Producto nuevoProducto)

{

NodoProducto nuevoNodo = new NodoProducto(nuevoProducto);

if (Cabeza == null)

{

Cabeza = nuevoNodo;

}

else

{

NodoProducto actual = Cabeza;

while (actual.Siguiente != null)

{

actual = actual.Siguiente;

}

actual.Siguiente = nuevoNodo;

}

}

// Busca un producto por su ID

public Producto BuscarPorId(int id)

```

{
    NodoProducto actual = Cabeza;

    while (actual != null)
    {
        if (actual.Valor.Id == id)
        {
            return actual.Valor; // Producto encontrado
        }
        actual = actual.Siguiente;
    }

    return null; // No se encontró el producto
}

// Muestra todos los productos
public void MostrarProductos()
{
    NodoProducto actual = Cabeza;
    Console.WriteLine("Lista de productos:");

    while (actual != null)
    {
        Console.WriteLine($"ID: {actual.Valor.Id}, Nombre: {actual.Valor.Nombre}, Precio:
        ${actual.Valor.Precio:N0} COP");
        actual = actual.Siguiente;
    }
}
}

```

```

// Programa principal
class Program

```

```

{
    static void Main()
    {
        ListaProductos lista = new ListaProductos();

        // Agregar productos
        lista.AgregarProducto(new Producto(101, "Mouse", 25990));
        lista.AgregarProducto(new Producto(102, "Teclado", 45500));
        lista.AgregarProducto(new Producto(103, "Monitor", 199990));

        // Mostrar todos los productos
        lista.MostrarProductos();

        // Buscar un ID que sí existe
        int idExistente = 102;
        Producto productoEncontrado = lista.BuscarPorId(idExistente);
        if (productoEncontrado != null)
        {
            Console.WriteLine($"Producto encontrado (ID {idExistente}):
{productoEncontrado.Nombre}, Precio: ${productoEncontrado.Precio:N0} COP");
        }
        else
        {
            Console.WriteLine($"No se encontró ningún producto con ID {idExistente}.");
        }

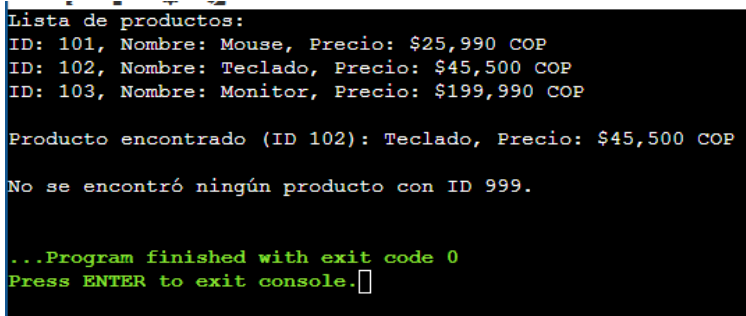
        // Buscar un ID que no existe
        int idInexistente = 999;
        Producto productoNoEncontrado = lista.BuscarPorId(idInexistente);
        if (productoNoEncontrado != null)
        {
            Console.WriteLine($"Producto encontrado (ID {idInexistente}):

```

```

{productoNoEncontrado.Nombre}, Precio: ${productoNoEncontrado.Precio:N0} COP");
    }
    else
    {
        Console.WriteLine($"No se encontró ningún producto con ID {idInexistente}.");
    }
}
}
}

```



```

Lista de productos:
ID: 101, Nombre: Mouse, Precio: $25,990 COP
ID: 102, Nombre: Teclado, Precio: $45,500 COP
ID: 103, Nombre: Monitor, Precio: $199,990 COP

Producto encontrado (ID 102): Teclado, Precio: $45,500 COP

No se encontró ningún producto con ID 999.

...Program finished with exit code 0
Press ENTER to exit console.

```

Ejercicio 9: Actualizar Precio de Producto Implementa un método `bool ActualizarPrecio(int id, double nuevoPrecio)` que busque un producto por su `Id` y actualice su precio. Devuelve `true` si la actualización fue exitosa y `false` si no se encontró el producto.

Script

```

using System;

// Clase Producto con ID, nombre y precio
public class Producto
{
    public int Id;
    public string Nombre;
    public double Precio;

    public Producto(int id, string nombre, double precio)
    {
        Id = id;
        Nombre = nombre;
    }
}

```

```
        Precio = precio;
    }
}
```

// Nodo que almacena un Producto

```
public class NodoProducto
{
    public Producto Valor;
    public NodoProducto Siguiente;

    public NodoProducto(Producto valor)
    {
        Valor = valor;
        Siguiente = null;
    }
}
```

// Lista enlazada de productos

```
public class ListaProductos
{
    public NodoProducto Cabeza;

    public ListaProductos()
    {
        Cabeza = null;
    }
}
```

// Agrega un producto al final de la lista

```
public void AgregarProducto(Producto nuevoProducto)
{
    NodoProducto nuevoNodo = new NodoProducto(nuevoProducto);
```

```

if (Cabeza == null)
{
    Cabeza = nuevoNodo;
}
else
{
    NodoProducto actual = Cabeza;
    while (actual.Siguiente != null)
    {
        actual = actual.Siguiente;
    }
    actual.Siguiente = nuevoNodo;
}
}

```

```

// Actualiza el precio de un producto por su ID
public bool ActualizarPrecio(int id, double nuevoPrecio)
{
    NodoProducto actual = Cabeza;

    while (actual != null)
    {
        if (actual.Valor.Id == id)
        {
            actual.Valor.Precio = nuevoPrecio;
            return true; // Actualización exitosa
        }
        actual = actual.Siguiente;
    }
}

```

```

        return false; // Producto no encontrado
    }

    // Muestra todos los productos
    public void MostrarProductos()
    {
        NodoProducto actual = Cabeza;

        Console.WriteLine("Lista de productos:");

        while (actual != null)
        {
            Console.WriteLine($"ID: {actual.Valor.Id}, Nombre: {actual.Valor.Nombre}, Precio:
            ${actual.Valor.Precio:N0} COP");
            actual = actual.Siguiente;
        }
    }
}

// Programa principal
class Program
{
    static void Main()
    {
        ListaProductos lista = new ListaProductos();

        // Agregar productos
        lista.AgregarProducto(new Producto(101, "Mouse", 25990));
        lista.AgregarProducto(new Producto(102, "Teclado", 45500));
        lista.AgregarProducto(new Producto(103, "Monitor", 199990));

        // Mostrar productos originales
        Console.WriteLine("Antes de actualizar:");
    }
}

```



```

lista.MostrarProductos();

// Actualizar precio de un producto existente
bool actualizado = lista.ActualizarPrecio(102, 49900);
Console.WriteLine($"¿Actualización exitosa para ID 102? {actualizado}");

// Intentar actualizar un producto inexistente
bool noEncontrado = lista.ActualizarPrecio(999, 99900);
Console.WriteLine($"¿Actualización exitosa para ID 999? {noEncontrado}");

// Mostrar productos después de la actualización
Console.WriteLine("\nDespués de actualizar:");
lista.MostrarProductos();
}
}

```

```

Antes de actualizar:
Lista de productos:
ID: 101, Nombre: Mouse, Precio: $25,990 COP
ID: 102, Nombre: Teclado, Precio: $45,500 COP
ID: 103, Nombre: Monitor, Precio: $199,990 COP

¿Actualización exitosa para ID 102? True
¿Actualización exitosa para ID 999? False

Después de actualizar:
Lista de productos:
ID: 101, Nombre: Mouse, Precio: $25,990 COP
ID: 102, Nombre: Teclado, Precio: $49,900 COP
ID: 103, Nombre: Monitor, Precio: $199,990 COP

```

Ejercicio 10: Eliminar Productos por Nombre Crea un método `int EliminarPorNombre(string nombre)` que elimine todos los productos que coincidan con un nombre específico y devuelva la cantidad de productos eliminados.

Script

```

using System;

// Clase Producto con ID, nombre y precio
public class Producto
{
    public int Id;

```

```
public string Nombre;
public double Precio;

public Producto(int id, string nombre, double precio)
{
    Id = id;
    Nombre = nombre;
    Precio = precio;
}
}
```

// Nodo que almacena un Producto

```
public class NodoProducto
{
    public Producto Valor;
    public NodoProducto Siguiente;

    public NodoProducto(Producto valor)
    {
        Valor = valor;
        Siguiente = null;
    }
}
```

// Lista enlazada de productos

```
public class ListaProductos
{
    public NodoProducto Cabeza;

    public ListaProductos()
    {
        Cabeza = null;
    }
}
```

```
}
```

```
// Agrega un producto al final de la lista
```

```
public void AgregarProducto(Producto nuevoProducto)
```

```
{
```

```
    NodoProducto nuevoNodo = new NodoProducto(nuevoProducto);
```

```
    if (Cabeza == null)
```

```
    {
```

```
        Cabeza = nuevoNodo;
```

```
    }
```

```
    else
```

```
    {
```

```
        NodoProducto actual = Cabeza;
```

```
        while (actual.Siguiente != null)
```

```
        {
```

```
            actual = actual.Siguiente;
```

```
        }
```

```
        actual.Siguiente = nuevoNodo;
```

```
    }
```

```
}
```

```
// Elimina todos los productos que coincidan con el nombre y devuelve cuántos fueron  
eliminados
```

```
public int EliminarPorNombre(string nombre)
```

```
{
```

```
    int eliminados = 0;
```

```
// Eliminar coincidencias al inicio de la lista
```

```
while (Cabeza != null && Cabeza.Valor.Nombre == nombre)
```

```
{
```

```
    Cabeza = Cabeza.Siguiente;
```

```

        eliminados++;
    }

    // Eliminar coincidencias en el resto de la lista
    NodoProducto actual = Cabeza;
    while (actual != null && actual.Siguiente != null)
    {
        if (actual.Siguiente.Valor.Nombre == nombre)
        {
            actual.Siguiente = actual.Siguiente.Siguiente;
            eliminados++;
        }
        else
        {
            actual = actual.Siguiente;
        }
    }

    return eliminados;
}

// Muestra todos los productos
public void MostrarProductos()
{
    NodoProducto actual = Cabeza;
    Console.WriteLine("Lista de productos:");

    while (actual != null)
    {
        Console.WriteLine($"ID: {actual.Valor.Id}, Nombre: {actual.Valor.Nombre}, Precio:
        ${actual.Valor.Precio:N0} COP");
        actual = actual.Siguiente;
    }
}

```

```
    }  
  }  
}
```

// Programa principal

class Program

```
{
```

```
    static void Main()
```

```
    {
```

```
        ListaProductos lista = new ListaProductos();
```

```
        // Agregar productos (algunos con el mismo nombre)
```

```
        lista.AgregarProducto(new Producto(101, "Mouse", 25990));
```

```
        lista.AgregarProducto(new Producto(102, "Teclado", 45500));
```

```
        lista.AgregarProducto(new Producto(103, "Mouse", 26990));
```

```
        lista.AgregarProducto(new Producto(104, "Monitor", 199990));
```

```
        lista.AgregarProducto(new Producto(105, "Mouse", 27990));
```

```
        // Mostrar productos antes de eliminar
```

```
        Console.WriteLine("Antes de eliminar:");
```

```
        lista.MostrarProductos();
```

```
        // Eliminar productos por nombre
```

```
        int eliminados = lista.EliminarPorNombre("Mouse");
```

```
        Console.WriteLine($"Productos eliminados con nombre 'Mouse': {eliminados}");
```

```
        // Mostrar productos después de eliminar
```

```
        Console.WriteLine("\nDespués de eliminar:");
```

```
        lista.MostrarProductos();
```

```
    }
```

```
}
```

```

Antes de eliminar:
Lista de productos:
ID: 101, Nombre: Mouse, Precio: $25,990 COP
ID: 102, Nombre: Teclado, Precio: $45,500 COP
ID: 103, Nombre: Mouse, Precio: $26,990 COP
ID: 104, Nombre: Monitor, Precio: $199,990 COP
ID: 105, Nombre: Mouse, Precio: $27,990 COP

Productos eliminados con nombre 'Mouse': 3

Después de eliminar:
Lista de productos:
ID: 102, Nombre: Teclado, Precio: $45,500 COP
ID: 104, Nombre: Monitor, Precio: $199,990 COP

```

Ejercicio 11: Obtener Nombres de Productos Escribe un método `List<string> ObtenerNombres()` que recorra la lista y devuelva una `List<string>` con los nombres de todos los productos.

Script

```
using System;
```

```
using System.Collections.Generic;
```

```
// Clase Producto con ID, nombre y precio
```

```
public class Producto
```

```
{
```

```
    public int Id;
```

```
    public string Nombre;
```

```
    public double Precio;
```

```
    public Producto(int id, string nombre, double precio)
```

```
    {
```

```
        Id = id;
```

```
        Nombre = nombre;
```

```
        Precio = precio;
```

```
    }
```

```
}
```

```
// Nodo que almacena un Producto
```

```
public class NodoProducto
```

```
{
```

```

    public Producto Valor;
    public NodoProducto Siguiente;

    public NodoProducto(Producto valor)
    {
        Valor = valor;
        Siguiente = null;
    }
}

// Lista enlazada de productos
public class ListaProductos
{
    public NodoProducto Cabeza;

    public ListaProductos()
    {
        Cabeza = null;
    }

    // Agrega un producto al final de la lista
    public void AgregarProducto(Producto nuevoProducto)
    {
        NodoProducto nuevoNodo = new NodoProducto(nuevoProducto);

        if (Cabeza == null)
        {
            Cabeza = nuevoNodo;
        }
        else
        {
            NodoProducto actual = Cabeza;

```

```

        while (actual.Siguiente != null)
        {
            actual = actual.Siguiente;
        }
        actual.Siguiente = nuevoNodo;
    }
}

```

// Devuelve una lista con los nombres de todos los productos

```

public List<string> ObtenerNombres()
{
    List<string> nombres = new List<string>();
    NodoProducto actual = Cabeza;

    while (actual != null)
    {
        nombres.Add(actual.Valor.Nombre);
        actual = actual.Siguiente;
    }

    return nombres;
}

```

// Muestra todos los productos

```

public void MostrarProductos()
{
    NodoProducto actual = Cabeza;
    Console.WriteLine("Lista de productos:");

    while (actual != null)
    {
        Console.WriteLine($"ID: {actual.Valor.Id}, Nombre: {actual.Valor.Nombre}, Precio:

```



```

    ${actual.Valor.Precio:N0} COP");
        actual = actual.Siguiente;
    }
}
}

```

// Programa principal

```
class Program
```

```
{
```

```
    static void Main()
```

```
    {
```

```
        ListaProductos lista = new ListaProductos();
```

```
        // Agregar productos
```

```
        lista.AgregarProducto(new Producto(101, "Mouse", 25990));
```

```
        lista.AgregarProducto(new Producto(102, "Teclado", 45500));
```

```
        lista.AgregarProducto(new Producto(103, "Monitor", 199990));
```

```
        // Mostrar productos
```

```
        lista.MostrarProductos();
```

```
        // Obtener y mostrar nombres
```

```
        List<string> nombres = lista.ObtenerNombres();
```

```
        Console.WriteLine("\nNombres de productos:");
```

```
        foreach (string nombre in nombres)
```

```
        {
```

```
            Console.WriteLine("- " + nombre);
```

```
        }
```

```
    }
```

```
}
```

```
Lista de productos:
ID: 101, Nombre: Mouse, Precio: $25,990 COP
ID: 102, Nombre: Teclado, Precio: $45,500 COP
ID: 103, Nombre: Monitor, Precio: $199,990 COP

Nombres de productos:
- Mouse
- Teclado
- Monitor
```

Ejercicio 12: Calcular Valor Total del Inventario Implementa un método double CalcularValorTotal() que calcule la suma de los precios de todos los productos en la lista.

Script

```
using System;
```

```
// Clase Producto con ID, nombre y precio
```

```
public class Producto
```

```
{
```

```
    public int Id;
```

```
    public string Nombre;
```

```
    public double Precio;
```

```
    public Producto(int id, string nombre, double precio)
```

```
    {
```

```
        Id = id;
```

```
        Nombre = nombre;
```

```
        Precio = precio;
```

```
    }
```

```
}
```

```
// Nodo que almacena un Producto
```

```
public class NodoProducto
```

```
{
```

```
    public Producto Valor;
```

```
    public NodoProducto Siguiente;
```

```
public NodoProducto(Producto valor)
{
    Valor = valor;
    Siguiente = null;
}
}
```

// Lista enlazada de productos

```
public class ListaProductos
```

```
{
    public NodoProducto Cabeza;
```

```
    public ListaProductos()
```

```
{
    Cabeza = null;
}
```

// Agrega un producto al final de la lista

```
public void AgregarProducto(Producto nuevoProducto)
```

```
{
    NodoProducto nuevoNodo = new NodoProducto(nuevoProducto);
```

```
    if (Cabeza == null)
```

```
{
    Cabeza = nuevoNodo;
```

```
}
```

```
else
```

```
{
```

```
    NodoProducto actual = Cabeza;
```

```
    while (actual.Siguiente != null)
```

```
{
```

```

        actual = actual.Siguiente;
    }
    actual.Siguiente = nuevoNodo;
}
}

```

// Calcula el valor total del inventario sumando los precios de todos los productos

```

public double CalcularValorTotal()
{
    double total = 0;
    NodoProducto actual = Cabeza;

    while (actual != null)
    {
        total += actual.Valor.Precio;
        actual = actual.Siguiente;
    }

    return total;
}

```

// Muestra todos los productos

```

public void MostrarProductos()
{
    NodoProducto actual = Cabeza;
    Console.WriteLine("Lista de productos:");

    while (actual != null)
    {
        Console.WriteLine($"ID: {actual.Valor.Id}, Nombre: {actual.Valor.Nombre}, Precio:
        ${actual.Valor.Precio:N0} COP");
        actual = actual.Siguiente;
    }
}

```

```

    }
}

// Programa principal
class Program
{
    static void Main()
    {
        ListaProductos lista = new ListaProductos();

        // Agregar productos
        lista.AgregarProducto(new Producto(101, "Mouse", 25990));
        lista.AgregarProducto(new Producto(102, "Teclado", 45500));
        lista.AgregarProducto(new Producto(103, "Monitor", 199990));

        // Mostrar productos
        lista.MostrarProductos();

        // Calcular y mostrar el valor total del inventario
        double totalInventario = lista.CalcularValorTotal();
        Console.WriteLine($"Valor total del inventario: ${totalInventario:N0} COP");
    }
}

```

```

Lista de productos:
ID: 101, Nombre: Mouse, Precio: $25,990 COP
ID: 102, Nombre: Teclado, Precio: $45,500 COP
ID: 103, Nombre: Monitor, Precio: $199,990 COP
Valor total del inventario: $271,480 COP

```

Ejercicio 13: Implementar Inserción al Inicio Dentro de la clase ListaEnlazadaArray, implementa el método void InsertarAlInicio(int valor). Deberás encontrar un índice libre en el arreglo memoria para el nuevo nodo.

Script

```
using System;
```

```
public class Nodo
```

```
{
```

```
    public int Valor;
```

```
    public int Siguiente; // Índice del siguiente nodo en el arreglo
```

```
    public Nodo(int valor)
```

```
    {
```

```
        Valor = valor;
```

```
        Siguiente = -1; // -1 indica que no hay siguiente nodo
```

```
    }
```

```
}
```

```
public class ListaEnlazadaArray
```

```
{
```

```
    private Nodo[] memoria; // Arreglo que simula la memoria de nodos
```

```
    private int inicio;      // Índice del primer nodo en la lista
```

```
    public ListaEnlazadaArray(int tamaño)
```

```
    {
```

```
        memoria = new Nodo[tamaño];
```

```
        inicio = -1; // Lista vacía
```

```
    }
```

```
    // Método para insertar un valor al inicio de la lista
```

```
    public void InsertarAlInicio(int valor)
```

```
    {
```

```
        int indiceLibre = BuscarIndiceLibre();
```

```
        if (indiceLibre == -1)
```

```

{
    Console.WriteLine("No hay espacio disponible en la memoria.");
    return;
}

// Crear el nuevo nodo en el índice libre
memoria[indiceLibre] = new Nodo(valor);

// Enlazar el nuevo nodo al inicio actual
memoria[indiceLibre].Siguiente = inicio;

// Actualizar el inicio de la lista
inicio = indiceLibre;
}

// Busca el primer índice libre en el arreglo memoria
private int BuscarIndiceLibre()
{
    for (int i = 0; i < memoria.Length; i++)
    {
        if (memoria[i] == null)
        {
            return i;
        }
    }
    return -1; // No hay espacio disponible
}

// Muestra los elementos de la lista
public void MostrarLista()
{
    int actual = inicio;

```

```

        Console.WriteLine("Lista enlazada:");

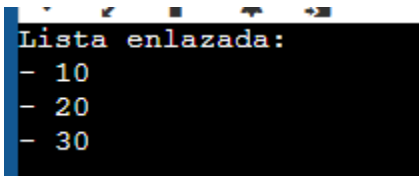
        while (actual != -1)
        {
            Console.WriteLine($"- {memoria[actual].Valor}");
            actual = memoria[actual].Siguiente;
        }
    }
}

// Programa principal
class Program
{
    static void Main()
    {
        ListaEnlazadaArray lista = new ListaEnlazadaArray(10);

        lista.InsertarAllInicio(30);
        lista.InsertarAllInicio(20);
        lista.InsertarAllInicio(10);

        lista.MostrarLista();
    }
}

```



```

Lista enlazada:
- 10
- 20
- 30

```

Ejercicio 14: Implementar Eliminación por Valor Escribe un método `bool EliminarPorValor(int valor)` que busque un nodo con el valor dado, lo elimine y marque su espacio en el arreglo como "libre" para ser reutilizado.

Script


```

using System;

public class Nodo
{
    public int Valor;
    public int Siguiente; // Índice del siguiente nodo en el arreglo

    public Nodo(int valor)
    {
        Valor = valor;
        Siguiente = -1; // -1 indica que no hay siguiente nodo
    }
}

public class ListaEnlazadaArray
{
    private Nodo[] memoria; // Arreglo que simula la memoria
    private int inicio;     // Índice del primer nodo

    public ListaEnlazadaArray(int tamaño)
    {
        memoria = new Nodo[tamaño];
        inicio = -1; // Lista vacía
    }

    // Inserta un nuevo nodo al inicio
    public void InsertarAlInicio(int valor)
    {
        int libre = BuscarIndiceLibre();
        if (libre == -1)
        {
            Console.WriteLine("Memoria llena.");
            return;
        }

        memoria[libre] = new Nodo(valor);
        memoria[libre].Siguiente = inicio;
        inicio = libre;
    }
}

```

```
}
```

```
// Elimina el primer nodo que contenga el valor dado
```

```
public bool EliminarPorValor(int valor)
```

```
{
```

```
    int actual = inicio;
```

```
    int anterior = -1;
```

```
    while (actual != -1)
```

```
    {
```

```
        if (memoria[actual].Valor == valor)
```

```
        {
```

```
            if (anterior == -1)
```

```
            {
```

```
                // El nodo a eliminar es el primero
```

```
                inicio = memoria[actual].Siguiente;
```

```
            }
```

```
        else
```

```
        {
```

```
            // Saltar el nodo actual
```

```
            memoria[anterior].Siguiente = memoria[actual].Siguiente;
```

```
        }
```

```
        // Marcar el nodo como libre
```

```
        memoria[actual] = null;
```

```
        return true;
```

```
    }
```

```
    anterior = actual;
```

```
    actual = memoria[actual].Siguiente;
```

```
}
```

```
return false; // No se encontró el valor
```

```
}
```

```
// Busca el primer índice libre en el arreglo
```

```
private int BuscarIndiceLibre()
```

```
{
```

```

        for (int i = 0; i < memoria.Length; i++)
        {
            if (memoria[i] == null)
                return i;
        }
        return -1;
    }

    // Muestra los elementos de la lista
    public void MostrarLista()
    {
        int actual = inicio;
        Console.WriteLine("Lista enlazada:");

        while (actual != -1)
        {
            Console.WriteLine($"- {memoria[actual].Valor}");
            actual = memoria[actual].Siguiente;
        }
    }
}

// Programa de prueba
class Program
{
    static void Main()
    {
        ListaEnlazadaArray lista = new ListaEnlazadaArray(10);

        lista.InsertarAllInicio(30);
        lista.InsertarAllInicio(20);
        lista.InsertarAllInicio(10);
        lista.InsertarAllInicio(40);

        Console.WriteLine("Antes de eliminar:");
        lista.MostrarLista();

        bool eliminado = lista.EliminarPorValor(20);
    }
}

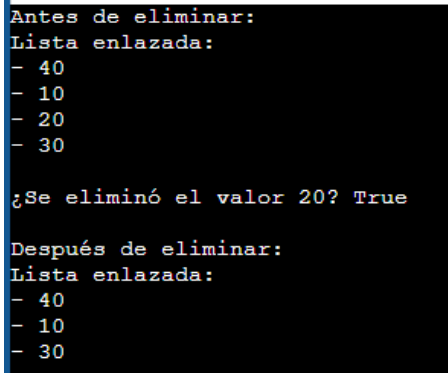
```

```

        Console.WriteLine($"¿Se eliminó el valor 20? {eliminado}");

        Console.WriteLine("\nDespués de eliminar:");
        lista.MostrarLista();
    }
}

```



```

Antes de eliminar:
Lista enlazada:
- 40
- 10
- 20
- 30

¿Se eliminó el valor 20? True

Después de eliminar:
Lista enlazada:
- 40
- 10
- 30

```

Ejercicio 15: Imprimir la Lista Crea un método void ImprimirLista() que recorra la lista siguiendo los SiguienteIndice desde la cabeza hasta llegar a -1.

Script

```

using System;

public class Nodo
{
    public int Valor;

    public int SiguienteIndice; // Índice del siguiente nodo en el arreglo

    public Nodo(int valor)
    {
        Valor = valor;

        SiguienteIndice = -1; // -1 indica fin de la lista
    }
}

```

```
}  
}
```

```
public class ListaEnlazadaArray
```

```
{  
    private Nodo[] memoria; // Arreglo que simula la memoria  
    private int inicio;     // Índice del primer nodo
```

```
  
    public ListaEnlazadaArray(int tamaño)  
    {  
        memoria = new Nodo[tamaño];  
        inicio = -1; // Lista vacía  
    }
```

```
  
    // Inserta un nuevo nodo al inicio  
    public void InsertarAlInicio(int valor)  
    {  
        int libre = BuscarIndiceLibre();  
        if (libre == -1)  
        {  
            Console.WriteLine("Memoria llena.");  
            return;  
        }
```

```
  
        memoria[libre] = new Nodo(valor);  
        memoria[libre].SiguienteIndice = inicio;  
        inicio = libre;  
    }
```

```
  
    // Imprime la lista recorriendo desde la cabeza hasta -1  
    public void ImprimirLista()
```

```

{
    int actual = inicio;

    Console.WriteLine("Contenido de la lista:");

    while (actual != -1)
    {
        Console.WriteLine($"- {memoria[actual].Valor}");
        actual = memoria[actual].SiguienteIndice;
    }
}

// Busca el primer índice libre en el arreglo
private int BuscarIndiceLibre()
{
    for (int i = 0; i < memoria.Length; i++)
    {
        if (memoria[i] == null)
            return i;
    }
    return -1;
}

}

// Programa principal
class Program
{
    static void Main()
    {
        ListaEnlazadaArray lista = new ListaEnlazadaArray(10);

        lista.InsertarAllInicio(30);
    }
}

```

```
        lista.InsertarAllInicio(20);  
        lista.InsertarAllInicio(10);  
  
        lista.ImprimirLista();  
    }  
}
```

```
Contenido de la lista:  
- 10  
- 20  
- 30
```

Ejercicio 16: Desfragmentar la Lista Escribe un método void Desfragmentar() que reorganice los nodos en el arreglo para que todos los nodos activos estén en posiciones contiguas al principio del arreglo, actualizando los índices correspondientes.

Script:

```
using System;
```

```
class Nodo{  
    public int valor;    //dato que guarda el nodo  
    public int siguiente; //índice del siguiente Nodo  
    public bool activo;  //indica si el nodo está libre o en uso  
  
    //Constructor que inicializa los valores del nodo  
    public Nodo(int valor, int siguiente, bool activo){  
        this.valor = valor;  
        this.siguiente = siguiente;  
        this.activo = activo;  
    }  
}
```

```
class ListaEstatica  
{  
    Nodo[] nodos; //Arreglo que almacena los nodos  
    int cabeza;  //índice del primero nodo activo  
  
    public ListaEstatica()  
    {  
        nodos = new Nodo[6];  
  
        // llenamos manualmente el arreglo con algunos nodos inactivos  
        nodos[0] = new Nodo(10, 1, true);  
        nodos[1] = new Nodo(20, 2, true);  
        nodos[2] = new Nodo(0, -1, false); // inactivo  
    }  
}
```



```
    nodos[3] = new Nodo(30, 4, true);
    nodos[4] = new Nodo(0, -1, false); // inactivo
    nodos[5] = new Nodo(40, -1, true);
    cabeza = 0;
}
```

//método Desfragmentar() que reorganiza los nodos activos al principio del arreglo

```
public void Desfragmentar()
{
    int destino = 0;
    // recorremos todos los nodos del arreglo
    for (int origen = 0; origen < nodos.Length; origen++)
    {
        // si el nodo está activo lo mueve hacia adelante
        if (nodos[origen].activo)
        {
            //solo lo movemos si origen y destino son diferentes
            if (origen != destino)
            {
                //copiamos el valor del nodo activo al destino
                nodos[destino].valor = nodos[origen].valor;
                nodos[destino].activo = true;
            }
            destino++;
        }
    }
}
```

// Desactiva los sobrantes

```
for (int i = destino; i < nodos.Length; i++)
{
    nodos[i].activo = false;
}
```

```

        Console.WriteLine("Lista desfragmentada con exito.");
    }

    public void Mostrar()
    {
        for (int i = 0; i < nodos.Length; i++)
        {
            Console.WriteLine($"Pos {i}: Valor={nodos[i].valor}, Activo={nodos[i].activo}");
        }
    }
}

// programa principal para ejecutar y probar la lista
class Program
{
    static void Main()
    {
        ListaEstatica lista = new ListaEstatica();

        Console.WriteLine("Antes de desfragmentar:");
        lista.Mostrar();

        lista.Desfragmentar();

        Console.WriteLine("\nDespues de desfragmentar:");
        lista.Mostrar();
    }
}

```

```
Antes de desfragmentar:
Pos 0: Valor=10, Activo=True
Pos 1: Valor=20, Activo=True
Pos 2: Valor=0, Activo=False
Pos 3: Valor=30, Activo=True
Pos 4: Valor=0, Activo=False
Pos 5: Valor=40, Activo=True
Lista desfragmentada con exito.
```

```
Despues de desfragmentar:
Pos 0: Valor=10, Activo=True
Pos 1: Valor=20, Activo=True
Pos 2: Valor=30, Activo=True
Pos 3: Valor=40, Activo=True
Pos 4: Valor=0, Activo=False
Pos 5: Valor=40, Activo=False
```

Ejercicio 17: Validar si es Circular Escribe un método bool EsCircular() que, dado un nodo cabeza de una lista (que podría ser simple o circular), determine si la lista es circular.

Script:

```
using System;
```

```
// Clase Nodo: representa un nodo de la lista enlazada
```

```
class Nodo
```

```
{
```

```
    public int valor;    // Dato almacenado en el nodo
```

```
    public Nodo siguiente; // Referencia al siguiente nodo
```

```
    // Constructor para crear un nodo nuevo
```

```
    public Nodo(int valor)
```

```
    {
```

```
        this.valor = valor;
```

```
        this.siguiente = null;
```

```
    }
```

```
}
```

```
// Clase Lista: maneja las operaciones de la lista enlazada
```

```
class Lista
```

```
{
```

```
public Nodo cabeza; // Referencia al primer nodo de la lista
```

```
// Constructor: inicia la lista vacía
```

```
public Lista()
```

```
{
```

```
    cabeza = null;
```

```
}
```

```
// Método para agregar un nodo al final (lista simple)
```

```
public void InsertarAlFinal(int valor)
```

```
{
```

```
    Nodo nuevo = new Nodo(valor); // Se crea el nuevo nodo
```

```
    if (cabeza == null)
```

```
    {
```

```
        // Si la lista está vacía, el nuevo nodo es la cabeza
```

```
        cabeza = nuevo;
```

```
    }
```

```
    else
```

```
    {
```

```
        // Si no, recorremos hasta el último nodo
```

```
        Nodo actual = cabeza;
```

```
        while (actual.siguiente != null)
```

```
        {
```

```
            actual = actual.siguiente;
```

```
        }
```

```
        // Enlazamos el nuevo nodo al final
```

```
        actual.siguiente = nuevo;
```

```
    }
```

```
}
```

```
// Método para convertir la lista en circular (solo para probar)
```

```

public void HacerCircular()
{
    if (cabeza == null)
        return;

    Nodo actual = cabeza;
    // Recorremos hasta el último nodo
    while (actual.siguiente != null)
    {
        actual = actual.siguiente;
    }
    // Hacemos que el último apunte de nuevo a la cabeza → lista circular
    actual.siguiente = cabeza;
}

```

// Método que determina si la lista es circular

```

public bool EsCircular()
{
    if (cabeza == null)
        return false; // Una lista vacía no es circular

```

Nodo lento = cabeza; // Avanza de uno en uno

Nodo rapido = cabeza; // Avanza de dos en dos

// Recorremos hasta que el puntero rápido o su siguiente sean nulos

```

while (rapido != null && rapido.siguiente != null)

```

```

{
    lento = lento.siguiente;          // Avanza 1 paso
    rapido = rapido.siguiente.siguiente; // Avanza 2 pasos

```

// Si ambos punteros se encuentran, la lista es circular

```

        if (lento == rapido)
            return true;
    }

    // Si salimos del bucle, no se encontraron → no es circular
    return false;
}

// Método para mostrar los nodos (solo para listas no circulares)
public void Mostrar()
{
    Nodo actual = cabeza;
    while (actual != null)
    {
        Console.Write(actual.valor + " -> ");
        actual = actual.siguiente;
    }
    Console.WriteLine("null");
}
}

// Programa principal
class Program
{
    static void Main()
    {
        // Creamos una lista simple
        Lista lista = new Lista();
        lista.InsertarAlFinal(10);
        lista.InsertarAlFinal(20);
        lista.InsertarAlFinal(30);
    }
}

```

```

        lista.InsertarAlFinal(40);

        Console.WriteLine("Lista creada:");
        lista.Mostrar();

        // Verificamos si es circular (debería ser falso)
        Console.WriteLine("¿La lista es circular? " + lista.EsCircular());

        // Ahora hacemos la lista circular
        lista.HacerCircular();

        // Verificamos nuevamente
        Console.WriteLine("¿La lista es circular después de modificarla? " +
        lista.EsCircular());
    }
}

```

```

Lista creada:
10 -> 20 -> 30 -> 40 -> null
¿La lista es circular? False
¿La lista es circular después de modificarla? True

```

Ejercicio 18: Contar Nodos en Lista Circular Implementa un método `int ContarNodosCirculares()` para una lista circular. Ten cuidado de no entrar en un bucle infinito.

Script:

```

using System;

// Clase Nodo: representa cada elemento de la lista circular
class Nodo
{
    public int valor;    // Dato que guarda el nodo
    public Nodo siguiente; // Referencia al siguiente nodo
}

```

```

// Constructor: inicializa el valor y el puntero siguiente
public Nodo(int valor)
{
    this.valor = valor;
    this.siguiente = null;
}
}

// Clase ListaCircular: contiene las operaciones sobre una lista circular
class ListaCircular
{
    public Nodo cabeza; // Puntero al primer nodo

    // Constructor: comienza la lista vacía
    public ListaCircular()
    {
        cabeza = null;
    }

    // Método para insertar un nodo al final de la lista circular
    public void Insertar(int valor)
    {
        Nodo nuevo = new Nodo(valor); // Se crea un nuevo nodo

        if (cabeza == null)
        {
            // Si la lista está vacía, el nuevo nodo apunta a sí mismo
            cabeza = nuevo;
            cabeza.siguiente = cabeza;
        }
        else
        {

```



```

// Recorremos hasta el último nodo (el que apunta a la cabeza)
Nodo actual = cabeza;
while (actual.siguiente != cabeza)
{
    actual = actual.siguiente;
}

// Enlazamos el nuevo nodo al final
actual.siguiente = nuevo;
nuevo.siguiente = cabeza; // El nuevo último apunta a la cabeza
}
}

```

```

// Método para contar los nodos de la lista circular
public int ContarNodosCirculares()
{
    // Si la lista está vacía, no hay nodos
    if (cabeza == null)
        return 0;

    int contador = 1; // Empezamos en 1 porque ya contamos la cabeza
    Nodo actual = cabeza.siguiente; // Iniciamos desde el siguiente

    // Recorremos mientras no volvamos a la cabeza
    while (actual != cabeza)
    {
        contador++; // Contamos cada nodo
        actual = actual.siguiente; // Avanzamos al siguiente nodo
    }

    return contador; // Retornamos el total
}

```

```

// Método para mostrar los elementos de la lista circular
public void Mostrar()
{
    if (cabeza == null)
    {
        Console.WriteLine("La lista está vacía.");
        return;
    }

    Nodo actual = cabeza;
    Console.Write("Lista circular: ");
    do
    {
        Console.Write(actual.valor + " -> ");
        actual = actual.siguiente;
    }
    while (actual != cabeza); // Se detiene al volver al inicio
    Console.WriteLine("(vuelve a la cabeza)");
}

}

// Programa principal
class Program
{
    static void Main()
    {
        // Creamos una lista circular
        ListaCircular lista = new ListaCircular();

        // Insertamos algunos valores
        lista.Insertar(10);

```

```

        lista.Insertar(20);

        lista.Insertar(30);

        lista.Insertar(40);


        // Mostramos la lista circular

        lista.Mostrar();


        // Contamos los nodos de la lista circular

        int total = lista.ContarNodosCirculares();

        Console.WriteLine("Número de nodos en la lista circular: " + total);
    }
}

```

```

Lista circular: 10 -> 20 -> 30 -> 40 -> (vuelve a la cabeza)
Número de nodos en la lista circular: 4

```

Ejercicio 19: Insertar al Final de una Lista Circular Crea un método void InsertarAlFinalCircular(int valor) que agregue un nuevo nodo justo antes de la cabeza, convirtiéndose en el nuevo "último" nodo.

Script:

```

using System;


// Clase Nodo: representa un nodo dentro de la lista circular
class Nodo
{
    public int valor;    // Dato almacenado en el nodo

    public Nodo siguiente; // Referencia al siguiente nodo en la lista


    // Constructor para inicializar el nodo con un valor
    public Nodo(int valor)
    {
        this.valor = valor;

        this.siguiente = null;
    }
}

```

```
}  
}
```

// Clase ListaCircular: contiene los métodos para manipular la lista circular

```
class ListaCircular
```

```
{
```

```
    public Nodo cabeza; // Referencia al primer nodo (cabeza)
```

// Constructor: inicia una lista vacía

```
    public ListaCircular()
```

```
    {
```

```
        cabeza = null;
```

```
    }
```

// Método para insertar un nuevo nodo al final de la lista circular

```
    public void InsertarAlFinalCircular(int valor)
```

```
    {
```

```
        Nodo nuevo = new Nodo(valor); // Se crea el nuevo nodo
```

// Caso 1: si la lista está vacía, el nuevo nodo apunta a sí mismo

```
        if (cabeza == null)
```

```
        {
```

```
            cabeza = nuevo;
```

```
            cabeza.siguiete = cabeza; // El nodo se conecta consigo mismo
```

```
            Console.WriteLine("Se insertó el primer nodo (lista creada).");
```

```
            return;
```

```
        }
```

// Caso 2: si la lista ya tiene elementos

```
        Nodo actual = cabeza;
```

```

// Recorremos hasta el último nodo (el que apunta a la cabeza)
while (actual.siguiiente != cabeza)
{
    actual = actual.siguiiente;
}

// Enlazamos el nuevo nodo al final de la lista
actual.siguiiente = nuevo; // El último apunta al nuevo nodo
nuevo.siguiiente = cabeza; // El nuevo nodo apunta a la cabeza

Console.WriteLine("Nodo insertado al final de la lista circular.");
}

// Método para mostrar los elementos de la lista circular
public void Mostrar()
{
    if (cabeza == null)
    {
        Console.WriteLine("La lista está vacía.");
        return;
    }

    Nodo actual = cabeza;
    Console.Write("Lista circular: ");

    // Bucle do-while, porque siempre hay al menos un nodo
    do
    {
        Console.Write(actual.valor + " -> ");
        actual = actual.siguiiente;
    }

```

```

        while (actual != cabeza); // Se detiene al regresar a la cabeza

        Console.WriteLine("(vuelve a la cabeza)");
    }
}

// Programa principal
class Program
{
    static void Main()
    {
        // Creamos una lista circular vacía
        ListaCircular lista = new ListaCircular();

        // Insertamos varios nodos al final
        lista.InsertarAlFinalCircular(10);
        lista.InsertarAlFinalCircular(20);
        lista.InsertarAlFinalCircular(30);
        lista.InsertarAlFinalCircular(40);

        // Mostramos el resultado final
        lista.Mostrar();
    }
}

```

```

Se insertó el primer nodo (lista creada).
Nodo insertado al final de la lista circular.
Nodo insertado al final de la lista circular.
Nodo insertado al final de la lista circular.
Lista circular: 10 -> 20 -> 30 -> 40 -> (vuelve a la cabeza)

```

Ejercicio 20: Eliminar la Cabeza de una Lista Circular Escribe un método void EliminarCabezaCircular() que elimine el nodo cabeza y ajuste los punteros para que el segundo nodo se convierta en la nueva cabeza.

Script:

```
using System;
```

```
// Clase Nodo: representa un nodo en la lista circular
```

```
class Nodo
```

```
{
```

```
    public int valor;    // Valor almacenado en el nodo
```

```
    public Nodo siguiente; // Referencia al siguiente nodo
```

```
    // Constructor que inicializa el nodo con un valor
```

```
    public Nodo(int valor)
```

```
    {
```

```
        this.valor = valor;
```

```
        this.siguiente = null;
```

```
    }
```

```
}
```

```
// Clase ListaCircular: contiene las operaciones de la lista
```

```
class ListaCircular
```

```
{
```

```
    public Nodo cabeza; // Referencia al primer nodo de la lista
```

```
    // Constructor: inicializa una lista vacía
```

```
    public ListaCircular()
```

```
    {
```

```
        cabeza = null;
```

```
    }
```

```
    // Método para insertar un nodo al final de la lista circular
```

```

public void InsertarAlFinalCircular(int valor)
{
    Nodo nuevo = new Nodo(valor);

    // Caso 1: la lista está vacía → el nodo apunta a sí mismo
    if (cabeza == null)
    {
        cabeza = nuevo;
        cabeza.siguiete = cabeza;
        return;
    }

    // Caso 2: recorrer hasta el último nodo (el que apunta a la cabeza)
    Nodo actual = cabeza;
    while (actual.siguiete != cabeza)
    {
        actual = actual.siguiete;
    }

    // Enlazamos el nuevo nodo
    actual.siguiete = nuevo;
    nuevo.siguiete = cabeza;
}

// Método para eliminar la cabeza de la lista circular
public void EliminarCabezaCircular()
{
    // Caso 1: si la lista está vacía, no hay nada que eliminar
    if (cabeza == null)
    {
        Console.WriteLine("La lista está vacía. No hay nada que eliminar.");
        return;
    }

```



```
}
```

```
// Caso 2: si la lista tiene un solo nodo
```

```
if (cabeza.siguiete == cabeza)
```

```
{
```

```
    Console.WriteLine("Se eliminó el único nodo de la lista.");
```

```
    cabeza = null; // La lista queda vacía
```

```
    return;
```

```
}
```

```
// Caso 3: lista con varios nodos
```

```
Nodo actual = cabeza;
```

```
// Recorremos hasta el último nodo (el que apunta a la cabeza)
```

```
while (actual.siguiete != cabeza)
```

```
{
```

```
    actual = actual.siguiete;
```

```
}
```

```
// Ahora 'actual' apunta al último nodo
```

```
// Saltamos la cabeza → el segundo nodo será la nueva cabeza
```

```
actual.siguiete = cabeza.siguiete;
```

```
Console.WriteLine($"Se eliminó el nodo con valor {cabeza.valor}.");
```

```
// Movemos la referencia de la cabeza al siguiente nodo
```

```
cabeza = cabeza.siguiete;
```

```
}
```

```
// Método para mostrar el contenido de la lista circular
```

```
public void Mostrar()
```

```
{
```

```

        if (cabeza == null)
        {
            Console.WriteLine("La lista está vacía.");
            return;
        }

        Nodo actual = cabeza;
        Console.Write("Lista circular: ");
        do
        {
            Console.Write(actual.valor + " -> ");
            actual = actual.siguiente;
        }
        while (actual != cabeza);

        Console.WriteLine("(vuelve a la cabeza)");
    }
}

```

```

// Programa principal
class Program
{
    static void Main()
    {
        // Creamos una lista circular
        ListaCircular lista = new ListaCircular();

        // Insertamos algunos valores
        lista.InsertarAlFinalCircular(10);
        lista.InsertarAlFinalCircular(20);
        lista.InsertarAlFinalCircular(30);
        lista.InsertarAlFinalCircular(40);
    }
}

```

```

// Mostramos la lista original
Console.WriteLine("Lista original:");
lista.Mostrar();

// Eliminamos la cabeza
lista.EliminarCabezaCircular();

// Mostramos la lista después de eliminar la cabeza
Console.WriteLine("\nLista después de eliminar la cabeza:");
lista.Mostrar();

// Eliminamos nuevamente para ver el comportamiento
lista.EliminarCabezaCircular();

Console.WriteLine("\nLista después de eliminar otra vez la cabeza:");
lista.Mostrar();
}
}

```

```

Lista original:
Lista circular: 10 -> 20 -> 30 -> 40 -> (vuelve a la cabeza)
Se eliminó el nodo con valor 10.

Lista después de eliminar la cabeza:
Lista circular: 20 -> 30 -> 40 -> (vuelve a la cabeza)
Se eliminó el nodo con valor 20.

Lista después de eliminar otra vez la cabeza:
Lista circular: 30 -> 40 -> (vuelve a la cabeza)

```

Ejercicio 21: Rotar la Lista Circular Implementa un método void Rotar(int pasos) que mueva la cabeza pasos posiciones hacia adelante en la lista. Por ejemplo, si la lista es 1 -> 2 -> 3 y se rota 1 paso, la nueva cabeza será 2 y la lista será 2 -> 3 -> 1.

Script:

```
using System;
```

```
// Clase Nodo: representa un nodo de la lista circular
class Nodo
{
    public int valor;    // Dato almacenado en el nodo
    public Nodo siguiente; // Puntero al siguiente nodo

    public Nodo(int valor)
    {
        this.valor = valor;
        this.siguiente = null;
    }
}
```

```
// Clase ListaCircular con operaciones básicas
class ListaCircular
{
    public Nodo cabeza; // Referencia al primer nodo

    public ListaCircular()
    {
        cabeza = null;
    }
}
```

```
// Método para insertar un nodo al final de la lista circular
public void InsertarAlFinalCircular(int valor)
{
    Nodo nuevo = new Nodo(valor);

    // Caso 1: lista vacía → el nodo se apunta a sí mismo
    if (cabeza == null)
    {

```

```
    cabeza = nuevo;
    cabeza.siguiete = cabeza;
    return;
}
```

// Caso 2: lista con nodos → recorrer hasta el último

```
Nodo actual = cabeza;
while (actual.siguiete != cabeza)
{
    actual = actual.siguiete;
}
```

// Enlazar nuevo nodo al final

```
actual.siguiete = nuevo;
nuevo.siguiete = cabeza;
}
```

// Método para rotar la lista circular hacia adelante 'pasos' veces

public void Rotar(int pasos)

```
{
    // Si la lista está vacía o solo tiene un nodo, no se hace nada
    if (cabeza == null || cabeza.siguiete == cabeza)
    {
        Console.WriteLine("No se puede rotar: lista vacía o con un solo nodo.");
        return;
    }
}
```

// Si pasos es 0, no hay rotación

```
if (pasos == 0)
{
    Console.WriteLine("Rotación de 0 pasos. La lista queda igual.");
    return;
}
```

```

    }

    // Movemos la cabeza 'pasos' veces hacia adelante
    for (int num = 0; num < pasos; num++)
    {
        cabeza = cabeza.siguiete; // La nueva cabeza será el siguiente nodo
    }

    Console.WriteLine($"La lista se ha rotado {pasos} paso(s).");
}

// Método para mostrar la lista circular
public void Mostrar()
{
    if (cabeza == null)
    {
        Console.WriteLine("La lista está vacía.");
        return;
    }

    Nodo actual = cabeza;
    Console.Write("Lista circular: ");
    do
    {
        Console.Write(actual.valor + " -> ");
        actual = actual.siguiete;
    } while (actual != cabeza);

    Console.WriteLine("(vuelve a la cabeza)");
}
}

```

```
// Programa principal
class Program
{
    static void Main()
    {
        // Crear una lista circular
        ListaCircular lista = new ListaCircular();

        // Insertar algunos valores
        lista.InsertarAlFinalCircular(1);
        lista.InsertarAlFinalCircular(2);
        lista.InsertarAlFinalCircular(3);
        lista.InsertarAlFinalCircular(4);
        lista.InsertarAlFinalCircular(5);

        // Mostrar lista original
        Console.WriteLine("Lista original:");
        lista.Mostrar();

        // Rotar la lista 1 paso
        lista.Rotar(1);
        Console.WriteLine("\nDespués de rotar 1 paso:");
        lista.Mostrar();

        // Rotar la lista 3 pasos
        lista.Rotar(3);
        Console.WriteLine("\nDespués de rotar 3 pasos:");
        lista.Mostrar();
    }
}
```

```
Lista original:
Lista circular: 1 -> 2 -> 3 -> 4 -> 5 -> (vuelve a la cabeza)
La lista se ha rotado 1 paso(s).

Después de rotar 1 paso:
Lista circular: 2 -> 3 -> 4 -> 5 -> 1 -> (vuelve a la cabeza)
La lista se ha rotado 3 paso(s).

Después de rotar 3 pasos:
Lista circular: 5 -> 1 -> 2 -> 3 -> 4 -> (vuelve a la cabeza)
```

Ejercicio 22: Invertir una Lista Enlazada Escribe un método void Invertir() que invierta el orden de los nodos en una lista enlazada simple sin crear una nueva lista. (Pista: necesitarás cambiar la dirección de los punteros Siguiente).

Script:

```
using System;
```

```
// Clase Nodo: representa cada elemento de la lista enlazada
```

```
class Nodo
```

```
{
    public int valor;      // Dato del nodo
    public Nodo siguiente; // Puntero al siguiente nodo

    public Nodo(int valor)
    {
        this.valor = valor;
        this.siguiente = null;
    }
}
```

```
// Clase ListaEnlazada: contiene los métodos de manipulación
```

```
class ListaEnlazada
```

```
{
    public Nodo cabeza; // Referencia al primer nodo
```



```
public ListaEnlazada()
```

```
{
```

```
    cabeza = null;
```

```
}
```

```
// Método para insertar un nodo al final de la lista
```

```
public void InsertarAlFinal(int valor)
```

```
{
```

```
    Nodo nuevo = new Nodo(valor);
```

```
    // Caso 1: si la lista está vacía
```

```
    if (cabeza == null)
```

```
    {
```

```
        cabeza = nuevo;
```

```
        return;
```

```
    }
```

```
    // Caso 2: recorrer hasta el último nodo
```

```
    Nodo actual = cabeza;
```

```
    while (actual.siguiente != null)
```

```
    {
```

```
        actual = actual.siguiente;
```

```
    }
```

```
    // Enlazar el nuevo nodo al final
```

```
    actual.siguiente = nuevo;
```

```
}
```

```
// Método para invertir la lista enlazada sin crear una nueva
```

```
public void Invertir()
```

```
{
```

```

    Nodo anterior = null;    // Apuntador al nodo previo
    Nodo actual = cabeza;    // Apuntador al nodo actual
    Nodo siguiente = null;    // Apuntador temporal para guardar el siguiente nodo

    // Recorremos la lista y vamos invirtiendo los enlaces
    while (actual != null)
    {
        siguiente = actual.siguiente; // Guardamos el siguiente nodo
        actual.siguiente = anterior;  // Invertimos el puntero
        anterior = actual;            // Avanzamos el puntero anterior
        actual = siguiente;           // Avanzamos al siguiente nodo
    }

    // Al final, el último nodo recorrido se convierte en la nueva cabeza
    cabeza = anterior;

    Console.WriteLine("La lista ha sido invertida correctamente.");
}

// Método para mostrar los valores de la lista
public void Mostrar()
{
    if (cabeza == null)
    {
        Console.WriteLine("La lista está vacía.");
        return;
    }

    Nodo actual = cabeza;
    Console.Write("Lista: ");
    while (actual != null)

```

```

        {
            Console.Write(actual.valor + " -> ");
            actual = actual.siguiente;
        }
        Console.WriteLine("null");
    }
}

```

// Programa principal

class Program

```

{
    static void Main()
    {
        ListaEnlazada lista = new ListaEnlazada();

        // Insertamos algunos valores
        lista.InsertarAlFinal(10);
        lista.InsertarAlFinal(20);
        lista.InsertarAlFinal(30);
        lista.InsertarAlFinal(40);
        lista.InsertarAlFinal(50);

        Console.WriteLine("Lista original:");
        lista.Mostrar();

        // Invertimos la lista
        lista.Invertir();

        Console.WriteLine("\nLista invertida:");
        lista.Mostrar();
    }
}

```

```
}
```

```
Lista original:  
Lista: 10 -> 20 -> 30 -> 40 -> 50 -> null  
La lista ha sido invertida correctamente.  
  
Lista invertida:  
Lista: 50 -> 40 -> 30 -> 20 -> 10 -> null
```

Ejercicio 23: Insertar un Nodo en una Posición Específica Crea un método bool InsertarEnPosicion(int valor, int posicion) que inserte un nuevo nodo en un índice específico. Devuelve true si es exitoso, false si la posición está fuera de rango.

Script:

```
using System;
```

```
// Clase Nodo: representa un elemento de la lista
```

```
class Nodo
```

```
{
```

```
    public int valor;    // Dato del nodo
```

```
    public Nodo siguiente; // Referencia al siguiente nodo
```

```
    public Nodo(int valor)
```

```
    {
```

```
        this.valor = valor;
```

```
        this.siguiente = null;
```

```
    }
```

```
}
```

```
// Clase ListaEnlazada con operaciones básicas
```

```
class ListaEnlazada
```

```
{
```

```
    public Nodo cabeza; // Nodo inicial de la lista
```

```
    public ListaEnlazada()
```

```
    {
```

```

        cabeza = null;
    }

    // Método para insertar un nodo al final (de apoyo)
    public void InsertarAlFinal(int valor)
    {
        Nodo nuevo = new Nodo(valor);

        if (cabeza == null)
        {
            cabeza = nuevo;
            return;
        }

        Nodo actual = cabeza;
        while (actual.siguiente != null)
        {
            actual = actual.siguiente;
        }

        actual.siguiente = nuevo;
    }

    // Método para insertar un nodo en una posición específica
    public bool InsertarEnPosicion(int valor, int posicion)
    {
        // Si la posición es negativa, no es válida
        if (posicion < 0)
        {
            Console.WriteLine("Posición inválida. Debe ser mayor o igual a 0.");
            return false;
        }
    }

```

```
Nodo nuevo = new Nodo(valor);
```

```
// Caso 1: Insertar al inicio (posición 0)
```

```
if (posicion == 0)
```

```
{
```

```
    nuevo.siguiente = cabeza;
```

```
    cabeza = nuevo;
```

```
    Console.WriteLine($"Nodo {valor} insertado en posición {posicion} (al inicio).");
```

```
    return true;
```

```
}
```

```
// Recorremos hasta llegar al nodo anterior a la posición deseada
```

```
Nodo actual = cabeza;
```

```
int indice = 0;
```

```
while (actual != null && indice < posicion - 1)
```

```
{
```

```
    actual = actual.siguiente;
```

```
    indice++;
```

```
}
```

```
// Si llegamos al final sin alcanzar la posición, es inválida
```

```
if (actual == null)
```

```
{
```

```
    Console.WriteLine("Posición fuera de rango.");
```

```
    return false;
```

```
}
```

```
// Enlazamos el nuevo nodo en la posición indicada
```

```
nuevo.siguiente = actual.siguiente;
```

```
actual.siguiente = nuevo;
```

```

        Console.WriteLine($"Nodo {valor} insertado en posición {posicion} correctamente.");
        return true;
    }

    // Método para mostrar la lista
    public void Mostrar()
    {
        if (cabeza == null)
        {
            Console.WriteLine("La lista está vacía.");
            return;
        }

        Nodo actual = cabeza;
        Console.Write("Lista: ");
        while (actual != null)
        {
            Console.Write(actual.valor + " -> ");
            actual = actual.siguiente;
        }
        Console.WriteLine("null");
    }
}

// Programa principal
class Program
{
    static void Main()
    {
        ListaEnlazada lista = new ListaEnlazada();
    }
}

```

```

        // Insertamos algunos nodos al final
        lista.InsertarAlFinal(10);
        lista.InsertarAlFinal(20);
        lista.InsertarAlFinal(30);
        lista.InsertarAlFinal(40);

        Console.WriteLine("Lista original:");
        lista.Mostrar();

        // Insertar en distintas posiciones
        lista.InsertarEnPosicion(5, 0); // Al inicio
        lista.InsertarEnPosicion(25, 3); // En el medio
        lista.InsertarEnPosicion(50, 10); // Fuera de rango

        Console.WriteLine("\nLista después de las inserciones:");
        lista.Mostrar();
    }
}

```

```

Lista original:
Lista: 10 -> 20 -> 30 -> 40 -> null
Nodo 5 insertado en posición 0 (al inicio).
Nodo 25 insertado en posición 3 correctamente.
Posición fuera de rango.

Lista después de las inserciones:
Lista: 5 -> 10 -> 20 -> 25 -> 30 -> 40 -> null

```

Ejercicio 24: Eliminar un Nodo en una Posición Específica Implementa un método `bool EliminarDePosicion(int posicion)` que elimine el nodo en un índice dado.

Script:

```
using System;
```

```
// Clase Nodo: representa un nodo de la lista enlazada
```

```
class Nodo
```



```
{  
    public int valor;    // Dato del nodo  
    public Nodo siguiente; // Puntero al siguiente nodo  
  
    public Nodo(int valor)  
    {  
        this.valor = valor;  
        this.siguiente = null;  
    }  
}
```

// Clase ListaEnlazada: contiene los métodos de manipulación

```
class ListaEnlazada
```

```
{  
    public Nodo cabeza; // Nodo inicial de la lista  
  
    public ListaEnlazada()  
    {  
        cabeza = null;  
    }  
}
```

// Método auxiliar para insertar al final (para crear la lista)

```
public void InsertarAlFinal(int valor)  
{  
    Nodo nuevo = new Nodo(valor);  
  
    if (cabeza == null)  
    {  
        cabeza = nuevo;  
        return;  
    }  
}
```

```

    Nodo actual = cabeza;
    while (actual.siguiente != null)
    {
        actual = actual.siguiente;
    }

    actual.siguiente = nuevo;
}

```

// Método para eliminar un nodo en una posición específica

```

public bool EliminarDePosicion(int posicion)
{
    // Validar posición negativa
    if (posicion < 0)
    {
        Console.WriteLine("Posición inválida. Debe ser mayor o igual a 0.");
        return false;
    }
}

```

// Caso 1: lista vacía

```

if (cabeza == null)
{
    Console.WriteLine("No se puede eliminar: la lista está vacía.");
    return false;
}

```

// Caso 2: eliminar el primer nodo (posición 0)

```

if (posicion == 0)
{

```

```

    Console.WriteLine($"Nodo con valor {cabeza.valor} eliminado en posición 0
(inicio).");
}

```

```

        cabeza = cabeza.siguiete; // La cabeza ahora apunta al segundo nodo
        return true;
    }

    // Caso 3: eliminar nodo en otra posición
    Nodo actual = cabeza;
    int indice = 0;

    // Recorremos hasta el nodo anterior al que queremos eliminar
    while (actual != null && indice < posicion - 1)
    {
        actual = actual.siguiete;
        indice++;
    }

    // Si el nodo siguiente no existe, la posición está fuera de rango
    if (actual == null || actual.siguiete == null)
    {
        Console.WriteLine("Posición fuera de rango.");
        return false;
    }

    // Guardamos el valor eliminado (solo para mostrar)
    int eliminado = actual.siguiete.valor;

    // Saltamos el nodo que queremos eliminar
    actual.siguiete = actual.siguiete.siguiete;

    Console.WriteLine($"Nodo con valor {eliminado} eliminado en posición {posicion}.");
    return true;
}

```

```

// Método para mostrar los valores de la lista
public void Mostrar()
{
    if (cabeza == null)
    {
        Console.WriteLine("La lista está vacía.");
        return;
    }

    Nodo actual = cabeza;
    Console.Write("Lista: ");
    while (actual != null)
    {
        Console.Write(actual.valor + " -> ");
        actual = actual.siguiente;
    }
    Console.WriteLine("null");
}

}

// Programa principal
class Program
{
    static void Main()
    {
        ListaEnlazada lista = new ListaEnlazada();

        // Insertamos algunos valores
        lista.InsertarAlFinal(10);
        lista.InsertarAlFinal(20);
    }
}

```

```

        lista.InsertarAlFinal(30);
        lista.InsertarAlFinal(40);
        lista.InsertarAlFinal(50);

        Console.WriteLine("Lista original:");
        lista.Mostrar();

        // Eliminamos en distintas posiciones
        lista.EliminarDePosicion(0); // Elimina el primer nodo
        lista.EliminarDePosicion(2); // Elimina un nodo del medio
        lista.EliminarDePosicion(10); // Intento fuera de rango

        Console.WriteLine("\nLista después de las eliminaciones:");
        lista.Mostrar();
    }
}

```

```

Lista original:
Lista: 10 -> 20 -> 30 -> 40 -> 50 -> null
Nodo con valor 10 eliminado en posición 0 (inicio).
Nodo con valor 40 eliminado en posición 2.
Posición fuera de rango.

Lista después de las eliminaciones:
Lista: 20 -> 30 -> 50 -> null

```

Ejercicio 25: Encontrar el Nodo de En Medio Escribe un método `Nodo EncontrarNodoMedio()` que devuelva el nodo del medio de la lista. Si la lista tiene un número par de nodos, devuelve el segundo de los dos nodos centrales. (Pista: usa dos punteros, uno que se mueva el doble de rápido que el otro).

Script:

```
using System;
```

```
class Nodo
```

```
{
```

```

public int Valor;      // Dato almacenado en el nodo
public Nodo Siguiente; // Apuntador al siguiente nodo

public Nodo(int valor)
{
    Valor = valor;
    Siguiente = null;
}
}

class ListaEnlazada
{
    public Nodo cabeza; // Referencia al primer nodo de la lista

    // Agregar un nodo al final de la lista
    public void InsertarAlFinal(int valor)
    {
        Nodo nuevo = new Nodo(valor);

        if (cabeza == null)
        {
            cabeza = nuevo; // Si la lista está vacía, el nuevo nodo es la cabeza
        }
        else
        {
            Nodo actual = cabeza;
            while (actual.Siguiente != null)
            {
                actual = actual.Siguiente; // Avanzamos hasta el último nodo
            }
            actual.Siguiente = nuevo; // Agregamos el nuevo nodo al final
        }
    }
}

```

```
}
```

```
// Método que encuentra el nodo del medio
```

```
public Nodo EncontrarNodoMedio()
```

```
{
```

```
    if (cabeza == null)
```

```
    {
```

```
        Console.WriteLine("La lista está vacía.");
```

```
        return null;
```

```
    }
```

```
// Usamos dos punteros: lento y rápido
```

```
Nodo lento = cabeza;
```

```
Nodo rapido = cabeza;
```

```
// El puntero rápido avanza de dos en dos,
```

```
// mientras que el lento avanza de uno en uno.
```

```
// Cuando el rápido llegue al final, el lento estará en el medio.
```

```
while (rapido != null && rapido.Siguiente != null)
```

```
{
```

```
    lento = lento.Siguiente;          // Avanza de a uno
```

```
    rapido = rapido.Siguiente.Siguiente; // Avanza de a dos
```

```
}
```

```
return lento; // El nodo lento queda en el medio
```

```
}
```

```
// Mostrar todos los elementos de la lista
```

```
public void Mostrar()
```

```
{
```

```
    Nodo actual = cabeza;
```

```
    while (actual != null)
```

```

    {
        Console.Write(actual.Valor + " -> ");
        actual = actual.Siguiente;
    }
    Console.WriteLine("null");
}
}

```

class Program

```

{
    static void Main()
    {
        ListaEnlazada lista = new ListaEnlazada();

        // Insertamos algunos elementos
        lista.InsertarAlFinal(10);
        lista.InsertarAlFinal(20);
        lista.InsertarAlFinal(30);
        lista.InsertarAlFinal(40);
        lista.InsertarAlFinal(50);
        lista.InsertarAlFinal(60);

        Console.WriteLine("Lista actual:");
        lista.Mostrar();

        // Buscar el nodo del medio
        Nodo medio = lista.EncontrarNodoMedio();

        if (medio != null)
        {
            Console.WriteLine($"El nodo del medio tiene el valor: {medio.Valor}");
        }
    }
}

```



```
}  
}
```

```
Lista actual:  
10 -> 20 -> 30 -> 40 -> 50 -> 60 -> null  
El nodo del medio tiene el valor: 40
```

Ejercicio 26: Eliminar Duplicados de una Lista Ordenada Crea un método void EliminarDuplicadosOrdenados() que elimine todos los nodos con valores duplicados de una lista enlazada que ya está ordenada.

Scripts:

```
using System;
```

```
class Nodo
```

```
{
    public int Valor;      // Dato almacenado en el nodo
    public Nodo Siguiente; // Referencia al siguiente nodo

    public Nodo(int valor)
    {
        Valor = valor;
        Siguiente = null;
    }
}
```

```
class ListaEnlazada
```

```
{
    public Nodo cabeza; // Primer nodo de la lista

    // Método para insertar un nodo al final de la lista
    public void InsertarAlFinal(int valor)
    {
        Nodo nuevo = new Nodo(valor);

        if (cabeza == null)
        {
            cabeza = nuevo; // Si la lista está vacía, el nuevo nodo es la cabeza
        }
        else
        {
            Nodo actual = cabeza;
            while (actual.Siguiente != null)
            {
                actual = actual.Siguiente; // Recorremos hasta el último nodo
            }
        }
    }
}
```

```

        actual.Siguiente = nuevo; // Agregamos el nuevo nodo al final
    }
}

// Método que elimina los valores duplicados en una lista ordenada
public void EliminarDuplicadosOrdenados()
{
    if (cabeza == null)
    {
        Console.WriteLine("La lista está vacía.");
        return;
    }

    Nodo actual = cabeza;

    // Recorremos la lista mientras haya un siguiente nodo
    while (actual != null && actual.Siguiente != null)
    {
        // Si el valor actual y el siguiente son iguales, eliminamos el duplicado
        if (actual.Valor == actual.Siguiente.Valor)
        {
            actual.Siguiente = actual.Siguiente.Siguiente;
            // No avanzamos el puntero, porque puede haber más duplicados
        }
        else
        {
            actual = actual.Siguiente; // Avanzamos si no hay duplicado
        }
    }
}

// Método para mostrar la lista
public void Mostrar()
{
    Nodo actual = cabeza;
    while (actual != null)
    {
        Console.Write(actual.Valor + " -> ");
    }
}

```

```

        actual = actual.Siguiente;
    }
    Console.WriteLine("null");
}
}

class Program
{
    static void Main()
    {
        ListaEnlazada lista = new ListaEnlazada();

        // Insertamos valores (la lista debe estar ORDENADA)
        lista.InsertarAlFinal(10);
        lista.InsertarAlFinal(20);
        lista.InsertarAlFinal(20);
        lista.InsertarAlFinal(30);
        lista.InsertarAlFinal(30);
        lista.InsertarAlFinal(30);
        lista.InsertarAlFinal(40);
        lista.InsertarAlFinal(50);
        lista.InsertarAlFinal(50);

        Console.WriteLine("Lista original:");
        lista.Mostrar();

        // Eliminamos duplicados
        lista.EliminarDuplicadosOrdenados();

        Console.WriteLine("Lista después de eliminar duplicados:");
        lista.Mostrar();
    }
}

```

```

Lista original:
10 -> 20 -> 20 -> 30 -> 30 -> 30 -> 40 -> 50 -> 50 -> null
Lista después de eliminar duplicados:
10 -> 20 -> 30 -> 40 -> 50 -> null

```

Ejercicio 27: Fusionar dos Listas Ordenadas Implementa un método estático `Nodo FusionarListasOrdenadas(Nodo cabeza1, Nodo cabeza2)` que tome dos listas ordenadas y las fusione en una única lista ordenada.

script:

```
using System;
```

```
class Nodo
```

```
{
    public int Valor;      // Dato almacenado en el nodo
    public Nodo Siguiente; // Referencia al siguiente nodo

    public Nodo(int valor)
    {
        Valor = valor;
        Siguiente = null;
    }
}
```

```
class ListaEnlazada
```

```
{
    public Nodo cabeza; // Primer nodo de la lista

    // Método para insertar un nodo al final de la lista
    public void InsertarAlFinal(int valor)
    {
        Nodo nuevo = new Nodo(valor);

        if (cabeza == null)
        {
            cabeza = nuevo; // Si la lista está vacía, el nuevo nodo es la cabeza
        }
        else
```

```

{
    Nodo actual = cabeza;
    while (actual.Siguiente != null)
    {
        actual = actual.Siguiente; // Avanzamos hasta el último nodo
    }
    actual.Siguiente = nuevo; // Agregamos el nuevo nodo al final
}
}

```

// Método para mostrar los elementos de la lista

```

public void Mostrar()
{
    Nodo actual = cabeza;
    while (actual != null)
    {
        Console.Write(actual.Valor + " -> ");
        actual = actual.Siguiente;
    }
    Console.WriteLine("null");
}

```

// ♦ MÉTODO ESTÁTICO: Fusionar dos listas ordenadas

```

public static Nodo FusionarListasOrdenadas(Nodo cabeza1, Nodo cabeza2)
{
    // Si una de las listas está vacía, devolvemos la otra directamente
    if (cabeza1 == null) return cabeza2;
    if (cabeza2 == null) return cabeza1;

    // Creamos un nodo "ficticio" para simplificar la lógica de fusión
    Nodo dummy = new Nodo(0);

```

```

    Nodo actual = dummy;

    // Mientras ambas listas tengan elementos
    while (cabeza1 != null && cabeza2 != null)
    {
        if (cabeza1.Valor <= cabeza2.Valor)
        {
            actual.Siguiente = cabeza1; // Enlazamos el nodo menor
            cabeza1 = cabeza1.Siguiente; // Avanzamos en la primera lista
        }
        else
        {
            actual.Siguiente = cabeza2; // Enlazamos el nodo menor
            cabeza2 = cabeza2.Siguiente; // Avanzamos en la segunda lista
        }
        actual = actual.Siguiente; // Avanzamos en la lista resultante
    }

    // Si una lista aún tiene nodos, los unimos al final
    if (cabeza1 != null)
        actual.Siguiente = cabeza1;
    else
        actual.Siguiente = cabeza2;

    // El primer nodo real está después del dummy
    return dummy.Siguiente;
}

}

class Program
{

```

```

static void Main()
{
    // ♦ Creamos la primera lista ordenada
    ListaEnlazada lista1 = new ListaEnlazada();
    lista1.InsertarAlFinal(1);
    lista1.InsertarAlFinal(3);
    lista1.InsertarAlFinal(5);
    lista1.InsertarAlFinal(7);

    // ♦ Creamos la segunda lista ordenada
    ListaEnlazada lista2 = new ListaEnlazada();
    lista2.InsertarAlFinal(2);
    lista2.InsertarAlFinal(4);
    lista2.InsertarAlFinal(6);
    lista2.InsertarAlFinal(8);

    Console.WriteLine("Lista 1:");
    lista1.Mostrar();

    Console.WriteLine("Lista 2:");
    lista2.Mostrar();

    // ♦ Fusionamos ambas listas
    Nodo fusionada = ListaEnlazada.FusionarListasOrdenadas(lista1.cabeza,
lista2.cabeza);

    // ♦ Mostramos la lista resultante
    Console.WriteLine("Lista fusionada ordenada:");
    Nodo actual = fusionada;
    while (actual != null)
    {
        Console.Write(actual.Valor + " -> ");
    }
}

```



```

        actual = actual.Siguiente;
    }
    Console.WriteLine("null");
}
}

```

```

Lista 1:
1 -> 3 -> 5 -> 7 -> null
Lista 2:
2 -> 4 -> 6 -> 8 -> null
Lista fusionada ordenada:
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> null

```

Ejercicio 28: Detectar un Ciclo (no necesariamente circular) Escribe un método bool TieneCiclo() que determine si una lista enlazada contiene un ciclo (es decir, un nodo apunta a un nodo anterior en la secuencia, creando un bucle infinito). No asumas que es una lista circular completa.

Script:

```
using System;
```

```
class Nodo
```

```

{
    public int Valor;      // Dato almacenado en el nodo
    public Nodo Siguiente; // Referencia al siguiente nodo

    public Nodo(int valor)
    {
        Valor = valor;
        Siguiente = null;
    }
}

```

```
class ListaEnlazada
```

```

{
    public Nodo cabeza; // Primer nodo de la lista

```

// Método para insertar un nodo al final de la lista

public void InsertarAlFinal(int valor)

{

 Nodo nuevo = new Nodo(valor);

 if (cabeza == null)

 {

 cabeza = nuevo; // Si la lista está vacía, el nuevo nodo es la cabeza

 }

 else

 {

 Nodo actual = cabeza;

 while (actual.Siguiente != null)

 {

 actual = actual.Siguiente; // Recorremos hasta el último nodo

 }

 actual.Siguiente = nuevo; // Enlazamos el nuevo nodo al final

 }

}

// Método que detecta si la lista tiene un ciclo

public bool TieneCiclo()

{

 // Usamos el algoritmo de Floyd (puntero rápido y lento)

 Nodo lento = cabeza;

 Nodo rapido = cabeza;

 while (rapido != null && rapido.Siguiente != null)

 {

 lento = lento.Siguiente; // Avanza 1 paso

 rapido = rapido.Siguiente.Siguiente; // Avanza 2 pasos

```

        // Si se encuentran, hay un ciclo
        if (lento == rapido)
        {
            return true;
        }
    }

    // Si el puntero rápido llega al final, no hay ciclo
    return false;
}

// Método para mostrar los valores (solo para listas SIN ciclo)
public void Mostrar()
{
    Nodo actual = cabeza;
    while (actual != null)
    {
        Console.Write(actual.Valor + " -> ");
        actual = actual.Siguiente;
    }
    Console.WriteLine("null");
}

}

class Program
{
    static void Main()
    {
        ListaEnlazada lista = new ListaEnlazada();

        // Insertamos elementos

```

```

lista.InsertarAlFinal(10);
lista.InsertarAlFinal(20);
lista.InsertarAlFinal(30);
lista.InsertarAlFinal(40);
lista.InsertarAlFinal(50);

// Mostramos la lista (solo si no tiene ciclo)
Console.WriteLine("Lista enlazada:");
lista.Mostrar();

// Detectamos ciclo (en este punto no hay)
Console.WriteLine("¿La lista tiene ciclo? " + lista.TieneCiclo());

// Forzamos un ciclo manualmente (solo para probar)
// Hacemos que el último nodo apunte al nodo con valor 30
Nodo actual = lista.cabeza;
Nodo nodoConValor30 = null;
while (actual.Siguiente != null)
{
    if (actual.Valor == 30)
        nodoConValor30 = actual; // Guardamos el nodo con valor 30

    actual = actual.Siguiente; // Avanzamos
}
actual.Siguiente = nodoConValor30; // El último apunta al nodo 30 -> ciclo creado

// Volvemos a comprobar si hay ciclo
Console.WriteLine("¿La lista tiene ciclo ahora? " + lista.TieneCiclo());
}
}

```

```
Lista enlazada:  
10 -> 20 -> 30 -> 40 -> 50 -> null  
¿La lista tiene ciclo? False  
¿La lista tiene ciclo ahora? True
```

Ejercicio 29: Eliminar el N-ésimo Nodo desde el Final Crea un método void EliminarNesimoDesdeFinal(int n) que elimine el n-ésimo nodo contando desde el final de la lista.

Script:

```
using System;
```

```
class Nodo
```

```
{  
    public int Valor;      // Dato almacenado en el nodo  
    public Nodo Siguiente; // Referencia al siguiente nodo  
  
    public Nodo(int valor)  
    {  
        Valor = valor;  
        Siguiente = null;  
    }  
}
```

```
class ListaEnlazada
```

```
{  
    public Nodo cabeza; // Primer nodo de la lista  
  
    // Método para insertar un nodo al final de la lista  
    public void InsertarAlFinal(int valor)  
    {  
        Nodo nuevo = new Nodo(valor);  
  
        if (cabeza == null)  
        {
```

```

        cabeza = nuevo; // Si la lista está vacía, el nuevo nodo es la cabeza
    }
    else
    {
        Nodo actual = cabeza;
        while (actual.Siguiente != null)
        {
            actual = actual.Siguiente; // Recorremos hasta el último nodo
        }
        actual.Siguiente = nuevo; // Enlazamos el nuevo nodo al final
    }
}

```

// ♦ Método que elimina el n-ésimo nodo desde el final

```

public void EliminarNesimoDesdeFinal(int n)
{
    if (cabeza == null)
    {
        Console.WriteLine("La lista está vacía.");
        return;
    }
}

```

// Creamos un nodo auxiliar "ficticio" antes de la cabeza

```

Nodo dummy = new Nodo(0);
dummy.Siguiente = cabeza;

```

```

Nodo primero = dummy;

```

```

Nodo segundo = dummy;

```

// Avanzamos el primer puntero n+1 veces

```

for (int i = 0; i <= n; i++)

```

```

{
    // Si llegamos al final antes de completar los pasos, n es mayor que la longitud
    if (primero == null)
    {
        Console.WriteLine("El valor de n es mayor que la longitud de la lista.");
        return;
    }
    primero = primero.Siguiente;
}

// Movemos ambos punteros hasta que el primero llegue al final
while (primero != null)
{
    primero = primero.Siguiente;
    segundo = segundo.Siguiente;
}

// Ahora el puntero 'segundo' está justo antes del nodo a eliminar
if (segundo.Siguiente != null)
{
    segundo.Siguiente = segundo.Siguiente.Siguiente;
}

// Actualizamos la cabeza (por si se eliminó el primer nodo)
cabeza = dummy.Siguiente;
}

// Método para mostrar los valores de la lista
public void Mostrar()
{
    Nodo actual = cabeza;

```

```

while (actual != null)
{
    Console.Write(actual.Valor + " -> ");
    actual = actual.Siguiente;
}
Console.WriteLine("null");
}
}

```

```

class Program
{
    static void Main()
    {
        ListaEnlazada lista = new ListaEnlazada();

        // Insertamos algunos elementos
        lista.InsertarAlFinal(10);
        lista.InsertarAlFinal(20);
        lista.InsertarAlFinal(30);
        lista.InsertarAlFinal(40);
        lista.InsertarAlFinal(50);

        Console.WriteLine("Lista original:");
        lista.Mostrar();

        // Eliminamos el 2° nodo desde el final (debería eliminar 40)
        int n = 2;

        Console.WriteLine($"Eliminando el {n}° nodo desde el final...");
        lista.EliminarNesimoDesdeFinal(n);

        Console.WriteLine("\nLista después de la eliminación:");
    }
}

```



```

        lista.Mostrar();
    }
}

```

```

Lista original:
10 -> 20 -> 30 -> 40 -> 50 -> null

Eliminando el 2º nodo desde el final...

Lista después de la eliminación:
10 -> 20 -> 30 -> 50 -> null

```

Ejercicio 30: Separar la Lista en Pares e Impares Escribe un método void SepararParesImpares() que reorganice la lista de modo que todos los nodos con valores pares aparezcan antes que todos los nodos con valores impares, manteniendo el orden relativo entre los pares y entre los impares.

Script:

```
using System;
```

```
class Nodo
```

```

{
    public int Valor;      // Dato almacenado en el nodo
    public Nodo Siguiente; // Referencia al siguiente nodo

    public Nodo(int valor)
    {
        Valor = valor;
        Siguiente = null;
    }
}

```

```
class ListaEnlazada
```

```

{
    public Nodo cabeza; // Primer nodo de la lista

    // Método para insertar un nodo al final de la lista

```

```

public void InsertarAlFinal(int valor)
{
    Nodo nuevo = new Nodo(valor);

    if (cabeza == null)
    {
        cabeza = nuevo; // Si la lista está vacía, el nuevo nodo será la cabeza
    }
    else
    {
        Nodo actual = cabeza;
        while (actual.Siguiente != null)
        {
            actual = actual.Siguiente; // Recorremos hasta el último nodo
        }
        actual.Siguiente = nuevo; // Enlazamos el nuevo nodo al final
    }
}

```

// ♦ Método para separar la lista en pares e impares

```

public void SepararParesElImpares()
{
    if (cabeza == null)
    {
        Console.WriteLine("La lista está vacía.");
        return;
    }
}

```

// Creamos dos nuevas listas: una para pares y otra para impares

```

ListaEnlazada listaPares = new ListaEnlazada();
ListaEnlazada listaImpares = new ListaEnlazada();

```

```
Nodo actual = cabeza; // Empezamos desde la cabeza
```

```
// Recorremos toda la lista original
```

```
while (actual != null)
```

```
{
```

```
    if (actual.Valor % 2 == 0)
```

```
    {
```

```
        // Si el valor es par, lo agregamos a la lista de pares
```

```
        listaPares.InsertarAlFinal(actual.Valor);
```

```
    }
```

```
    else
```

```
    {
```

```
        // Si es impar, lo agregamos a la lista de impares
```

```
        listaImpares.InsertarAlFinal(actual.Valor);
```

```
    }
```

```
    actual = actual.Siguiente; // Pasamos al siguiente nodo
```

```
}
```

```
// Mostramos las dos listas resultantes
```

```
Console.WriteLine("Lista de números pares:");
```

```
listaPares.Mostrar();
```

```
Console.WriteLine("\nLista de números impares:");
```

```
listaImpares.Mostrar();
```

```
}
```

```
// Método para mostrar los valores de la lista
```

```
public void Mostrar()
```

```
{
```

```
    Nodo actual = cabeza;
```

```
    while (actual != null)
```

```
    {
```

```

        Console.Write(actual.Valor + " -> ");

        actual = actual.Siguiente;
    }
    Console.WriteLine("null");
}
}

```

```

class Program
{
    static void Main()
    {
        ListaEnlazada lista = new ListaEnlazada();

        // Insertamos algunos valores en la lista original
        lista.InsertarAlFinal(10);
        lista.InsertarAlFinal(15);
        lista.InsertarAlFinal(22);
        lista.InsertarAlFinal(33);
        lista.InsertarAlFinal(40);
        lista.InsertarAlFinal(51);

        Console.WriteLine("Lista original:");
        lista.Mostrar();

        Console.WriteLine("\nSeparando lista en pares e impares...");
        lista.SepararParesElImpares();
    }
}

```

```
Lista original:
10 -> 15 -> 22 -> 33 -> 40 -> 51 -> null

Separando lista en pares e impares...
Lista de números pares:
10 -> 22 -> 40 -> null

Lista de números impares:
15 -> 33 -> 51 -> null
```

Vigencia:

26/10/2025 hasta las 2:59 hr

Producto para entregar:

Archivo en formato c# con cada uno de los ejercicios con comentarios con el fin de saber cómo construyo el código y que obtuvo de salida.

Ing. Juan Carlos Arbeláez