Estructura de Datos

Mg. Cinthia J. Calderon Aquiño

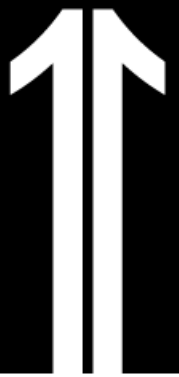
Semana 03

UPN.EDU.PE

Semana 03

Listas enlazadas Dobles.
Listas enlazadas Circulares.

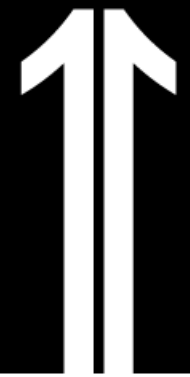
PRESENTACIÓN DE LA SESIÓN
Logro de la Sesión y Temario



Al término de la sesión, el estudiante aprende algoritmos de listas dobles, usándolos con eficacia.

- Listas enlazadas Dobles.
- Listas enlazadas Circulares.

Reflexiona



- ¿Qué es una lista enlazada doble?
- ¿Qué es un nodo doble?

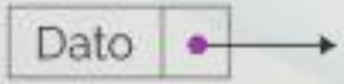
CLASIFICACIÓNES DE LAS LISTAS

Estructura básica de un nodo

Esta es la estructura básica de un nodo para crear listas de datos:

```
public class Nodo {
  public int info;
  public Nodo siguiente;
}
```

Representación gráfica de nodo:

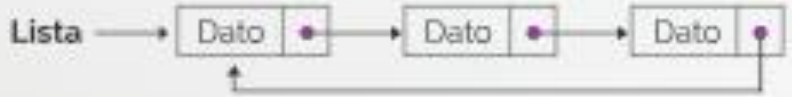


Listas de enlaces

1. Lista de enlace simple:

2. Lista de enlace doble

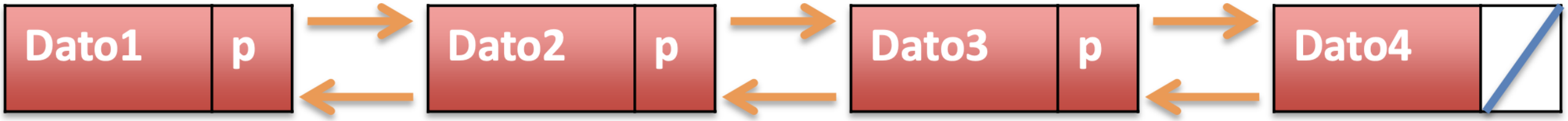
3. Lista enlace circular simple



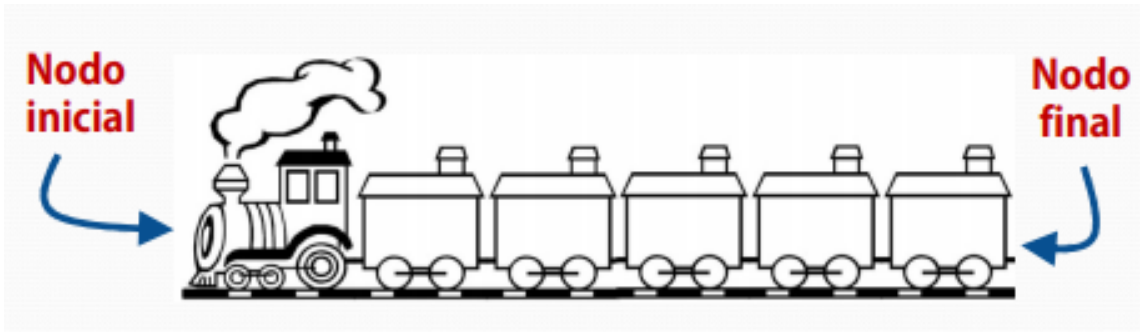
4. Lista enlace circular doble

LISTAS DOBLEMENTE ENLAZADAS

- Cada nodo contiene dos enlaces, uno a su nodo predecesor y el otro a su nodo sucesor. La lista es eficiente tanto en recorrido directo («adelante») como en recorrido inverso («atrás»).



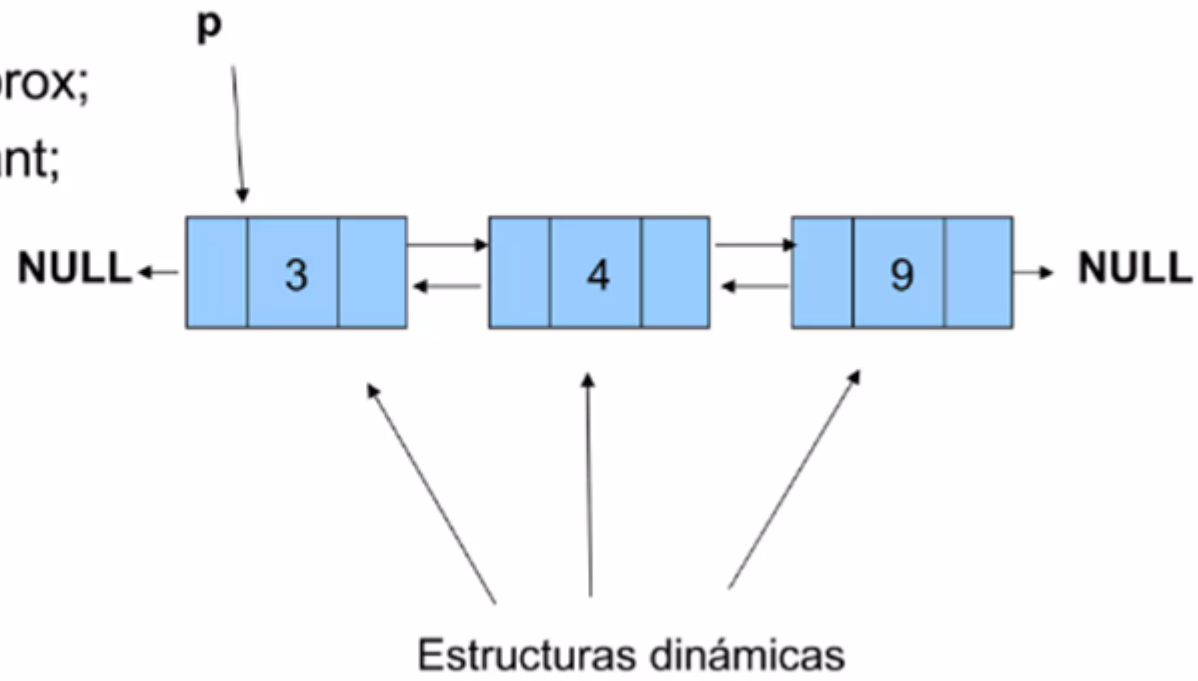
LISTAS DOBLEMENTE ENLAZADAS



- *Se verifica que la lista no esté vacía*
- *El recorrido empieza en el* `NodoInicial` *o en el* `NodoFinal`
- *Si se empieza por el* `NodoInicial` *entonces se avanza al próximo nodo a través del apuntador* `Siguiente`
- *Si se empieza por el* `NodoFinal` *entonces se retrocede al nodo anterior a través del apuntador* `Anterior`
- *En algunos casos es necesario guardar en una variable el nodo previo al cambiar de nodo*
- *El recorrido termina al llegar al nodo que apunta a nulo*

LISTAS DOBLEMENTE ENLAZADAS

```
struct l_doble
{
    int numero;
    struct l_doble *prox;
    struct l_doble *ant;
} L_DOBLE;

L_DOBLE *p;
```



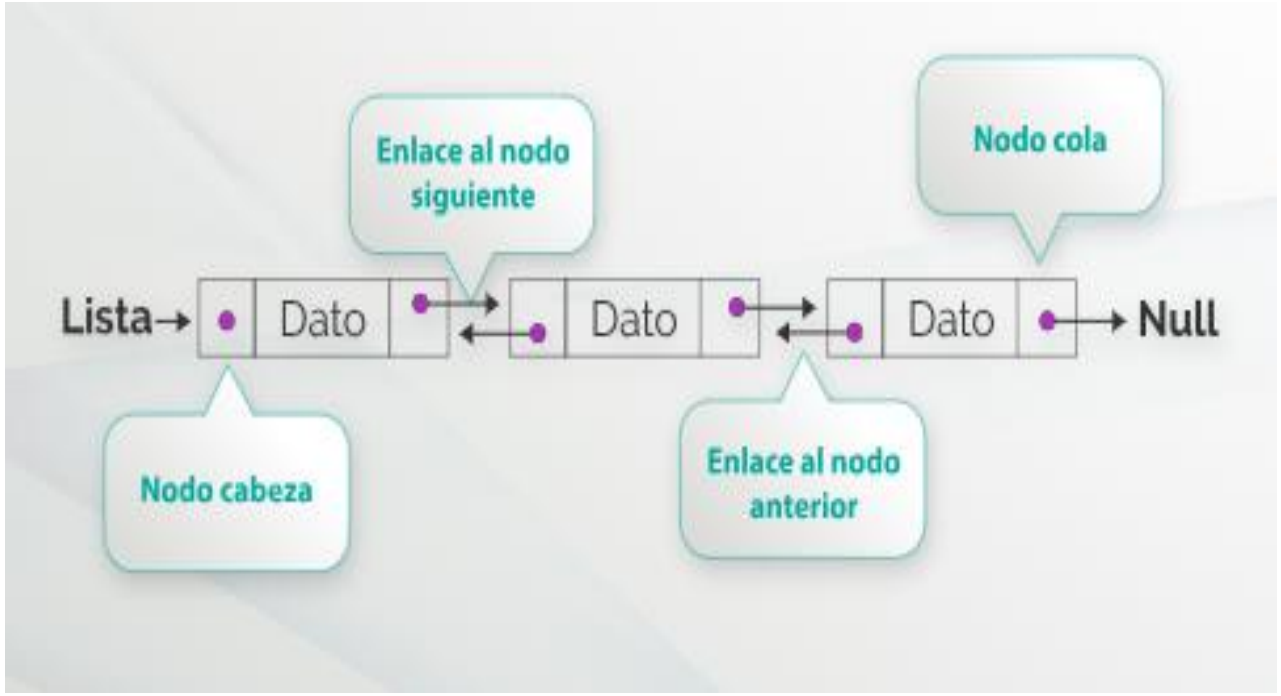
LISTAS DOBLEMENTE ENLAZADAS

Lista de enlace doble

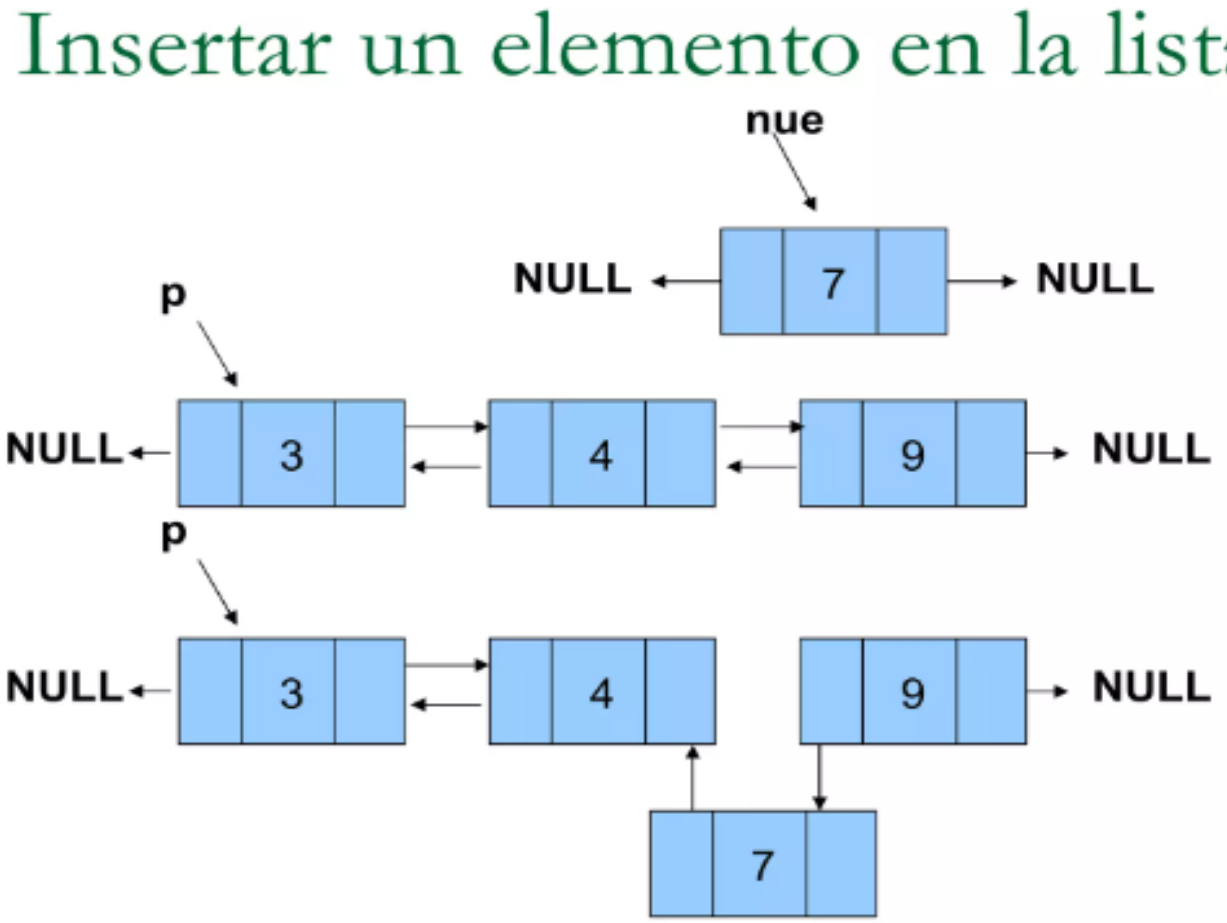
Es una estructura dinámica, donde el número de nodos puede variar dependiendo de las necesidades del proceso: agregando nodos por inserciones o disminuyendo nodos por eliminación.

La lista de enlace doble está caracterizada por tener únicamente dos enlaces: uno al siguiente nodo y otro al anterior nodo. Cuenta con un nodo cabeza y un nodo al final de la lista.

Se accede a la lista mediante el primer nodo de la lista llamado “cabeza” o “cabecera” y el último nodo llamado “cola”, cada enlace del nodo apuntará al siguiente y al anterior nodo.

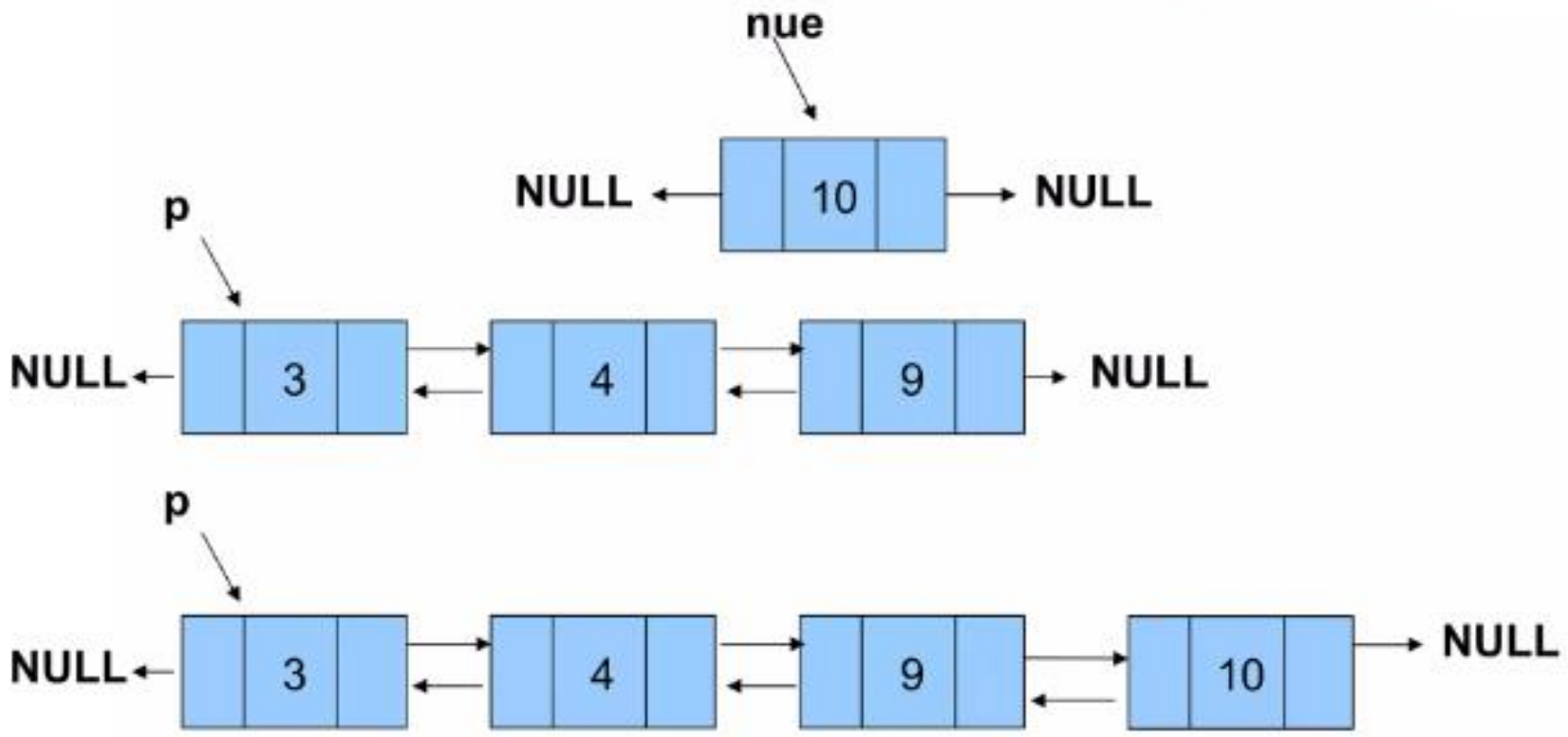


LISTAS DOBLEMENTE ENLAZADAS

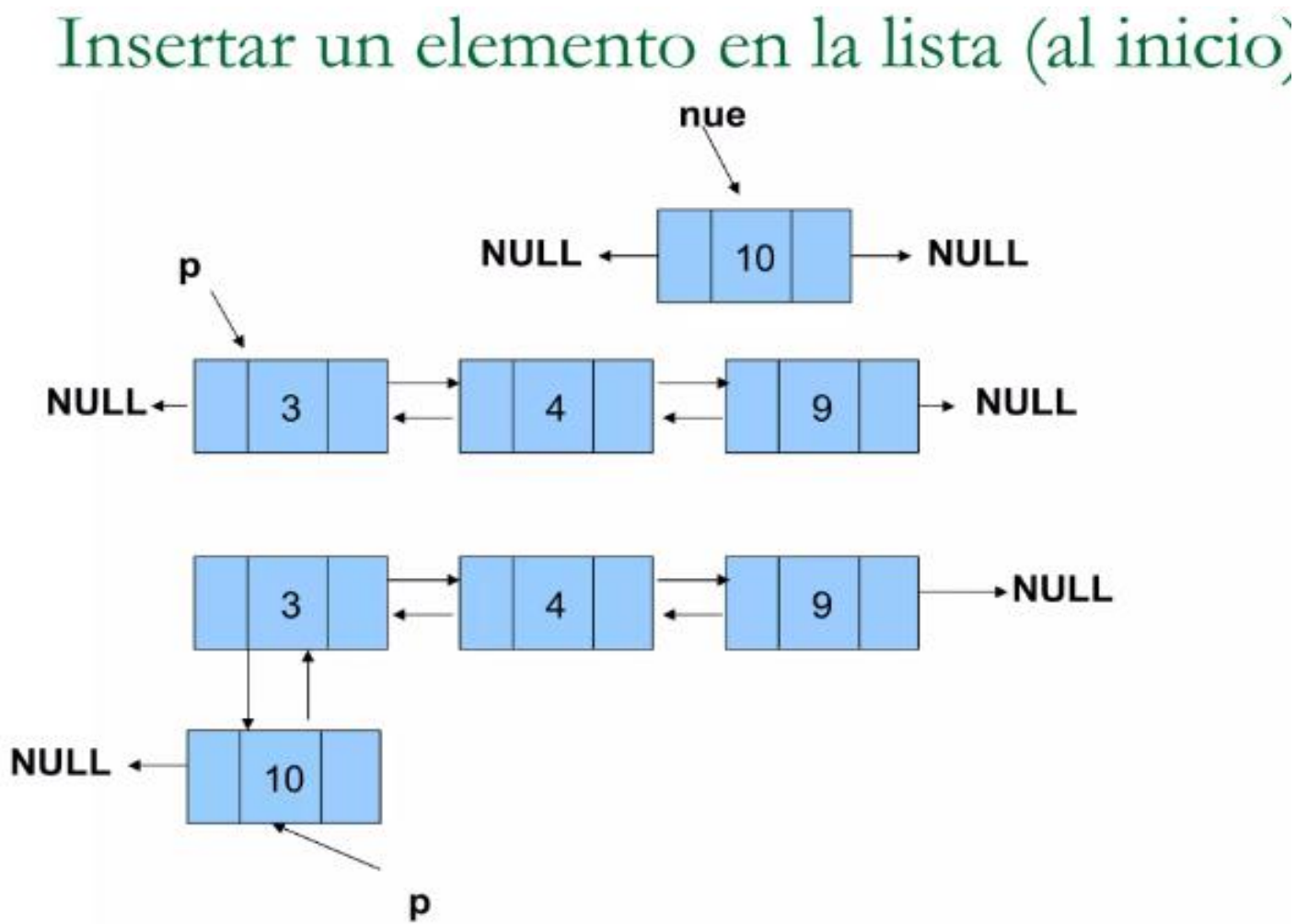


LISTAS DOBLEMENTE ENLAZADAS

Insertar un elemento en la lista (al final)

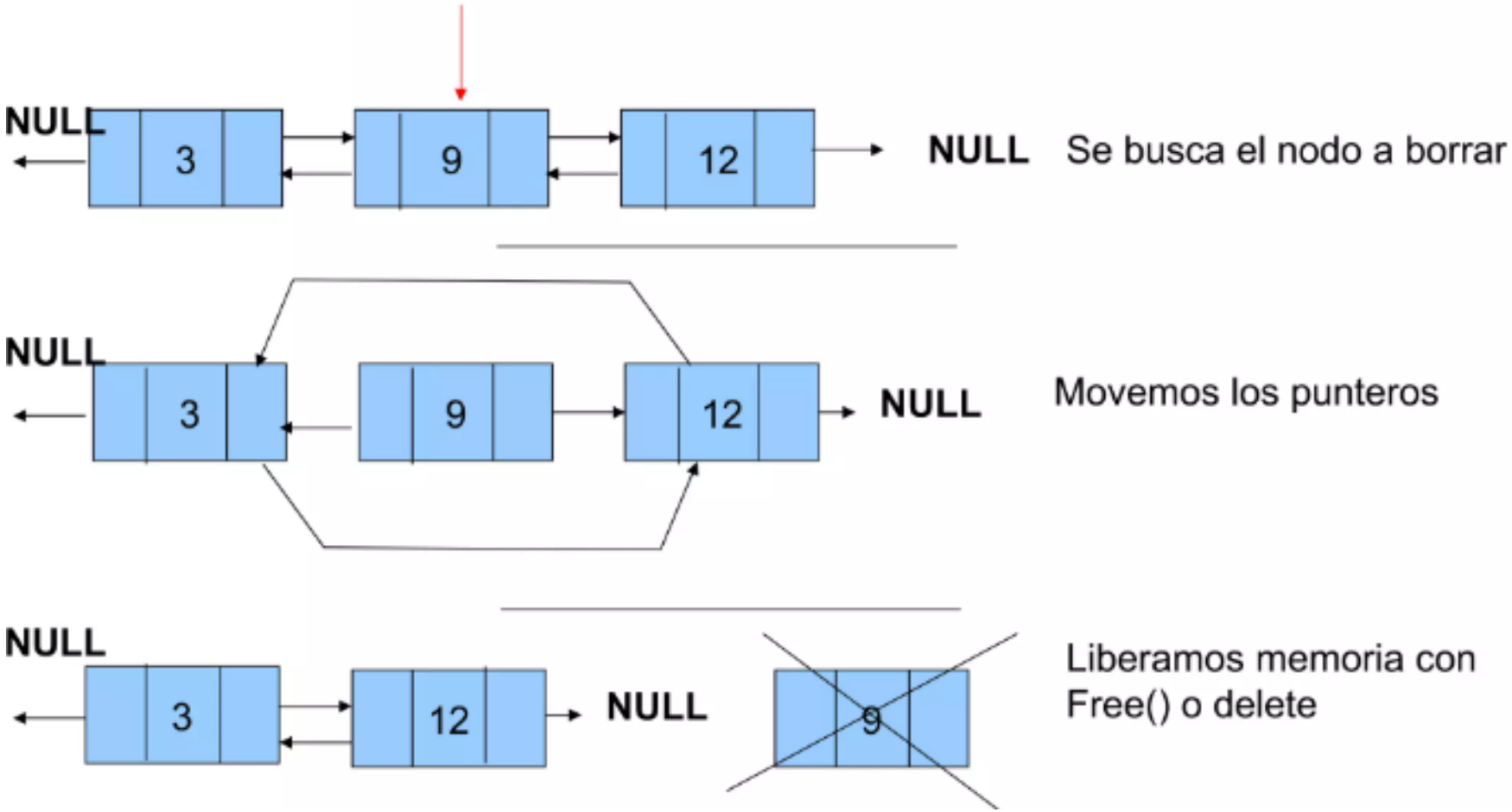


LISTAS DOBLEMENTE ENLAZADAS



LISTAS DOBLEMENTE ENLAZADAS

Eliminar un elemento de un nodo



LISTAS DOBLEMENTE ENLAZADAS

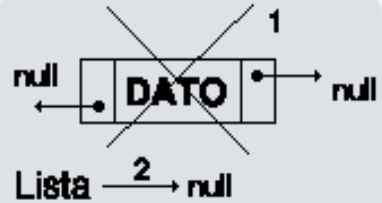
Eliminar el único nodo en una lista doblemente enlazada

Lista con un elemento

En este caso, ese nodo será el apuntado por Lista.

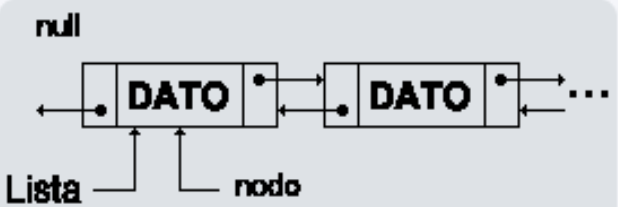
El proceso es simple:

1. Eliminamos el **nodo**.
2. Hacemos que Lista apunte a NULL.



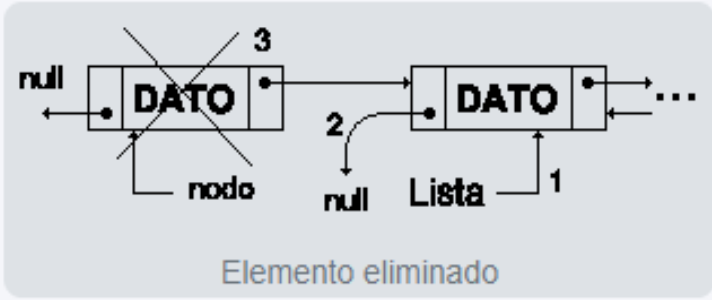
Lista vacía

Eliminar el primer nodo de una lista doblemente enlazada



Lista doblemente enlazada

Tenemos los dos casos posibles, que el nodo a borrar esté apuntado por Lista o que no. Si lo está, simplemente hacemos que Lista sea Lista->siguiente.



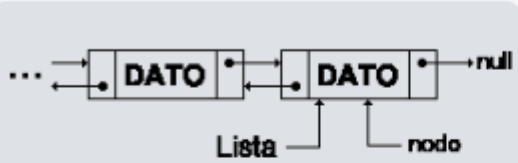
Elemento eliminado

1. Si **nodo** apunta a Lista, hacemos que Lista apunte a Lista->siguiente.
2. Hacemos que **nodo**->siguiente->anterior apunte a NULL
3. Borramos el nodo apuntado por **nodo**.

LISTAS DOBLEMENTE ENLAZADAS

Eliminar el último nodo de una lista doblemente enlazada

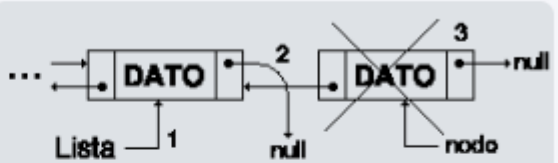
De nuevo tenemos los dos casos posibles, que el nodo a borrar esté apuntado por Lista o que no. Si lo está, simplemente hacemos que Lista sea Lista->anterior.



Lista doblemente enlazada

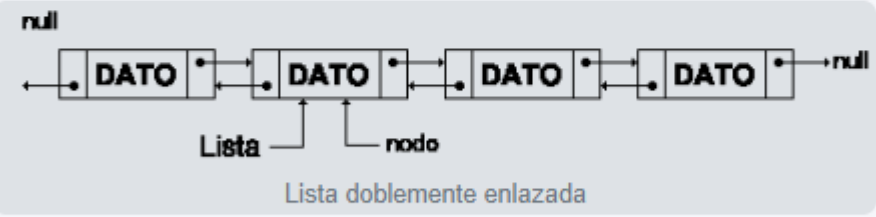
1. Si **nodo** apunta a Lista, hacemos que Lista apunte a Lista->anterior.
2. Hacemos que **nodo**->anterior->siguiente apunte a NULL
3. Borramos el nodo apuntado por **nodo**.

El paso 2 depara el nodo a borrar del resto de la lista, independientemente del nodo al que apunte Lista.



Elemento eliminado

Eliminar un nodo intermedio de una lista doblemente enlazada

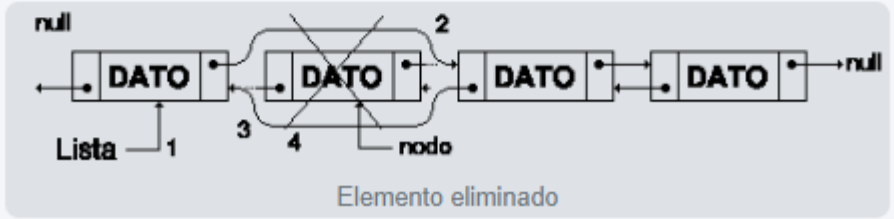


Lista doblemente enlazada

De nuevo tenemos los dos casos posibles, que el nodo a borrar esté apuntado por Lista o que no. Si lo está, simplemente hacemos que Lista sea Lista->anterior o Lista->siguiente

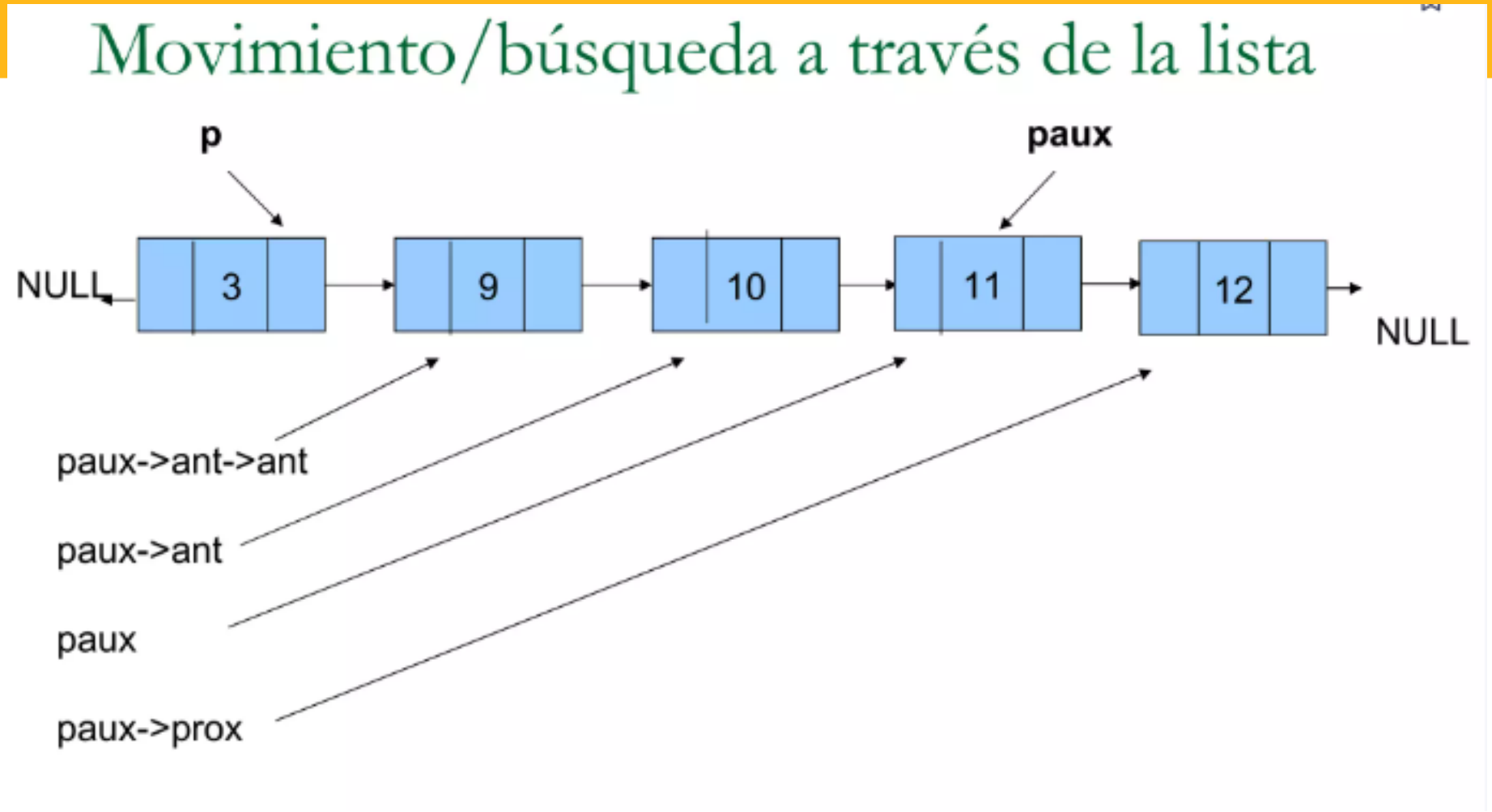
Se trata de un caso más general de los dos casos anteriores.

1. Si **nodo** apunta a Lista, hacemos que Lista apunte a Lista->anterior (o Lista->siguiente).
2. Hacemos que **nodo**->anterior->siguiente apunte a **nodo**->siguiente.
3. Hacemos que **nodo**->siguiente->anterior apunte a **nodo**->anterior.
4. Borramos el nodo apuntado por nodo.



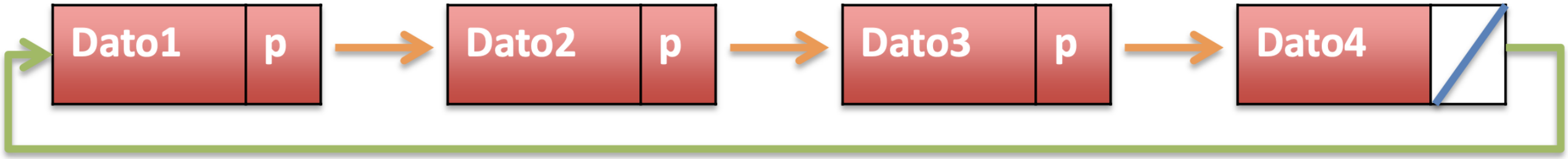
Elemento eliminado

LISTAS DOBLEMENTE ENLAZADAS



LISTA CIRCULAR SIMPLEMENTE ENLAZADA

- Una lista enlazada simplemente en la que el último elemento (cola) se enlaza al primer elemento (cabeza) de tal modo que la lista puede ser recorrida de modo circular («en anillo»).

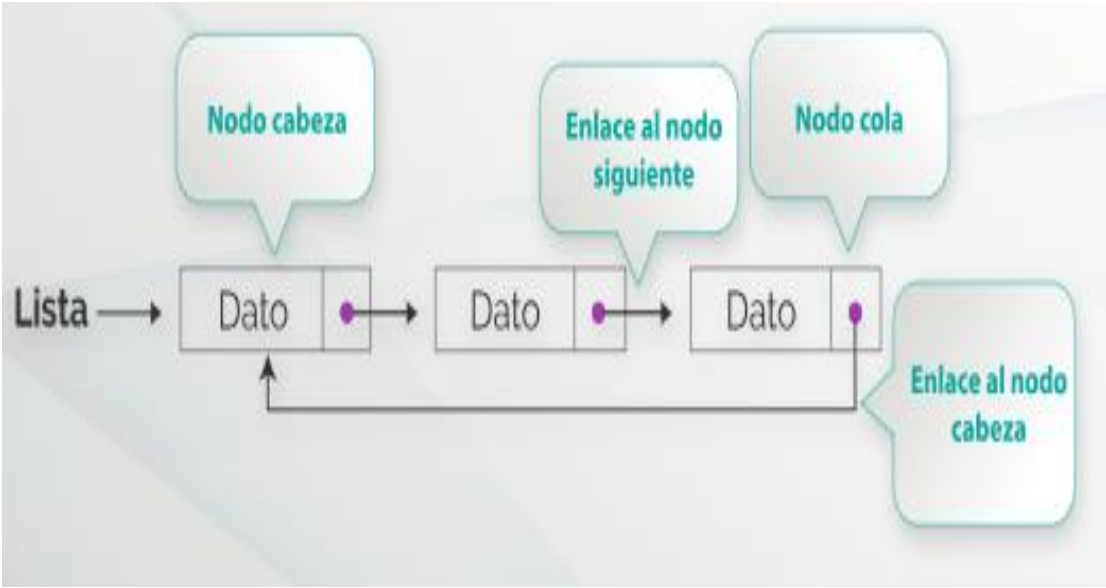


LISTA CIRCULAR DOBLEMENTE ENLAZADA

Lista circular de enlace simple

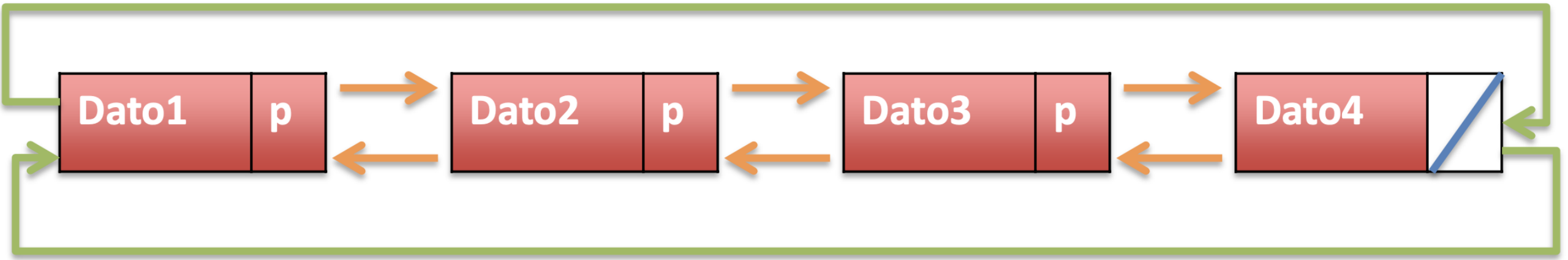
La lista circular de enlace simple es una estructura dinámica donde el número de nodos puede variar rápidamente dependiendo de los requerimientos del proceso: aumentando los nodos por inserciones a la lista o disminuyendo nodos por eliminación.

La lista circular de enlace simple se caracteriza por tener únicamente un enlace al siguiente nodo, pero que el enlace del último nodo apunta al primer nodo de la lista. Cuenta con un nodo cabeza y un nodo al final de la lista.



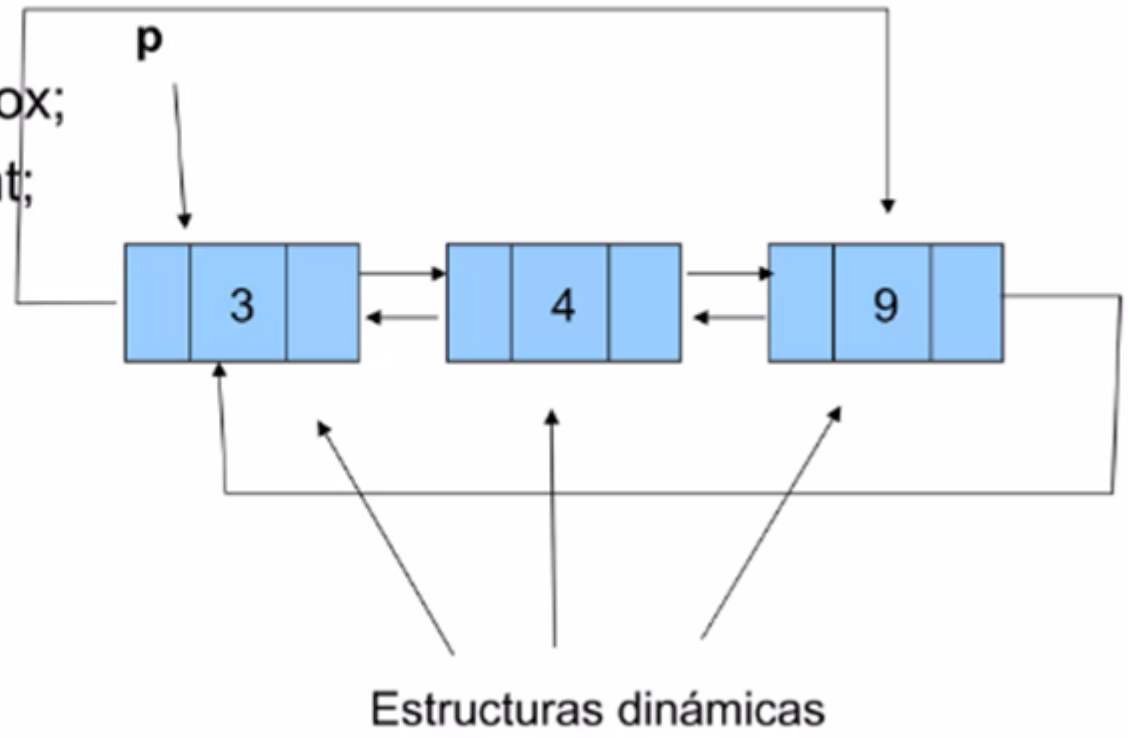
LISTA CIRCULAR DOBLEMENTE ENLAZADA:

- Una lista doblemente enlazada en la que el último elemento se enlaza al primer elemento y viceversa. Esta lista se puede recorrer de modo circular (en anillo) tanto en dirección directa («adelante») como inversa («atrás»).



LISTA CIRCULAR DOBLEMENTE ENLAZADA

```
struct l_circular
{
    int numero;
    struct l_circular *prox;
    struct l_circular *ant;
} L_CIRCULAR;

L_DOBLE *p;
```

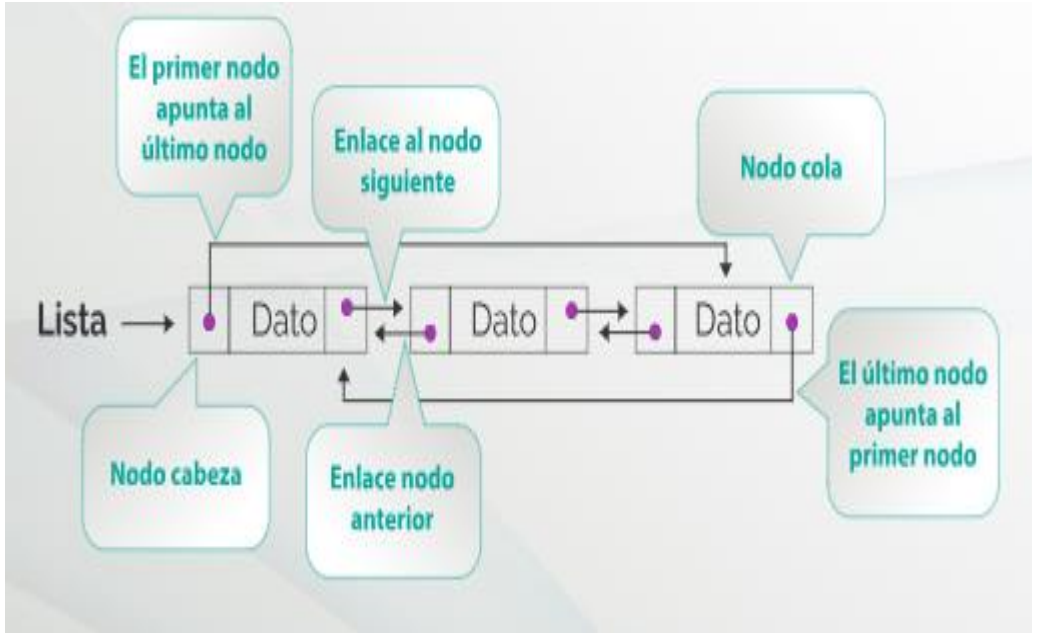


LISTA CIRCULAR DOBLEMENTE ENLAZADA

Lista circular de enlace doble

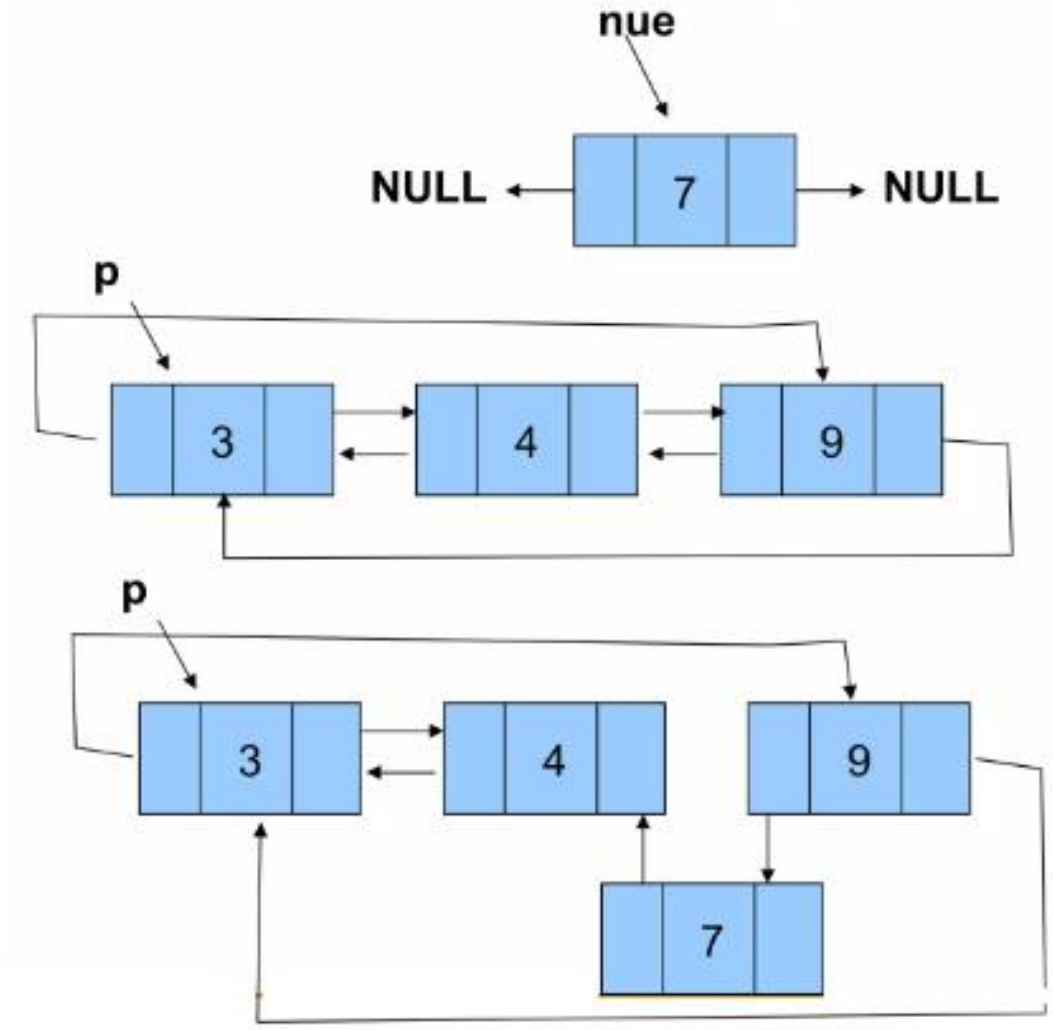
Es una estructura dinámica donde el número de nodos puede variar rápidamente dependiendo de los requerimientos del proceso: aumentando los nodos por inserciones a la lista o disminuyendo nodos por eliminación.

La lista circular de enlace doble se caracteriza por tener dos enlaces al siguiente nodo o predecesor y otro al anterior nodo de la lista o antecesor, pero que el enlace del último nodo apunta al primer nodo de la lista y el primer nodo (cabeza), apunta al último nodo de la lista (cola). Cuenta con un nodo cabeza y un nodo al final de la lista.



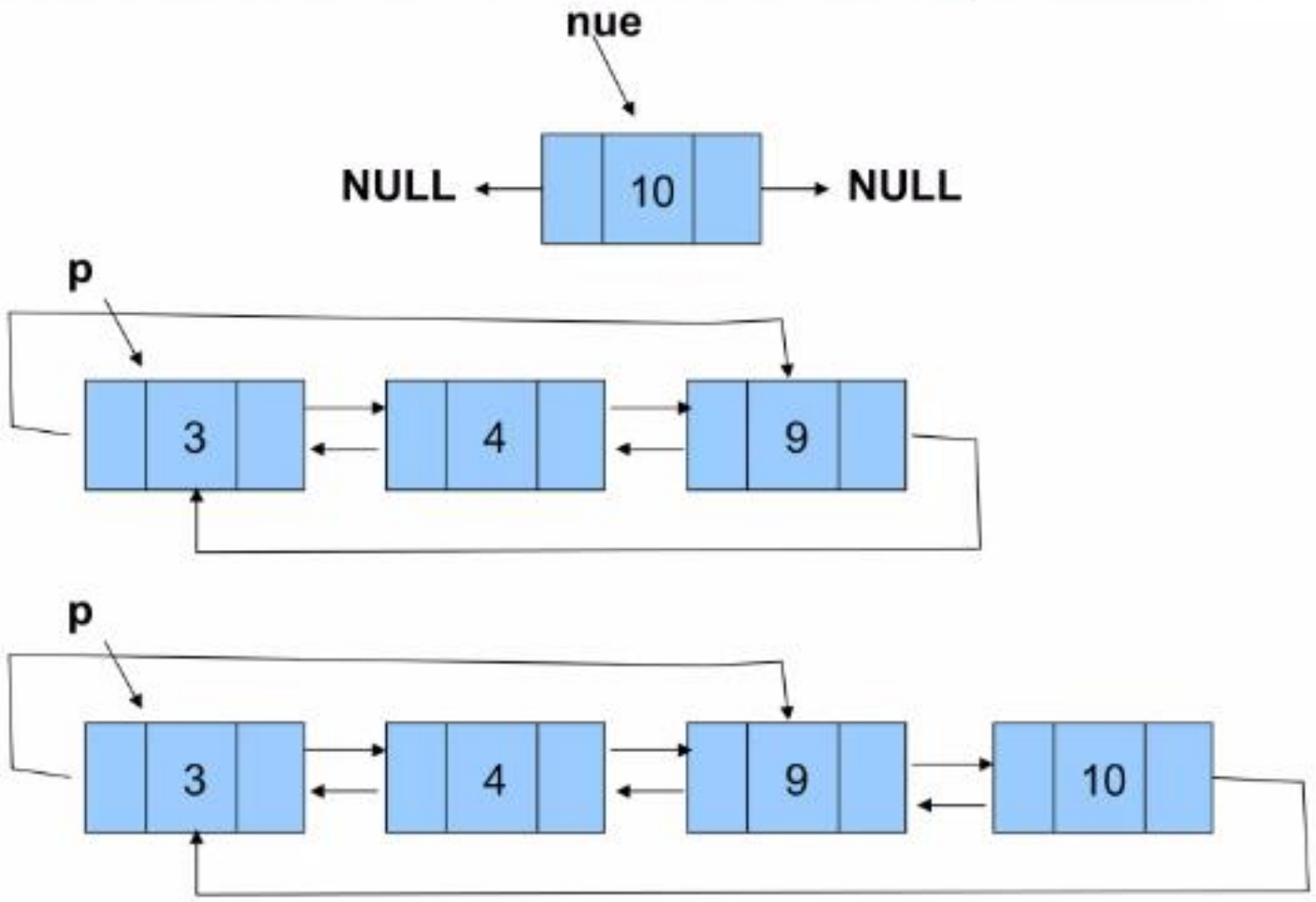
LISTA CIRCULAR DOBLEMENTE ENLAZADA

Insertar un elemento en la lista



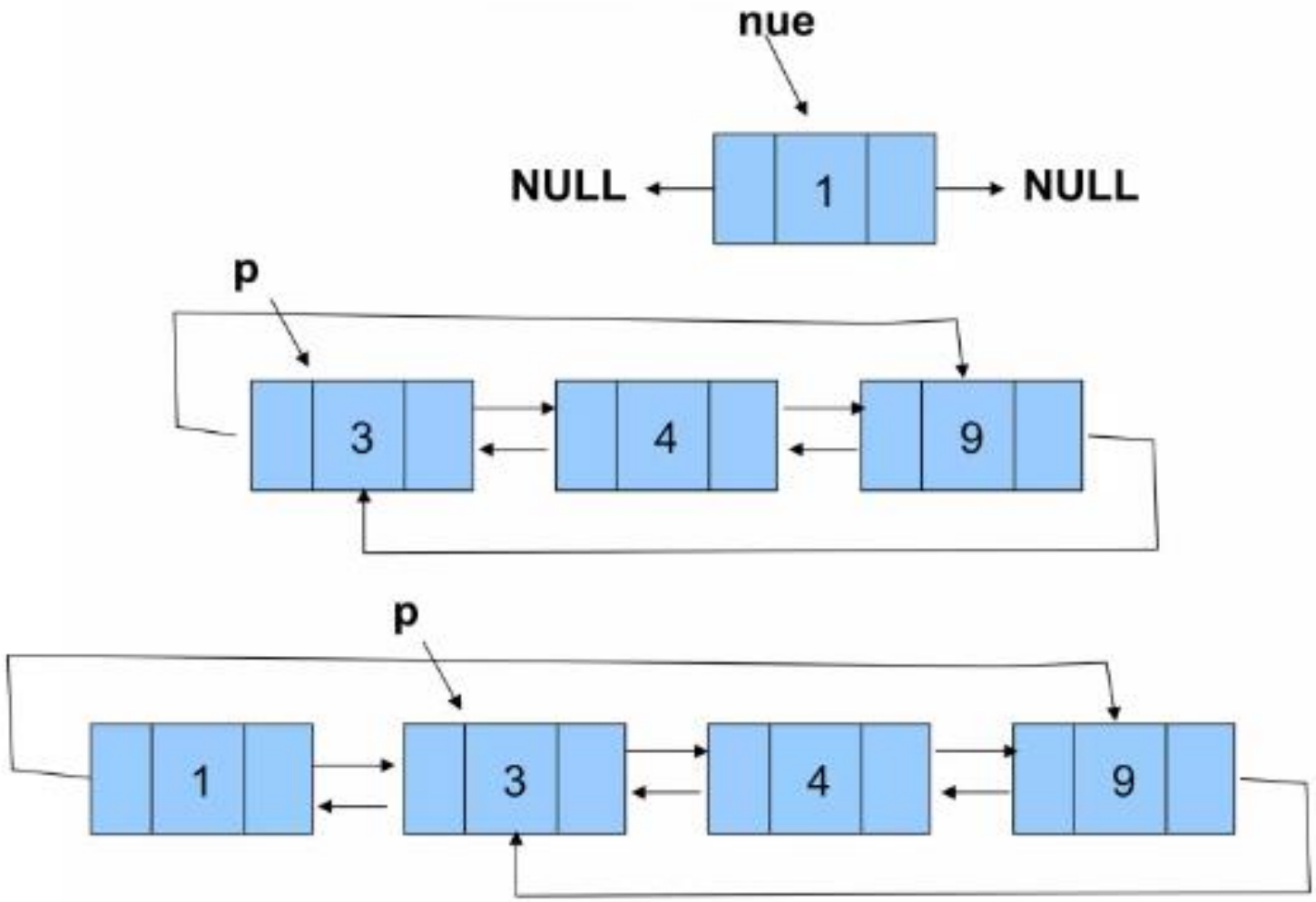
LISTA CIRCULAR DOBLEMENTE ENLAZADA

Insertar un elemento en la lista (al final)



LISTA CIRCULAR DOBLEMENTE ENLAZADA

Insertar un elemento en la lista (al inicio)



OPERACIONES EN LISTAS ENLAZADAS

- Insertar elementos en una lista enlazada
- Mostrar los elementos de una lista enlazada
- Buscar un elemento en una lista enlazada
- Eliminar un elemento en una lista enlazada

EJERCICIOS

Abrir un nuevo proyecto en Visual C# (el proyecto será en entorno CONSOLA).

1. Añada una clase nueva al proyecto, nómbrela **nodo** y codifique lo siguiente:

```
class nodo
{
    public int dato; //atributo que almacena el valor del nodo, en este caso entero
    public nodo siguiente; //atributo que señala al siguiente nodo, clase auto referenciada
}
```

2. Añada una clase nueva al proyecto, nómbrela **lista** y codifique lo siguiente para sus atributos y constructor:

```
class lista
{
    public nodo inicio; //cabeza de la lista

    //constructor por defecto
    public lista()
    { inicio = null; }
```

EJERCICIOS

2.1 Siempre dentro de la clase lista para el método que inserta en la cola de la lista

```
//método para insertar al final de la lista, como lo hace ArrayList
public void InsertarF(int item)
{
    nodo auxiliar = new nodo(); //nodo temporal que todavía no pertenece a la lista
    auxiliar.dato = item; //almacena en el atributo dato el valor recibido en el parámetro
    auxiliar.siguiente = null; //hace que el apuntador señale a null

    if(inicio == null) //verifica si la lista está vacia
    {
        inicio = auxiliar; //hacemos que nodo sea parte de la lista, lo hacemos la cabeza.
    }
    else
    {
        nodo puntero; //utilizamos este nodo para recorrer la lista, así no se mueve la cabeza
        puntero = inicio; //situamos a puntero señalando al mismo nodo que inicio
        while(puntero.siguiente!=null)
        {
            puntero = puntero.siguiente; //se desplaza por todos los nodos de la lista
        }
        puntero.siguiente = auxiliar; // hacemos que último nodo ahora señale al auxiliar
    }
}
```

EJERCICIOS

2.2 Para el método que inserta en la cabeza de la lista

```
//método insertar al inicio (insertar en la cabeza)
public void InsertarI(int item)
{
    nodo auxiliar = new nodo(); //nodo temporal que después se agrega a la lista

    auxiliar.dato = item; //almacena valor en el atributo dato
    auxiliar.siguiente = null; //hacemos que puntero siguiente señale a null

    if(inicio ==null)
    {
        inicio = auxiliar; //al estar vacia la cola lo hacemos la cabeza
    }
    else
    {
        nodo puntero; //para recorrer lista;
        puntero = inicio; //dos apuntadores señalando al mismo nodo
        inicio = auxiliar; //asignamos como cabeza al nodo auxiliar
        auxiliar.siguiente = puntero; //el puntero de auxiliar que ahora es cabeza se enlaza con
        // el nodo que era antes la cabeza y que tiene apuntador puntero
    }

}
```

EJERCICIOS

2.3 Para el método que elimina en la cabeza de la lista

```
//método para eliminar nodo que está a la cabeza de la lista
public void EliminarI()
{
    if(inicio == null) //cuando lista esté vacia
    { Console.WriteLine("Lista vacia, no se puede eliminar elemento"); }
    else
    { inicio = inicio.siguiente; //a quien estaba señalando inicio será nuevo inicio
    }
}
```

EJERCICIOS

2.4 Para el método que elimina en la cola de la lista

```
//método para eliminar nodo al final de la lista
public void EliminarF()
{
    if(inicio ==null) //cuando lista esté vacia
    { Console.WriteLine("Lista vacia, no se puede eliminar elemento"); }
    else
    {
        nodo punteroant, punteropost; //requiero dos punteros para mover porque no declaré la cola
        punteroant = inicio; //inicializo ambos en la cabeza de la lista
        punteropost = inicio;

        while (punteropost.siguiente!=null) //mientras puntero posterior no señale a null
        {
            punteroant = punteropost; //el puntero anterior será a quien señale el posterior
            punteropost = punteropost.siguiente; //puntero posterior será a quien señalaba antes el posterior
        }
        punteroant.siguiente = null; //con esto sacamos el que estaba al final de la lista, el último nodo
        //era el que estaba señalando el punteropost pero ahora el último será en donde se quedó punteroant
    }
}
```

2.5 Para el método que inserta por posición

```
// método para insertar en una posición específica de la lista
public void InsertarP(int item, int pos)
{
    nodo auxiliar = new nodo(); //definición de nuevo nodo con sus atributos
    auxiliar.dato = item;
    auxiliar.siguiente = null;

    if (inicio == null) //si lista está vacia
    {
        Console.WriteLine("La lista está vacia, por lo tanto se va a insertar en la 1a. posición");
        inicio = auxiliar; //inserto nodo
    }
    else
    {
        nodo puntero;
        puntero = inicio; //para recorrer lista
        if(pos ==1) //si la posición que pidieron es la 1, se inserta en la cabeza
        {
            inicio = auxiliar;
            auxiliar.siguiente = puntero;
        }
       else //si la posición solicitada no es la 1
        {
            for(int i=1; i<pos-1; i++)
            {
                puntero = puntero.siguiente;
                if (puntero.siguiente == null) //si llegamos al final de la lista
                    break;
            }

            nodo punteronext; //apuntador de ayuda
            punteronext = puntero.siguiente; //a quien señalaba el puntero ahí se ubicará el apuntador punteronext
            puntero.siguiente = auxiliar; //ahora el apuntador puntero señalará a al nodo auxiliar (el nuevo)
            auxiliar.siguiente = punteronext; //el nodo recien ingresado señalará a punteronext
            //con estas cuatro líneas anteriores es cómo se corta momentaneamente la lista y luego se une de nuevo
        }
    }
}
```

EJERCICIOS

2.6 Para el método que muestra el contenido de la lista

```
    //método que muestra el contenido de la lista
    public void mostrar()
    {
        if(inicio ==null) //si la lista está vacia
        { Console.WriteLine("La lista está vacia"); }
        else
        {
            nodo puntero;
            puntero = inicio; //inicializamos a puntero en el mismo nodo que la cabeza;
            Console.Write("{0} -> \t", puntero.dato); //imprimir valor de nodo. Write y no WriteLine
            while(puntero.siguiente!=null)
            {
                puntero = puntero.siguiente; //avanzamos un nodo en la lista
                Console.Write("{0} -> \t", puntero.dato); //usar Write para que no salte de linea
            }
        }
        Console.WriteLine();
    }
} //fin de la clase lista
```

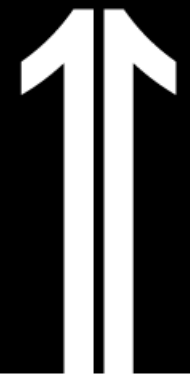
3. Una vez concluida la clase lista para poder utilizarla debemos crear los objetos pertinentes, es por ello que iremos ahora a la clase Program y en el Main escribiremos:

```
static void Main(string[] args)
{
    //creamos una instancia de la clase lista para que podamos utilizar los métodos
    lista listanueva = new lista();

    //le agregamos cuatro nodos a la lista
    listanueva.InsertarI(40);
    listanueva.InsertarI(30);
    listanueva.InsertarI(20);
    listanueva.InsertarI(10);
    //mostramos qué hay contenido dentro de la lista
    listanueva.mostrar();
```

4. Ahora ejecute el código trabajado y observe lo que realiza el programa

5. Una vez que ha realizado esto agregue las siguientes líneas a su código en el Main

```
//insertando en posición
listanueva.InsertarP(220, 3);
listanueva.mostrar();

Console.ReadKey();
```



PROBLEMAS

DECLARACIÓN DE UN NODO

1. Generar una lista Doble en consola que permita lo siguiente:
   - ❑Insertar al inicio
   - ❑Insertar al final
   - ❑Buscar un elemento de la lista
   - ❑Reemplazar un nodo de la lista
   - ❑Eliminar un nodo al inicio
   - ❑Eliminar un nodo al final
   - ❑Mostrar de inicio a fin
   - ❑Mostrar de fin a inicio
   - ❑Salir del programa

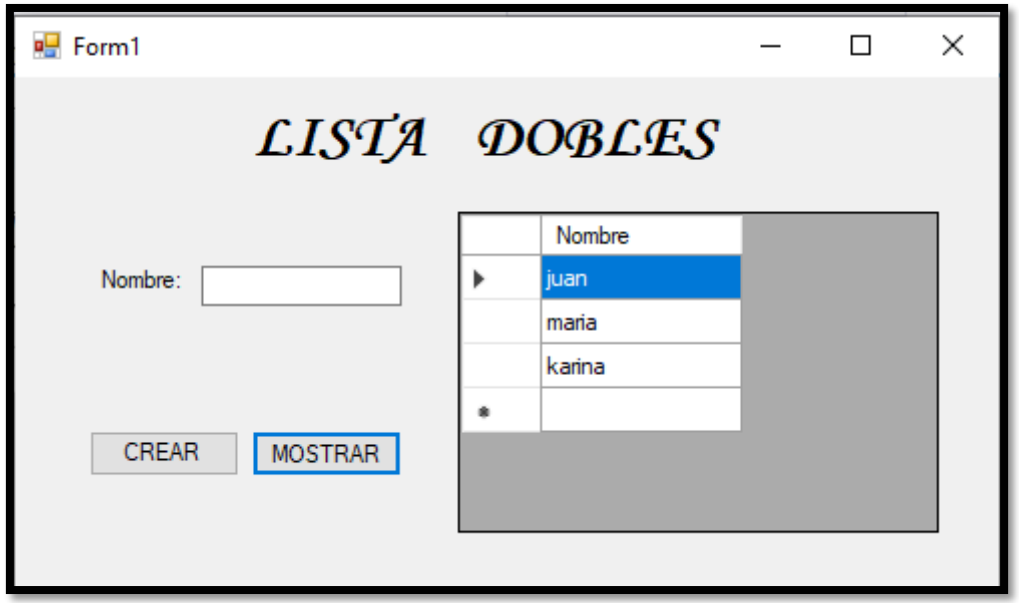
DECLARACIÓN DE UN NODO

2. Generar una lista Circular en consola que permita lo siguiente:

```
Console.WriteLine("\n");
Console.WriteLine(" |-------------------------------|");
Console.WriteLine(" |      * LISTA  CIRCULAR   *     |");
Console.WriteLine(" |---------------  |--------------|");
Console.WriteLine(" | 1.-Insertar     | 2. Buscar    |");
Console.WriteLine(" | 3.-Modificar    | 4. Eliminar  |");
Console.WriteLine(" | 5.-Mostrar      | 6. Salir     |");
Console.WriteLine(" |---------------  -|--------------|");
```

DECLARACIÓN DE UN NODO

3. Crear un aplicativo para el prototipo del ingreso de los usuarios. Mostrar lo siguiente:
   - ❑Insertar los nombres en la lista doble
   - ❑Mostrar los nombre en la lista doble



CLASE - NODO

```
class Nodo
   {
      public Nodo siguiente;
      public Nodo anterior;

      public string nombre;

      public Nodo (string nom)
      {
         siguiente = null;
         anterior = null;
         nombre = nom;
      }

      //get set para siguiente
      public Nodo getSiguiente()
      {
         return siguiente;
      }

      public void setSiguiente(Nodo set)
      {
         siguiente = set;
      }
      //get set para anterior
      public Nodo getAnterior()
      {
         return anterior;
      }

      public void setAnterior(Nodo ant)
      {
         anterior = ant;
      }
}
```

CLASE - LISTA

```
class Lista
   {
      //declaramos variables de tipo nodo
      public Nodo primero;
      public Nodo ultimo;
      public Nodo raiz;

      //creamos el constructor
      public Lista()
      {
         primero = null;
         ultimo = null;
         raiz = null;
      }
```

```
//creamos el metodo para crear los nodos
      public void insertar(string nom)
      {
         if(primero == null)
         {
            Nodo nuevo = new Nodo(nom);
            primero = nuevo; //se crea el primer nodo
            ultimo = primero; //el nuevo nodo para a ser el ultimo
         }
         else
         {
            Nodo nuevo = new Nodo(nom);
            ultimo.siguiente = nuevo;
            nuevo.anterior = ultimo;
            ultimo = nuevo;
         }
      }
   }
```

CLASE - FORM1

```
public partial class Form1 : Form
  {
      Lista lista = new Lista();
      public Form1()
      {
          InitializeComponent();
      }

      private void BtnCrear_Click(object sender,
EventArgs e)
      {

lista.insertar(Convert.ToString(TxtNombre.Text));
          TxtNombre.Text = "";
      }
```

```
private void BtnMostrar_Click(object sender, EventArgs e)
      {
          lista.raiz = lista.primero;
          while(lista.raiz !=null)
          {
             //MessageBox.Show(lista.raiz.nombre);
                lista.raiz = lista.raiz.siguiente; //apunta el nodo siguiente
                lista.imprimir(dgvLista);
          }

          lista.raiz = lista.ultimo;
          while (lista.raiz != null)
          {
             //MessageBox.Show(lista.raiz.nombre);
            lista.raiz = lista.raiz.anterior; //apunta el nodo anterior
          }
      }
  }
```

ACTIVIDAD DE CLASE

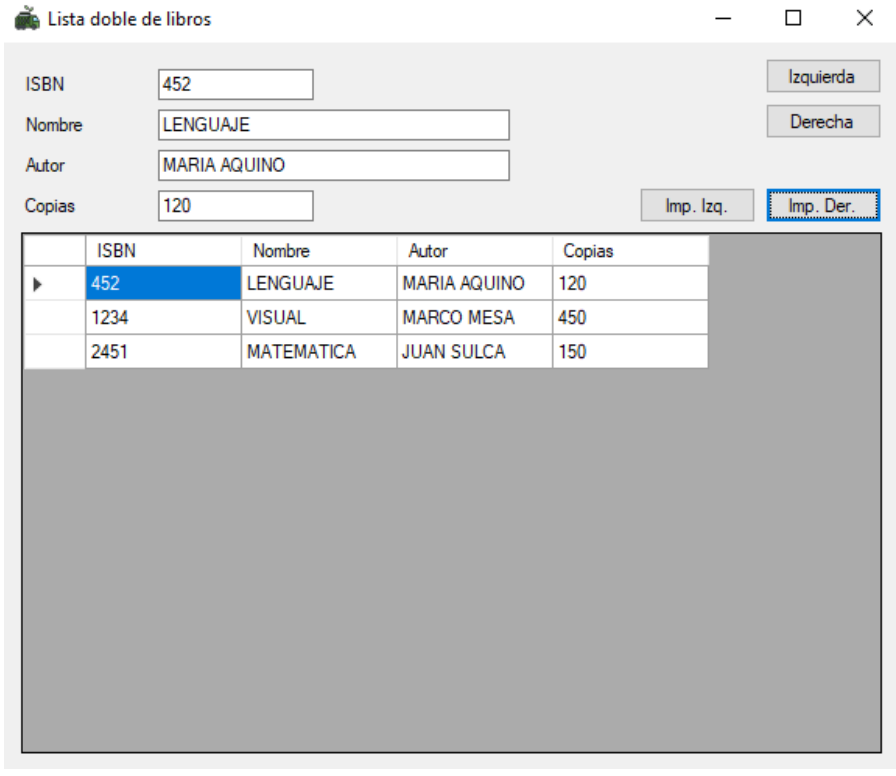
PROBLEMAS

PROBLEMAS

Problema 1

1. En la biblioteca de la San Marcos se pide en la área de TI la creación de un sistema para el control de libros por lo cual se requiere tener ISBN, Nombre, Autor y Paginas.

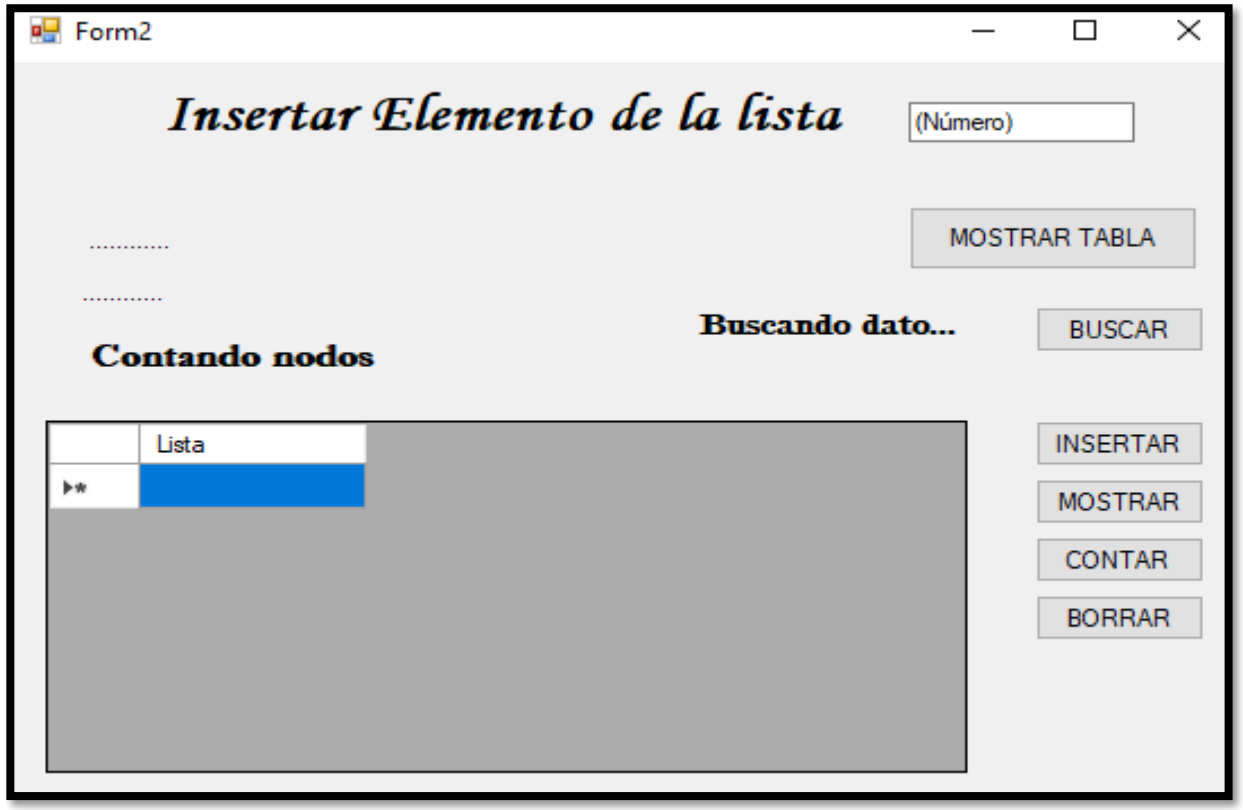
a) Generar el botón Izquierda en la lista
b) Generar el botón Derecha en la lista
c) Ordenar por izquierdo y derecho



PROBLEMAS

Problema 2- LISTA CIRCULAR

2. Implementar una lista circular que permita ingresar una lista de números. Por lo cual muestre con los siguientes herramienta tal como imagen.



CONCLUSIONES

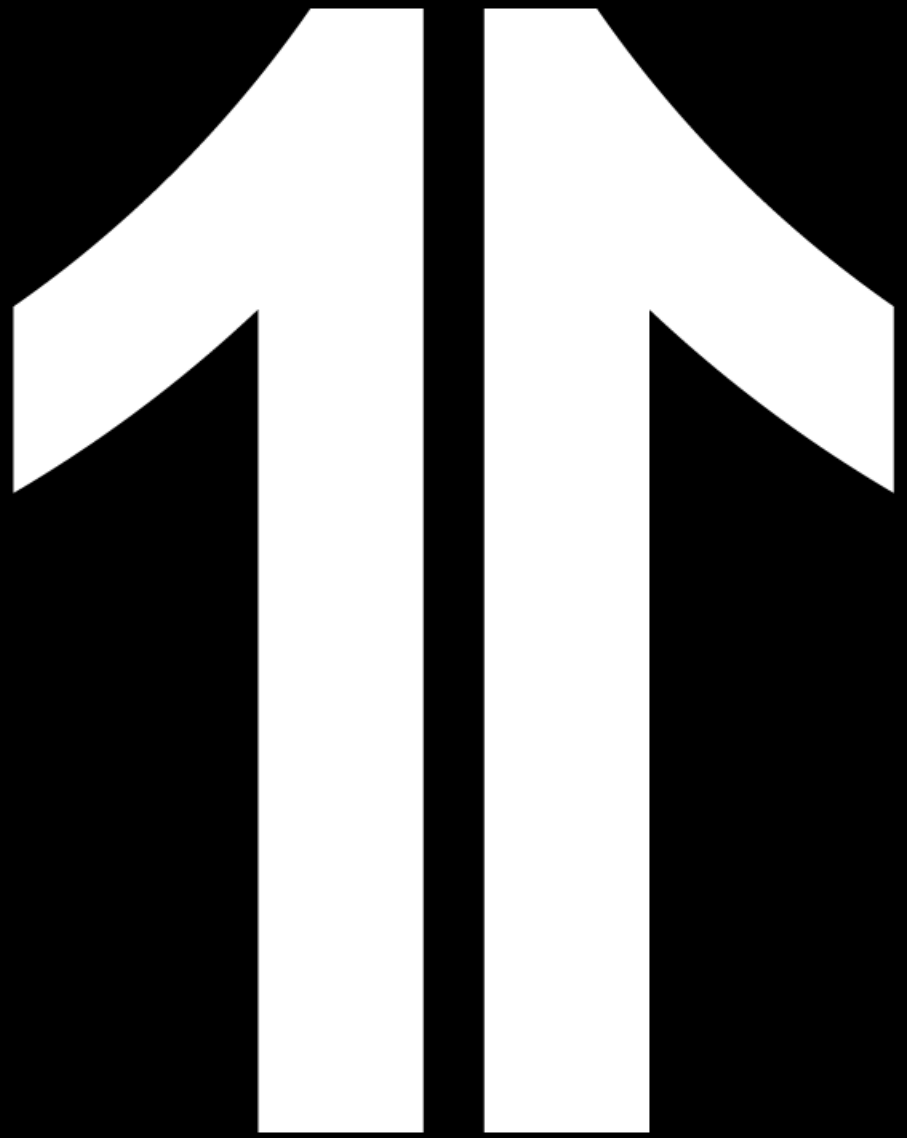
La **lista circular** es una estructura de datos homogénea, dinámica y lineal. Tienen un puntero a un elemento de la lista para poder iniciar su recorrido.

La **lista Doble** es una estructura de datos dinámica que se compone de un conjunto de nodos en secuencia enlazados mediante dos apuntadores (uno hacia adelante y otro hacia atrás)

¿Preguntas o comentarios?

BIBLIOGRAFIA REFERENCIAL

- Ceballos Sierra, F. Microsoft C#: Curso de Programación (2a.ed.) 2014 https://elibronet.eu1.proxy.openathens.net/es/lc/upnorte/titulos/106417
- Cesar Liza Avila; Estructura de datos con C/C++



GRACIAS


UPN
UNIVERSIDAD
PRIVADA
DEL NORTE