ARBOL BINARIO_ SEMANA 9

Crear un ejercicio con **ARBOL BINARIO** en el lenguaje de programación en c# en forma gráfica con formulario que tenga clase **Árbol** con métodos **(preOrden, inOrden, postOrden, insertar, verArbol, buscar, eliminar, esHoja, eliminarhoja, unhijo, doshijos, reemplazoderecho y cargaMasiva)** clase **Libros** con atributos **(código, nombre, copias)** y Nodo. **No se permite get; set; ni List<>**

Desarrollar los siguientes métodos:

- ❑ Ingresar el nuevo Árbol
- ❑ Mostrar en el Árbol
- ❑ Búsqueda por copias
- ❑ La búsqueda por DNI
- ❑ Ingresar PreOrden, InOrden y PostOrden

Control	Nombre sugerido	Función
TextBox	txtCodigo	Ingresar código del libro
TextBox	txtNombre	Ingresar nombre del libro
TextBox	TxtCopias	Permite ingresar la cantidad de copias del libro. Este campo se usa también para buscar y eliminar nodos
DataGridView	dgvLista	Muestra los resultados de los recorridos (PreOrden, InOrden, PostOrden). Cada fila representa un nodo con su código, nombre y copias.
DataGridView	dgvArbol	Representa visualmente la estructura del árbol, colocando cada nodo en su nivel correspondiente. Útil para observar la organización jerárquica del árbol.
MenuStrip	TsbNuevo	Inserta un nuevo libro en el árbol binario
	TsblArbol	Muestra gráficamente la estructura del árbol en el dgvArbol.
	TsmBuscar	Busca un nodo por el campo copias
	TsbEliminar	Elimina un nodo del árbol por el número de copias.
	TsmPreOrden	Realizan y muestran los recorridos del árbol
	TsmInOrden	Realizan y muestran los recorridos del árbol
	TsmPostOrden	Realizan y muestran los recorridos del árbol
2 Label		Se utilizan para indicar visualmente al usuario qué información debe ingresar en cada TextBox

```
14 referencias
public class Libro
{
    public string codigo, nombre;
    public int copias;

    1 referencia
    public Libro()
    {
        codigo = nombre = "";
        copias = 0;
    }

    1 referencia
    public Libro(string c, string n, int co)
    {
        codigo = c;
        nombre=n;
        copias = co;
    }
    1 referencia
    public override string ToString()
    {
        return codigo + " - " + nombre  + " - " + copias;
    }
}
```

```
29 referencias
internal class Nodo
{
    public Libro dato;
    public Nodo izq, der;

    0 referencias
    public Nodo()
    {
        dato = new Libro();
        izq = der = null;
    }

    1 referencia
    public Nodo(Libro x)
    {
        dato = x;
        izq = der = null;
    }
}
```

```
3 referencias
internal class Arbol
{
    public Nodo raiz;

    1 referencia
    public Arbol()
    {
        raiz = null;
    }

    2 referencias
    public void preOrden(Nodo r)
    {
        if (r != null)
        {
            //imprimir raiz. dato, mostrar, operar
            preOrden(r.izq);
            preOrden(r.der);
        }
    }
```

```
3 referencias
public void preOrden(Nodo r, DataGridView L)
{
    if (r !=null)
    {
        L.Rows.Add(r.dato.codigo + " - " +
            r.dato.nombre + " - " +
            r.dato.copias);
        preOrden(r.izq,L);
        preOrden(r.der,L);
    }
}

2 referencias
public void inOrden(Nodo r)
{
    if (r != null)
    {
        inOrden(r.izq);
        //imprimir raiz. dato ....
        //mostrar la raiz y despues del dato
        inOrden(r.der);
    }
}

3 referencias
public void inOrden(Nodo r, DataGridView L)
{
    if (r != null)
    {
        inOrden(r.izq,L);
        L.Rows.Add(r.dato.codigo + " - " +
            r.dato.nombre + " - " +
            r.dato.copias);
        inOrden(r.der,L);
    }
}

2 referencias
public void PostOrden(Nodo r)
{
    if(r != null)
    {
        PostOrden(r.izq);
        PostOrden(r.der);
        //imprimir raiz. dato, mostrar.....
    }
}
```

```
3 referencias
public void PostOrden(Nodo r, DataGridView L)
{
   if (r != null)
    {
        PostOrden(r.izq, L);
        PostOrden(r.der, L);
        L.Rows.Add(r.dato.codigo + " - " +
                r.dato.nombre + " - " +
                r.dato.copias);
    }
}
```

```
3 referencias
public void insertar(ref Nodo r, Libro dato)
{
    if (r == null) r = new Nodo(dato);
    else
        if (dato.copias > r.dato.copias)
        insertar(ref r.der, dato);
    else
        if (dato.copias < r.dato.copias)
        {
        insertar(ref r.izq, dato);
    }
    else MessageBox.Show("Elementos repetitivos ");
}
```

```
3 referencias
public void verArbol(Nodo r, Libro[]A, int[]pos,int n, ref int ce)
{
   if(r != null)
   {
       verArbol(r.izq, A, pos, n + 1, ref ce);
       A[ce] = r.dato;
       pos[ce] = n;//en la fila se dibujar en el datagridView
       ce++;//cantidad de elementos, indica cuantos nodos existe.
       verArbol(r.der, A, pos, n + 1, ref ce);
   }
}
```

```
1 referencia
public Libro buscar(string codigo)
{
   return buscarPorCodigo(raiz, codigo);
}
```

```
3 referencias
private Libro buscarPorCodigo(Nodo r, string codigo)
{
    if (r == null)
        return null;

    if (r.dato.codigo.Equals(codigo, StringComparison.OrdinalIgnoreCase))
        return r.dato;

    // Buscar primero en el subárbol izquierdo
    Libro encontrado = buscarPorCodigo(r.izq, codigo);
    if (encontrado != null)
        return encontrado;

    // Luego buscar en el subárbol derecho
    return buscarPorCodigo(r.der, codigo);
}
```

```
1 referencia
public bool eliminar(string copias)
{
    Nodo aux = raiz;
    Nodo padre = null;
    bool eshijoizq = true;


    if (!int.TryParse(copias, out int copiasInt))
    {
        return false; // Entrada inválida
    }

    while (aux != null && aux.dato.copias != copiasInt)
    {
        padre = aux;
        if (copiasInt < aux.dato.copias)
        {//si es menor
            aux = aux.izq;
            eshijoizq = true;
        }
        else
        {// si es mayor
            aux = aux.der;
            eshijoizq = false;
        }

        if (aux == null)
        {// no hay nodo
            return false;
        }
    }//fin mientras
    if (esHoja(aux))
    {     //hoja
        eliminarhoja(padre, aux, eshijoizq);
    }
    else
    {                     //un hijo
        if (aux.izq == null || aux.der == null)
        {
            unhijo(padre, aux, eshijoizq);
        }
        else
        {                     // 2 hijos
            doshijos(padre, aux, eshijoizq);
        }

    }
    return true;
}
```

```
1 referencia
public bool esHoja(Nodo r)
{
    return (r.izq == null && r.der == null);
}
1 referencia
private void eliminarhoja(Nodo padre, Nodo aux, bool eshijoizq)
{
    if (aux == raiz)
    {
        raiz = null;
    }
    else
    {
        if (eshijoizq)
        {
            padre.izq = null;
        }
        else
        {
            padre.der = null;
        }
    }
}
```

```
1 referencia
private void unhijo(Nodo padre, Nodo aux, bool eshijoizq)
{
    if (aux == raiz)
    {
        if (aux.izq == null)
        {
            raiz = aux.der;
        }
        else
        {
            raiz = aux.izq;
        }
    }
    else
    {
        if (eshijoizq)
        {
            if (aux.izq == null)
            {//derecho
                padre.izq = aux.der;
            }
```

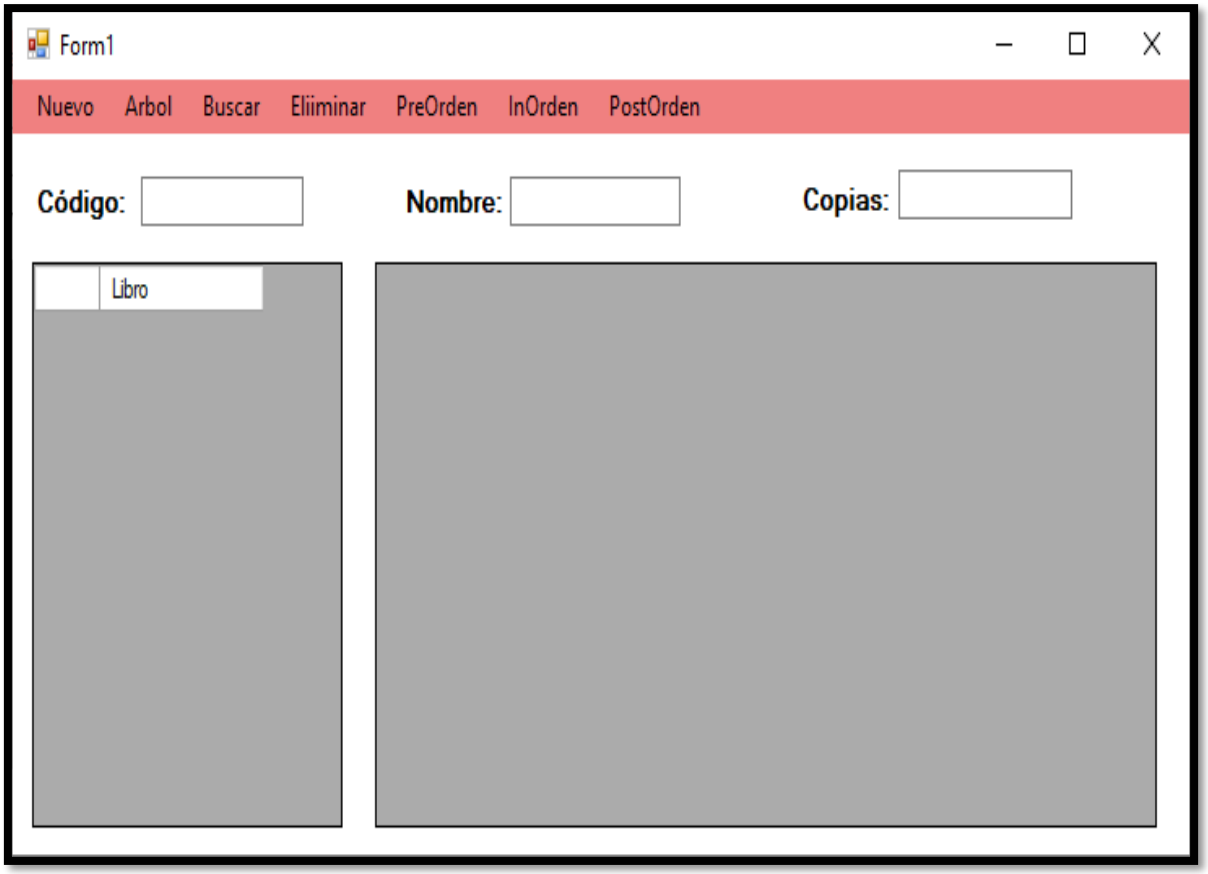
```
            else
            {//izquierdo
                padre.izq = aux.izq;
            }
        }
        else
        {
            if (aux.izq == null)
            {//derecho
                padre.der = aux.der;
            }
            else
            {//izquierdo
                padre.der = aux.izq;
            }
        }
    }

}
```

1 referencia

```
private void doshijos(Nodo padre, Nodo aux, bool eshijoizq)
{
    Nodo nodo = reemplazoderecho(aux);
    if (aux == raiz)
    {
        raiz = nodo;
    }
    else
    {
        if (eshijoizq)
        {
            padre.izq = nodo;
        }
        else
        {
            padre.der = nodo;
        }
    }
    nodo.izq = aux.izq;
}
```

```
// remplazo menor de los mayores
1 referencia
private Nodo reemplazoderecho(Nodo aux)
{
    Nodo padre = aux;
    Nodo auxiliar = aux.der;
    Nodo rem = aux;
    while (auxiliar != null)
    {
        padre = rem;
        rem = auxiliar;
        auxiliar = auxiliar.izq;
    }
    if (rem != aux.der)
    {
        padre.izq = rem.der;
        rem.der = aux.der;
    }
    return rem;
}
```



```
3 referencias
public partial class Form1 : Form
{
    Arbol unArbol = new Arbol();
    1 referencia
    public Form1()
    {
        InitializeComponent();
    }

    1 referencia
    private void TsmNuevo_Click(object sender, EventArgs e)
    {
        unArbol.insertar(ref unArbol.raiz,
            new Libro(TxtCodigo.Text, TxtNombre.Text,
            Convert.ToInt32(TxtCopias.Text)));
    }

    1 referencia
    private void TsmPreOrden_Click(object sender, EventArgs e)
    {
        dgvLista.Rows.Clear();
        unArbol.preOrden(unArbol.raiz, dgvLista);
    }

    1 referencia
    private void TsmInOrden_Click(object sender, EventArgs e)
    {
        dgvLista.Rows.Clear();
        unArbol.inOrden(unArbol.raiz, dgvLista);
    }

    1 referencia
    private void TsmPostOrden_Click(object sender, EventArgs e)
    {
        dgvLista.Rows.Clear();
        unArbol.PostOrden(unArbol.raiz, dgvLista);
    }

    1 referencia
    private void TsmArbol_Click(object sender, EventArgs e)
    {
        Libro[] L = new Libro[5];
        int[] p = new int[5];
        int ce = 0;
        dgvArbol.Columns.Clear();
        unArbol.verArbol(unArbol.raiz, L, p, 0, ref ce);
        for (int i = 0; i < ce; i++) dgvArbol.Columns.Add("","");
        for (int i = 0; i < ce; i++) dgvArbol.Rows.Add();
        for (int i = 0; i < ce; i++)
            dgvArbol[i,p[i]].Value= L[i].codigo + "-"+
                L[i].nombre + "-" + Convert.ToInt32(L[i].copias);

        TxtCodigo.Text = "";
        TxtNombre.Text = "";
        TxtCopias.Text = "";
    }
```

```
1 referencia
private void TsmBuscar_Click(object sender, EventArgs e)
{
    Libro encontrado = unArbol.buscar(TxtCodigo.Text);

    if (encontrado != null)
    {
        MessageBox.Show("Se encontro el empleado: " + encontrado.ToString());
    }
    else
    {
        MessageBox.Show("No se encontro");
    }
}
```

```
1 referencia
private void eliiminarToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (unArbol.eliminar(TxtCopias.Text))
    {
        MessageBox.Show("Se elimino el Nodo correctamente!!!");
    }
    else
    {
        MessageBox.Show("No se pudo eliminar");
    }
}
```