

# CODIGO PARA ARBOLES BINARIOS

S. A. B. Molina Balán

7690-25-4340 Universidad Mariano Gálvez

Matemática discreta

[smolinab@miumg.edu.gt](mailto:smolinab@miumg.edu.gt)

## 2.Resumen

En varios lenguajes de programación se tiene la posibilidad de trabajar con estructuras de datos, desde las simples, hasta las complejas que guardan incluso otras estructuras de datos en sí mismas. Esto conecta las estructuras de datos con la Programación Orientada a Objetos (POO), puesto que, utiliza elementos, crear clases, tipos de datos, manipular sus valores y hacer uso de ellas para representar no solo elementos individualmente, sino también una compleja estructura que, en la práctica, son representativos de casos reales. En el caso de los árboles como estructuras de datos, los mismos árboles son una estructura de datos compleja que almacena valores y las organiza de forma jerárquica según el valor que se obtiene, que se introduce, que se añade o la cual se le asigna manualmente. Sin embargo, la creación de un árbol de datos se puede crear con distintas estructuras.

La estructura con la que se concluyó en usar para un árbol es la estructura. Una estructura de datos valga la redundancia, en C++ funciona como una forma de organizar distintos tipos de datos, siendo ya sea heterogéneas u homogéneas.

El programa creado para la creación de árboles binarios tiene funciones como crear una raíz, donde se crea la base del árbol. Crear hijos a partir de nodos padres que, se eligen ingresando su valor. Recorrer todo el árbol para crear un árbol de forma gráfica en la terminal. Una función para recorrer el árbol y su altura, ordenar sus nodos según sus valores en función de las 3 formas de ordenación de valores: pre, in y post – orden, y averiguar si el árbol creado es uno binario, o si es que cumple con las normas y condiciones para que sea un árbol binario o sea considerado como tal uno.

El programa logra crear satisfactoriamente cada procedimiento con las que se interactúan.

**3. Palabras clave:** estructuras de datos, POO, tipos de datos, árboles, C++, hijos, nodos, binario, valores, padres, raíz, funciones, recursividad, instrucciones, lenguaje de programación.

## 4. Desarrollo del tema:

Los árboles de datos se componen de varias partes según su posición, hijos, nodos y en qué posición estén.

**Hojas:** Las hojas son aquellos nodos que, pueden tener padres, pero no tienen hijos. Son aquellos que componen o representan los últimos nodos que hay dentro del árbol y se encuentran, por lo regular, en lo más bajo del árbol de datos.

**Ramas:** Son nodos que tienen uno o dos hijos. No son nodos finales, puesto que tienen hijos que, durante su lectura u ordenación de sus nodos en los distintos tipos de datos, tienen a un nodo donde pueden trasladarse más debajo de la jerarquía.

**Raíz:** Es el nodo por donde comienza la lectura de nodos. El primer nodo que se encuentra en la jerarquía, por lo regular es el que se encuentra por encima de todos los nodos existentes.

Los tipos de árboles de datos que existen son, por ejemplo:

**Árboles binarios:** Los árboles binarios son aquellos árboles que en cada uno de sus nodos tienen como máximo 2 hijos, uno de cada lado, este tipo de árboles son los más sencillos de leer y de usar recursividad.

**Árboles ternarios:** Son árboles que, a diferencia del binario, tienen como máximo 3 hijos, los cuales se ubican en cada nodo por cada lado y uno en el medio. Estos árboles cumplen con la norma si es que uno de sus nodos cumple con la condición de tener 3 hijos o tener un hijo en medio o en la posición media del nodo como hijo por parte del padre.

**Árboles genéricos:** Los árboles genéricos son árboles que contienen según la voluntad sea de cuántos nodos o hijos tendrá un nodo padre. Es decir, la capacidad máxima de hijos no está predeterminada y pueden variar o simplemente no tenerlos. Por lo que, estos árboles como estructuras de datos no siguen una norma o estructura que ordenan sus datos.

**Ordenamiento de nodos:** El ordenamiento de nodos siguen un algoritmo o instrucciones según el tipo de árbol que sea.

En el programa creado, como únicamente están orientados a trabajar con árboles binarios de búsqueda, entonces el ordenamiento de nodos en un conjunto es tres y siguen los siguientes procedimientos:

**In-order:** La regla para ordenar los nodos es primero recorrer el nodo izquierdo o sub árbol izquierdo, de lo contrario, se procede a escribir o guardar la raíz, o en el nodo en el que se encuentre como tal el recorrido, o de lo contrario, recorrer el nodo derecho o sub árbol derecho. Este orden se le conoce como ordenamiento simétrico.

**Pre-orden:** La regla para ordenar los nodos es primero guardar o leer la raíz en donde se encuentra el proceso de recorrido. De lo contrario, recorre el sub árbol izquierdo, o de lo contrario, recorre el sub árbol derecho. A este ordenamiento se le conoce como recorrido a profundidad.

**Post-orden:** La regla para ordenar los nodos es primero recorrer el sub árbol de la izquierda, de lo contrario, se recorre el sub árbol de la derecha, o de lo contrario, se guarda o lee la raíz en donde se encuentra el recorrido.

### Recursividad

La recursión se considera uno de los conceptos principales en informática. Es un método para resolver problemas, similar a la inducción matemática: para que una función se ejecute, primero se debe obtener su resultado al llamarla con un valor diferente.

Funciona, por ejemplo: la función A de la unidad inicia la función A de dos, esta de tres, y así sucesivamente, hasta que se calcule el valor necesario. Para que la llamada recursiva termine, se debe escribir inmediatamente en la función A una condición de salida de la recursión: por ejemplo, si se obtiene algún valor, no se debe ejecutar la función de nuevo, sino devolver algún resultado.

Los programadores necesitan la recursión de vez en cuando para resolver diferentes problemas.

Hay tareas que son más fáciles e intuitivamente más claras de resolver con su ayuda, y hay otras para las que existen algoritmos óptimos.

Los ejemplos claros de recursión en el programa esta dentro de las funciones de búsqueda inorden, preorden y postorden. Que reutiliza, o la función se llama así mismo para recorrer y cumplir con los recorridos según las instrucciones. Sin embargo, no se trata como tal un bucle, sino, que reutiliza valores que, el mismo parámetro pide o requiere.

Sin embargo, la recursividad en una función como tal si tiene la similitud con un bucle respecto a su repetición. La función, debido a que se llama a sí misma siempre por cada ejecución, se vuelve en un bucle infinito de nunca acabar. Por lo que se establece reglas o condiciones para evitar que el bucle se repita de forma infinita y sobrecargue la memoria. Por lo general, se usa una condición para poder llamar a la función y no llamar a la condición sin esa condición, por lo que, si esa condición es falsa, esta función devolverá un valor a la primera ejecución del principio y no volverá a llamar. Esto provoca que el valor obtenido recaiga en todos los valores obtenidos por los

procesos abiertos y se terminen todas por igual. La cantidad de funciones abiertas o ejecutándose se le considera la profundidad de la recursión.

En el programa se utiliza recursividad en las funciones para mostrar o imprimir el árbol, insertar nodo automáticamente y en las funciones de ordenamiento pre, in y post orden. La característica que comparte cada una de estas funciones es que son funciones que recorren el árbol y buscando nodos.

Se puede observar que cada función realiza un recorrido ya sea para leer el árbol y los nodos que contenga, leer el árbol y ordenar los datos según el ordenamiento que se ejecuta y también leer el árbol para buscar algún nodo que se quiera agregar y que este mismo valor no esté ya integrado en alguna parte del árbol.

Todas estas funciones tienen una condicional que, se ejecuta si encuentra algún nodo que no se haya recorrido o cuando el árbol simplemente no tenga ningún valor o nodo. De lo contrario, estaría devolviendo y abriendo o ejecutando la misma función de forma infinita sin ningún límite. En adición, la función para averiguar si el árbol es binario también usa recursividad, puesto que, recorre todo el árbol ejecutando la misma función hasta que recorre el último nodo, devuelve un valor en booleano y este es devuelta por la función ejecutada desde el menú, dando un resultado que, si es falso, determina que el árbol no es binario, o de lo contrario, determina que el árbol si es binario. O de otra forma, devuelve que el árbol no tiene ningún valor.

## **5. Observaciones y comentarios:**

El programa es funcional para crear árboles binarios, crear la raíz, asignar hijos a un parente según su valor, inserción de valores, búsqueda de nodos o devolver el árbol completo, obtener la altura del árbol, verificar si el árbol es binario y utilizando funciones, sin usar librerías que trabajen directamente con árboles o estructuras similares, usando métodos o el principio de recursividad, además de tener funciones y condicionales que evitan errores lógicos como la duplicación de nodos. El menú cumple con el punto de tener una interfaz interactiva que cumple con las instrucciones que le da el usuario sin realizar solo una ejecución lineal sin interacción. El programa cumple con todo lo requerido, pedido en el documento, documentado, explicación, con un repositorio de GitHub y sus respectivas explicaciones.

## **6. Conclusiones:**

1. Las estructuras de datos que ofrecen varios lenguajes de programación pueden servir para representar un árbol binario. No únicamente puede ser una estructura como tal, sino también, listas en C++, diccionarios en Python, conjuntos genéricos en cualquier lenguaje de programación, entre otros.
2. Las funciones hacen que el programa no siempre tenga que hacer una ejecución lineal sin recursividad. Además, sirven como módulos que esclarecen el programa y su estructura. Los comentarios complementan a las mismas funciones y hacen que las mismas funciones sean más fáciles de localizar y organizar.
3. La recursividad, a pesar de parecer bucles, en realidad son varias funciones que se ejecutan a la vez debido a que, la misma función se llama a sí misma. Como se trata de un árbol, la recursividad funciona como un método para recorrer todos los nodos. Sin embargo, no existe como tal una función o comando para detener este tipo de bucle, por lo que se tiene que crear condicionales para ejecutar la misma u otra función de tercero.
4. El programa cumple con lo requerido, desde la creación del árbol y su estructura, hasta de recorrer el árbol y usar recursividad para determinar la altura, su impresión, búsqueda de nodos, en caso de que se repita y se desea agregar un nodo repetido, entre otros. Además de ordenar los valores de los nodos en los 3 distintos tipos de ordenamiento.

5. La interfaz que tiene, si bien no constituida en una interfaz gráfica de usuario (GUI en inglés, Graphic User Interface), tiene un orden claro y fácil de entender, con únicamente la opción de ingresar o elegir una opción por medio de números enteros.

### **.Bibliografia**

elblogdelprogramador. (6 de Agosto de 2025). *Cómo recorrer árboles binarios: orden in-orden, pre-orden y post-orden*. Obtenido de elblogdelprogramador:

<https://elblogdelprogramador.com/posts/como-recorrer-arboles-binarios-orden-in-order-pre-order-y-post-order/>

Guía de Arbolado. (s.f.). *Recorridos de árboles: preorden, inorden y postorden*. Obtenido de Guía de Arbolado: <https://www.guiadearbolado.com.ar/arboles-preorden-inorden-postorden/>

Hero Vired. (17 de Octubre de 2024). *Tipos de árboles en la estructura de datos: un análisis en profundidad*. Obtenido de herovired: <https://herovired.com/home/learning-hub/topics/types-of-trees-in-data-structure>

## Código

S. A. B. Molina Balán  
7690-25-4340  
Matemática discreta  
[smolinab@miumg.edu.gt](mailto:smolinab@miumg.edu.gt)

```
#include <iostream>

using namespace std;

// ----- Estructura del Nodo -----

struct Nodo {

    int dato;

    Nodo *izq;
    Nodo *der;

};

// ----- Crear un nuevo nodo -----

Nodo *crearNodoNuevo(int valor) {

    Nodo *n = new Nodo();

    n->dato = valor;
    n->izq = NULL;
    n->der = NULL;

    return n;
}

// ----- Insertar nodo automáticamente -----

void insertarAutomatico(Nodo *&raiz, int valor) {

    if (raiz == NULL) {

        raiz = crearNodoNuevo(valor);

        return;
    }

    if (valor == raiz->dato) {

        cout << "El valor ya existe en el arbol. No se agregara.\n";

        return;
    }

    if (valor < raiz->dato)

        insertarAutomatico(raiz->izq, valor);

    else
```

```

insertarAutomatico(raiz->der, valor);
}

//Recorridos

void preorden(Nodo *nodo) {
    if (nodo == NULL) return;
    cout << nodo->dato << " ";
    preorden(nodo->izq);
    preorden(nodo->der);
}

void inorden(Nodo *nodo) {
    if (nodo == NULL) return;
    inorden(nodo->izq);
    cout << nodo->dato << " ";
    inorden(nodo->der);
}

void postorden(Nodo *nodo) {
    if (nodo == NULL) return;
    postorden(nodo->izq);
    postorden(nodo->der);
    cout << nodo->dato << " ";
}

//Calcular la altura

int altura(Nodo *raiz) {
    if (raiz == NULL)
        return 0;
    int izq = altura(raiz->izq);
    int der = altura(raiz->der);
    return (izq > der ? izq : der) + 1;
}

// Imprimir arbol de forma visual ascendente

void mostrarArbol(Nodo *raiz, int nivel = 0) {
    if (raiz == NULL)

```

```

return;

// Primero el subarbol derecho (valores mayores)
mostrarArbol(raiz->der, nivel + 1);

// Imprime el nodo con sangria para representar el nivel
for (int i = 0; i < nivel; i++)
cout << " ";

cout << raiz->dato << endl;

// Luego el subarbol izquierdo (valores menores)
mostrarArbol(raiz->izq, nivel + 1);

}

//Verificar si el árbol es binario válido

bool esBinarioValido(Nodo* raiz, int min, int max) {

if (raiz == NULL)
return true;

if (raiz->dato <= min || raiz->dato >= max)
return false;

return esBinarioValido(raiz->izq, min, raiz->dato) &&
esBinarioValido(raiz->der, raiz->dato, max);
}

//Menu principal

int main() {

Nodo *raiz = NULL;

int opcion, valor;

do {

cout << "\n===== MENU ARBOL BINARIO =====\n";

cout << "1. Crear raiz del arbol\n";
cout << "2. Insertar nodo hijo\n";
cout << "3. Mostrar arbol (ascendente)\n";
cout << "4. Recorridos (Pre, In, Post)\n";
cout << "5. Verificar si es un arbol binario valido\n";
cout << "6. Mostrar altura del arbol\n";
cout << "7. Salir\n";
}

```

```
cout << "Seleccione una opcion: ";

cin >> opcion;

switch (opcion) {

case 1:

if (raiz != NULL)

cout << "La raiz ya fue creada.\n";

else {

cout << "Ingrese el valor de la raiz: ";

cin >> valor;

raiz = crearNodoNuevo(valor);

cout << "Raiz creada correctamente.\n";

}

break;

case 2:

if (raiz == NULL)

cout << "Debe crear la raiz primero.\n";

else {

cout << "Ingrese el valor del nuevo nodo: ";

cin >> valor;

insertarAutomatico(raiz, valor);

cout << "Nodo insertado correctamente.\n";

}

break;

case 3:

if (raiz == NULL)

cout << "El arbol esta vacio.\n";

else {

cout << "\nEstructura del arbol (ascendente):\n";

mostrarArbol(raiz);

}

break;

case 4:
```

```
if (raiz == NULL)
    cout << "El árbol esta vacio.\n";
else {
    cout << "\nPreorden: ";
    preorden(raiz);
    cout << "\nInorden: ";
    inorden(raiz);
    cout << "\nPostorden: ";
    postorden(raiz);
    cout << endl;
}
break;

case 5:
if (raiz == NULL)
    cout << "El arbol esta vacio.\n";
else {
    bool valido = esBinarioValido(raiz, -999999, 999999);
    if (valido)
        cout << "? El arbol es binario de busqueda valido.\n";
    else
        cout << "? El arbol NO cumple las propiedades de un ABB.\n";
}
break;

case 6:
cout << "La altura del arbol es: " << altura(raiz) << endl;
break;

case 7:
cout << "Saliendo del programa... \n";
break;

default:
cout << "Opcion invalida, intente de nuevo.\n";
}
```

```
} while (opcion != 7);

return 0;

}
```

// Capturas:

```
===== MENU ARBOL BINARIO =====
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 6
La altura del arbol es: 4

===== MENU ARBOL BINARIO =====
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 7
}Saliendo del programa...
```

```
===== MENU ARBOL BINARIO =====
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 3

Estructura del arbol (ascendente):
  90
  80
  70
 60
  40
  30
 20
```

```
===== MENU ARBOL BINARIO =====
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 4

Preorden: 60 20 40 30 80 70 90
Inorden: 20 30 40 60 70 80 90
Postorden: 30 40 20 70 90 80 60

===== MENU ARBOL BINARIO =====
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 5
? El arbol es binario de busqueda valido.
```

```
===== MENU ARBOL BINARIO =====
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 2
Ingrese el valor del nuevo nodo: 70
Nodo insertado correctamente.
```

```
===== MENU ARBOL BINARIO =====
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 2
Ingrese el valor del nuevo nodo: 90
Nodo insertado correctamente.
```

```
===== MENU ARBOL BINARIO =====
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 2
Ingrese el valor del nuevo nodo: 30
Nodo insertado correctamente.
```

```
===== MENU ARBOL BINARIO =====
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 2
Ingrese el valor del nuevo nodo: 80
Nodo insertado correctamente.
```

```
===== MENU ARBOL BINARIO =====
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 2
Ingrese el valor del nuevo nodo: 40
Nodo insertado correctamente.
```

```
===== MENU ARBOL BINARIO =====
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 2
Ingrese el valor del nuevo nodo: 20
Nodo insertado correctamente.
```

```
===== MENU ARBOL BINARIO =====
1. Crear raiz del arbol
2. Insertar nodo hijo
3. Mostrar arbol (ascendente)
4. Recorridos (Pre, In, Post)
5. Verificar si es un arbol binario valido
6. Mostrar altura del arbol
7. Salir
Seleccione una opcion: 1
Ingrese el valor de la raiz: 60
Raiz creada correctamente.
```