Data Structure

&

Algorithms

M.Rizwan

Computer Lecturer

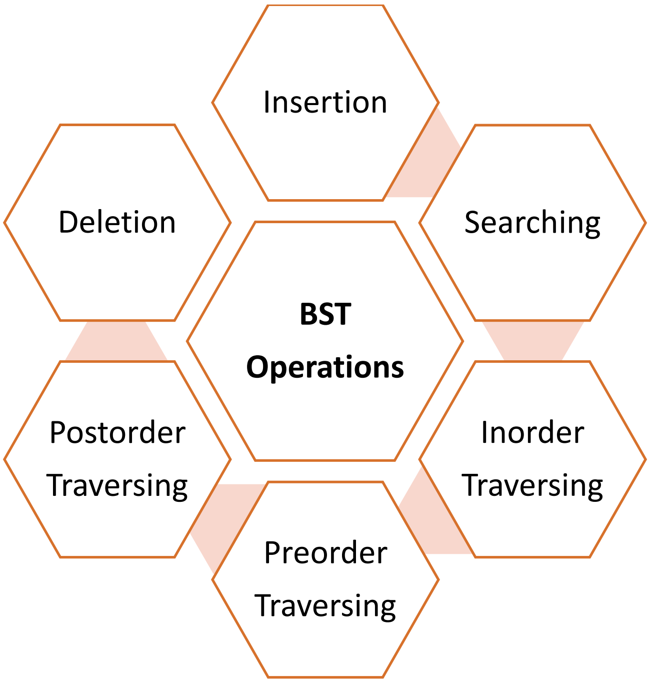
Full Stack .NET Developer/Trainer

Binary Search Tree (BST)

Implementation using **Visual C#**

BST Simple Implementation

BST Recursive Implementation



Insertion

Searching

Inorder Traversing

Preorder Traversing

Postorder Traversing

Deletion

BST Operations

BST

Simple Implementation using Visual C#

BST.cs

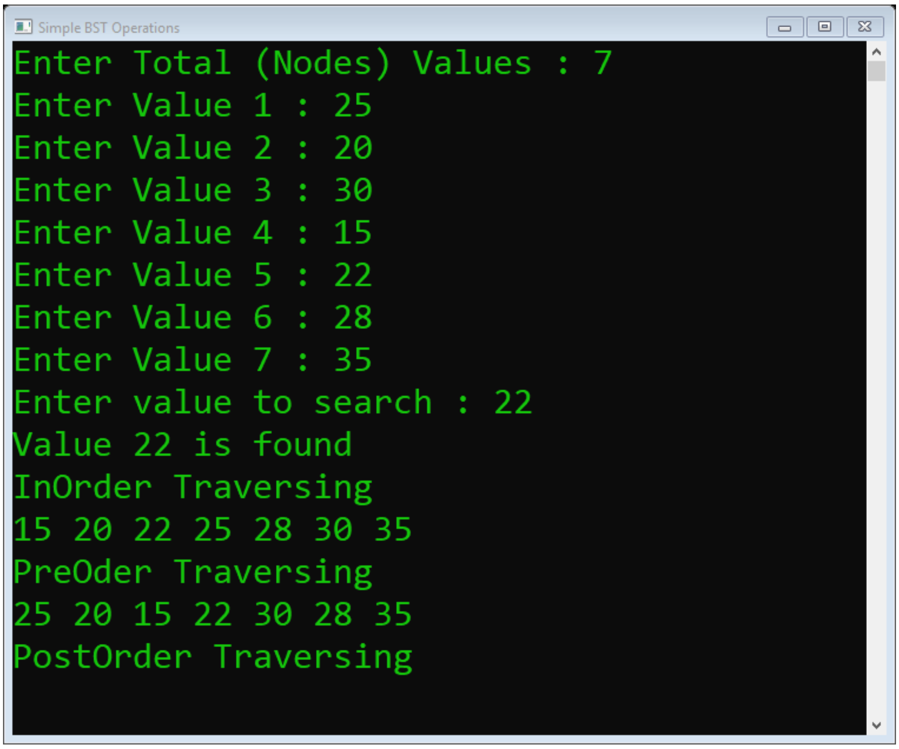
```
using System;
using static System.Console;

namespace Simple_BST_Operations
{
    public class Node
    {
        public int data;
        public Node left;
        public Node right;
    }
    public class BST
    {
        private Node root, temp, current, parent;
        private Node[] stack;
        private int top;
        public BST(int size)
        {
            root = null;
            stack = new Node[size];
            top = -1;
        }
        // Insertion Operation in BST (Simple-Logic ... Not Recursively)
        public void InsertNode(int value)
        {
            // Create node and assign data to it ...
            temp = new Node();
            temp.data = value;
            temp.left = temp.right = null;
            if (root == null)
            {
                root = temp;
                return;
            }
            // Go to proper position to insert node ...
            current = root;
            while (current != null)
            {
                if (value == current.data)
                {
                    WriteLine($"Value {value} is already exist");
                    return;
                }
                if (value < current.data)
                {
                    parent = current;
                    current = current.left;
                }
                else
                {
                    parent = current;
                    current = current.right;
                }
            }

            if (value < parent.data)
                parent.left = temp;
            else
                parent.right = temp;
        }
        // Searching Operation in BST (Simple-Logic ... Not Recursively)
        public void SearchNode(int value)
        {
            if (root == null)
            {
                WriteLine("Tree is Empty");
                return;
            }
            else
            {
                // search value ...
                current = root;
                while (current != null)
                {
                    if (value == current.data)
                    {
                        WriteLine($"Value {value} is found");
                        return;
                    }
                    if (value < current.data)
                        current = current.left;
                    else
                        current = current.right;
                }
            }
            if (current == null)
                WriteLine("Value not found");
        }
        // Traversing Operations in BST (Simple-Logic ... Not Recursively)
        // 1) Inorder Traversal
        public void InOrder()
        {
            if (root == null)
            {
                WriteLine("Tree is Empty");
                return;
            }
            else
            {
                current = root;
                Push(current);
                while (top >= 0)
                {
                    current = stack[top];  // Pop value
                    stack[top] = null;
                    top--;
                    Write($"{current.data} ");
```

```
                    if (current.right != null)
                        Push(current.right);
                }
            }
        }
        public void Push(Node temp)
        {
            while (temp != null)
            {
                top++;
                stack[top] = temp;
                temp = temp.left;
            }
        }
        // 2) Preorder Traversal
        public void PreOrder()
        {
            if (root == null)
            {
                WriteLine("Tree is Empty");
                return;
            }
            else
            {
                top++;
                stack[top] = root;
                while (top >= 0)
                {
                    current = stack[top];
                    top--;
                    while (current != null)
                    {
                        Write($"{current.data} ");
                        if(current.right != null)
                        {
                            top++;
                            stack[top] = current.right;
                        }
                        current = current.left;
                    }
                }
            }
        }
        // 3) Postorder Traversal
        public void PostOrder()
        {
            // Self-Try ...
        }
    }
}
```

Program.cs

```
using System;
using static System.Console;
using static System.Convert;

namespace Simple_BST_Operations
{
    class Program
    {
        static void Main(string[] args)
        {
            Write("Enter Total (Nodes) Values : ");
            int length = ToInt32(ReadLine());
            BST bst = new BST(length);
            for (int i = 0; i < length; i++)
            {
                Write($"Enter Value {i + 1} : ");
                int val = ToInt32(ReadLine());
                bst.InsertNode(val);
            }

            Write("Enter value to search : ");
            int v = ToInt32(ReadLine());
            bst.SearchNode(v);

            WriteLine("InOrder Traversing");
            bst.InOrder();

            WriteLine("\nPreOder Traversing");
            bst.PreOrder();

            WriteLine("\nPostOrder Traversing");
            bst.PostOrder();

            ReadKey();
        }
    }
}
```

Output



```
Enter Total (Nodes) Values : 7
Enter Value 1 : 25
Enter Value 2 : 20
Enter Value 3 : 30
Enter Value 4 : 15
Enter Value 5 : 22
Enter Value 6 : 28
Enter Value 7 : 35
Enter value to search : 22
Value 22 is found
InOrder Traversing
15 20 22 25 28 30 35
PreOder Traversing
25 20 15 22 30 28 35
PostOrder Traversing
```

BST

Recursive Implementation using Visual C#

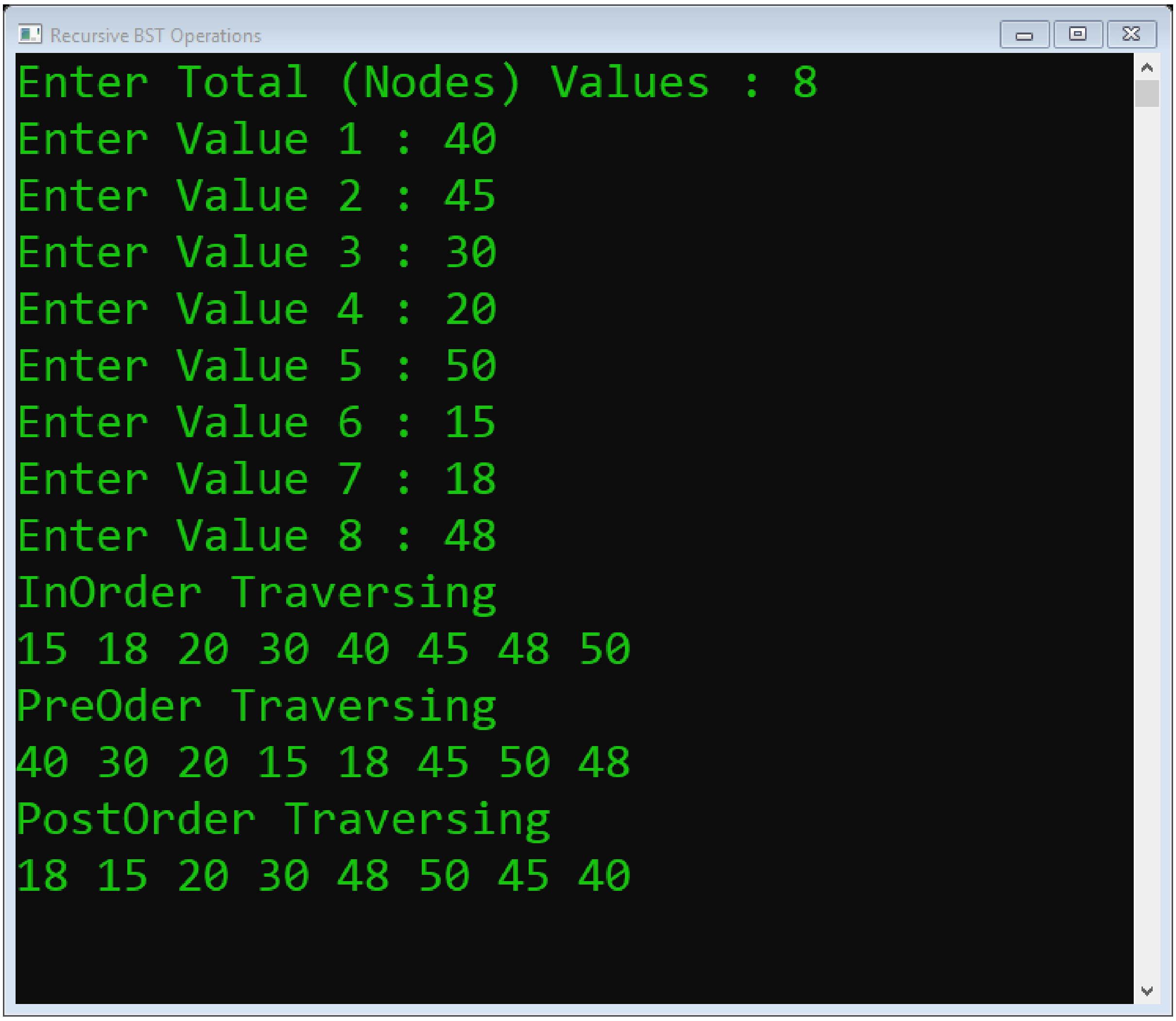
BST.cs

```
using static System.Console;

namespace Recursive_BST_Operations
{
    public class Node
    {
        public int data;
        public Node left, right;
    }
    public class BST
    {
        private Node r, temp;
        public BST() => r = null;

        // Insertion Operation in BST (Recursively)
        public void InsertNode(int v) => r = Insert(r, v);
        private Node Insert(Node root, int value)
        {
            // Create node and assign data to it ...
            temp = new Node();
            temp.data = value;
            temp.left = temp.right = null;

            if (root == null)
                root = temp;
            else if(value == root.data)
            {
                WriteLine($"Value {value} is already exist");
                return root;
            }
            else if (value < root.data)
                root.left = Insert(root.left, value);
            else
                root.right = Insert(root.right, value);
            return root;
        }

        // Searching Operation in BST (Recursively)
        public void SearchNode(int v)
        {
            // Self-Try ...
        }

        // Traversing Operations in BST (Recursively)
        // 1) InOrder Traversal
        public void InOrder() => InOrd(r);
        private void InOrd(Node root)
        {
            if (root == null) return;
            InOrd(root.left);
            Write($"{root.data} ");
            InOrd(root.right);
        }
        // 2) PreOrder Traversal
        public void PreOrder() => PreOrd(r);
        private void PreOrd(Node root)
        {
            if (root == null) return;
            Write($"{root.data} ");
            PreOrd(root.left);
            PreOrd(root.right);
        }
        // 3) PostOrder Traversal
        public void PostOrder() => PostOrd(r);
        private void PostOrd(Node root)
        {
            if (root == null) return;
            PostOrd(root.left);
            PostOrd(root.right);
            Write($"{root.data} ");
        }
    }
}
```

Program.cs

```
using System;
using static System.Console;
using static System.Convert;

namespace Recursive_BST_Operations
{
    class Program
    {
        static void Main(string[] args)
        {
            BST bst = new BST();

            Write("Enter Total (Nodes) Values : ");
            int length = ToInt32(ReadLine());
            for (int i = 0; i < length; i++)
            {
                Write($"Enter Value {i + 1} : ");
                int val = ToInt32(ReadLine());
                bst.InsertNode(val);
            }

            //Write("Enter value to search : ");
            //int v = ToInt32(ReadLine());
            //bst.SearchNode(v);

            WriteLine("InOrder Traversing");
            bst.InOrder();

            WriteLine("\nPreOder Traversing");
            bst.PreOrder();

            WriteLine("\nPostOrder Traversing");
            bst.PostOrder();

            ReadKey(true);
        }
    }
}
```

Output



```
Enter Total (Nodes) Values : 8
Enter Value 1 : 40
Enter Value 2 : 45
Enter Value 3 : 30
Enter Value 4 : 20
Enter Value 5 : 50
Enter Value 6 : 15
Enter Value 7 : 18
Enter Value 8 : 48
InOrder Traversing
15 18 20 30 40 45 48 50
PreOder Traversing
40 30 20 15 18 45 50 48
PostOrder Traversing
18 15 20 30 48 50 45 40
```

https://github.com/MRizwanSE/BST-VisualCSharp

Best of Luck 😊