**Muhammad Abdullah**

**SE(3A) | 19F-0916**

Data Structures Lab

Lab # 7 BST

**Question # 1:**

**PROGRAM**

#include <iostream>

using namespace std;

struct Node // Node Construction

{

int Data;

Node \*Left\_Node;

Node \*Right\_Node;

};

class Bst\_Tree // Class for Tree Formation

{

public:

Bst\_Tree()

{

Root = NULL;

}

Node \*Root;

Node \*Insertion(Node \*Current, int data) // Insertion of Nodes

{

if (Current == NULL)

{

Current = new Node;

Current->Data = data;

Current->Left\_Node = NULL;

Current->Right\_Node = NULL;

//Root = Current;

return Current;

}

else if (data < Current->Data) // If Data is lesser than Root

{

Current->Left\_Node = Insertion(Current->Left\_Node, data);

}

else if (data >= Current->Data) // If Data is greater than Root

{

Current->Right\_Node = Insertion(Current->Right\_Node, data);

}

return Current;

}

void Max\_root() // Showing Maximum value

{

if (Root != NULL)

{

Node \*temp = Root;

while (temp->Right\_Node != NULL)

{

temp = temp->Right\_Node;

}

cout << endl << " Max Node is : " << temp->Data << endl;

}

else

cout << "Tree is Empty !!" << endl;

}

void Min\_root() //Showing Minimum value

{

if (Root != NULL)

{

Node \*temp = Root;

while (temp->Left\_Node != NULL)

{

temp = temp->Left\_Node;

}

cout << endl << " Min Node is : " << temp->Data << endl;

}

else

cout << "Tree is Empty !!" << endl;

}

};

int main()

{

Bst\_Tree Tree;

int choice = 0, value = 0, i = 1;

while (i != 0)

{

system("cls");

cout << "=================================================" << endl;

cout << " Press 1 to Enter a Node !" << endl;

cout << " Press 2 to Check Maximum Value of Node !" << endl;

cout << " Press 3 to Check Minimum Value of Node !" << endl;

cout << " Press 0 to Exit through System !" << endl;

cout << "=================================================" << endl;

cout << " Enter Choice : ";

cin >> choice;

switch (choice)

{

case 1:

{

cout << endl << "Enter Value for the Node :";

cin >> value;

Tree.Root = Tree.Insertion(Tree.Root, value);

system("pause");

break;

}

case 2:

{

cout << endl;

Tree.Max\_root();

cout << endl;

system("pause");

break;

}

case 3:

{

cout << endl;

Tree.Min\_root();

cout << endl;

system("pause");

break;

}

case 0:

{

cout << endl << "You have exited from the system !" << endl;

i = 0;

break;

}

default:

cout << endl << "Invalid Entry !" << endl;

}

}

cout << endl << endl;

system("pause");

}

**MAXA picture containing text, monitor, screenshot, indoor

Description automatically generated**

**MINA picture containing text, monitor, screenshot, indoor

Description automatically generated**

**Question # 2:**

**PROGRAM**

#include <iostream>

using namespace std;

struct Node // Node Construction

{

int Data;

Node \*Left\_Node;

Node \*Right\_Node;

};

class Bst\_Tree // Class for Tree Formation

{

public:

Bst\_Tree()

{

Root = NULL;

}

Node \*Root;

Node \*Insertion(Node \*Current, int data) // Insertion of Nodes

{

if (Current == NULL)

{

Current = new Node;

Current->Data = data;

Current->Left\_Node = NULL;

Current->Right\_Node = NULL;

return Current;

}

else if (data < Current->Data) // If Data is lesser than Root

{

Current->Left\_Node = Insertion(Current->Left\_Node, data);

}

else if (data >= Current->Data) // If Data is greater than Root

{

Current->Right\_Node = Insertion(Current->Right\_Node, data);

}

return Current;

}

int Count\_Leaf(Node \*Temp) // Counting Leaf Nodes

{

if (Temp == NULL)

return 0;

if (Temp->Left\_Node == NULL && Temp->Right\_Node == NULL)

return 1;

else

return Count\_Leaf(Temp->Left\_Node) + Count\_Leaf(Temp->Right\_Node);

}

};

int main()

{

Bst\_Tree Tree;

int choice = 0, value = 0, i = 1;

while (i != 0)

{

system("cls");

cout << "=================================================" << endl;

cout << " Press 1 to Enter a Node !" << endl;

cout << " Press 2 to Check Total Numbers of Leaf Nodes !" << endl;

cout << " Press 0 To Exit From The System !" << endl;

cout << "=================================================" << endl;

cout << " Enter Choice : ";

cin >> choice;

switch (choice)

{

case 1:

{

cout << endl << "Enter Value for the Node :";

cin >> value;

Tree.Root = Tree.Insertion(Tree.Root, value);

system("pause");

break;

}

case 2:

{

cout << endl;

value = 0;

value = Tree.Count\_Leaf(Tree.Root);

cout << "Total Number of Leaf Nodes Are : " << value << endl;

system("pause");

break;

}

case 0:

{

cout << endl << "You have exited from the system !" << endl;

i = 0;

break;

}

default:

cout << endl << "Invalid Entry !" << endl;

}

}

cout << endl << endl;

system("pause");

}

**A picture containing text, screenshot, electronics, monitor

Description automatically generated**

**Question # 3:**

**PROGRAM**

#include <iostream>

using namespace std;

struct Node // Node Construction

{

int Data;

Node \*Left\_Node;

Node \*Right\_Node;

};

class Bst\_Tree // Class for Tree Formation

{

public:

Bst\_Tree()

{

Root = NULL;

}

Node \*Root;

Node \*Insertion(Node \*Current, int data) // Insertion of Nodes

{

if (Current == NULL)

{

Current = new Node;

Current->Data = data;

Current->Left\_Node = NULL;

Current->Right\_Node = NULL;

return Current;

}

else if (data < Current->Data) // If Data is lesser than Root

{

Current->Left\_Node = Insertion(Current->Left\_Node, data);

}

else if (data >= Current->Data) // If Data is greater than Root

{

Current->Right\_Node = Insertion(Current->Right\_Node, data);

}

return Current;

}

void Delete\_Node(Node \*root, int Key)

{

Node \*temp = root;

if (temp == NULL)

{

cout << endl << "Tree is Empty !" << endl;

return;

}

else if (temp->Left\_Node->Data == Key && temp->Left\_Node != NULL)

{

Node \*temp1 = temp->Left\_Node;

cout << endl << "Required Node is Deleted !" << endl;

temp->Left\_Node = NULL;

free(temp1);

return;

}

else if (temp->Right\_Node->Data == Key && temp->Right\_Node != NULL)

{

Node \*temp1 = temp->Right\_Node;

cout << endl << "Required Node is Deleted !" << endl;

temp->Right\_Node = NULL;

free(temp1);

return;

}

else if (temp->Left\_Node == NULL && temp->Right\_Node == NULL)

{

if (Key == temp->Data)

{

Node \*temp1 = Root;

cout << endl << "Required Node is Deleted !" << endl;

Root = NULL;

free(temp1);

return;

}

else

cout << endl << "There is NO Such Node in the Tree !" << endl;

}

else

{

Delete\_Node(temp->Left\_Node, Key);

if (Key == temp->Data)

{

Node \*temp1 = temp;

cout << endl << "Required Node is Deleted !" << endl;

Root = NULL;

free(temp1);

return;

}

Delete\_Node(temp->Right\_Node, Key);

}

}

};

int main()

{

Bst\_Tree Tree;

int choice = 0, value = 0, i = 1;

while (i != 0)

{

system("cls");

cout << "=================================================" << endl;

cout << " Press 1 to Enter a Node !" << endl;

cout << " Press 2 to Delete Any Node From Tree !" << endl;

cout << " Press 0 To Exit From The System !" << endl;

cout << "=================================================" << endl;

cout << " Enter Choice : ";

cin >> choice;

switch (choice)

{

case 1:

{

cout << endl << "Enter Value for the Node :";

cin >> value;

Tree.Root = Tree.Insertion(Tree.Root, value);

system("pause");

break;

}

case 2:

{

cout << endl << "Enter Value to Delete Respective Node :";

cin >> value;

Tree.Delete\_Node(Tree.Root, value);

system("pause");

break;

}

case 0:

{

cout << endl << "You have exited from the system !" << endl;

i = 0;

break;

}

default:

cout << endl << "Invalid Entry !" << endl;

}

}

cout << endl << endl;

system("pause");

}

**A picture containing text, monitor, screenshot, electronics

Description automatically generated**

**Question # 4:**

**PROGRAM**

#include <iostream>

using namespace std; // Not Sure About Splittion, Tried but....

struct Node // Node Construction

{

int Data;

Node \*Left\_Node;

Node \*Right\_Node;

};

class Bst\_Tree // Class for Tree Formation

{

public:

Bst\_Tree()

{

Root = NULL;

}

Node \*Root;

Node \*Insertion(Node \*Current, int data) // Insertion of Nodes

{

if (Current == NULL)

{

Current = new Node;

Current->Data = data;

Current->Left\_Node = NULL;

Current->Right\_Node = NULL;

return Current;

}

else if (data < Current->Data) // If Data is lesser than Root

{

Current->Left\_Node = Insertion(Current->Left\_Node, data);

}

else if (data >= Current->Data) // If Data is greater than Root

{

Current->Right\_Node = Insertion(Current->Right\_Node, data);

}

return Current;

}

void Nodes\_Height(Node \* temp)

{

if (temp == NULL)

return;

Nodes\_Height(temp->Left\_Node);

cout << " " << temp->Data;

Nodes\_Height(temp->Right\_Node);

}

};

int main()

{

Bst\_Tree Tree;

int choice = 0, value = 0, i = 1;

while (i != 0)

{

system("cls");

cout << "=================================================" << endl;

cout << " Press 1 to Enter a Value in Tree !" << endl << endl;

cout << " Press 2 to Check Nodes at Specific Height !" << endl << endl;

cout << " Press 0 To Exit From The System !" << endl;

cout << "=================================================" << endl;

cout << " Enter Choice : ";

cin >> choice;

switch (choice)

{

case 1:

{

cout << endl << "Enter Value to Enter it in Tree : ";

cin >> value;

Tree.Root = Tree.Insertion(Tree.Root, value);

system("pause");

break;

}

case 2:

{

cout << endl << "Enter Height : ";

cin >> value;

cout << endl << "Nodes at " << value << " Height are : ";

Tree.Nodes\_Height(Tree.Root);

cout << endl;

system("pause");

break;

}

case 0:

{

cout << endl << "You have exited from the system !" << endl;

i = 0;

break;

}

default:

cout << endl << "Invalid Entry !" << endl;

system("pause");

break;

}

}

cout << endl << endl;

system("pause");

}

**A picture containing text, monitor, indoor, electronics

Description automatically generated**

**Question # 5:**

**PROGRAM**

#include <iostream>

using namespace std;

struct Node // Node Construction

{

int Data;

Node \*Left\_Node;

Node \*Right\_Node;

};

class Bst\_Tree // Class for Tree Formation

{

public:

Bst\_Tree()

{

Root = NULL;

}

Node \*Root;

Node \*Insertion(Node \*Current, int data) // Insertion of Nodes

{

if (Current == NULL)

{

Current = new Node;

Current->Data = data;

Current->Left\_Node = NULL;

Current->Right\_Node = NULL;

return Current;

}

else if (data < Current->Data) // If Data is lesser than Root

{

Current->Left\_Node = Insertion(Current->Left\_Node, data);

}

else if (data >= Current->Data) // If Data is greater than Root

{

Current->Right\_Node = Insertion(Current->Right\_Node, data);

}

return Current;

}

void In\_Order(Node \*C) // In order traversal

{

if (Root == NULL)

{

cout << endl << "Tree is Empty !" << endl;

return;

}

else

{

if (C == NULL)

return;

In\_Order(C->Left\_Node);

cout << " " << C->Data;

In\_Order(C->Right\_Node);

}

}

void Pre\_Order(Node \*C) // Pre Order traversal

{

if (Root == NULL)

{

cout << endl << "Tree is Empty !" << endl;

return;

}

else

{

Node \*temp = C;

if (temp == NULL)

return;

cout << " " << C->Data;

Pre\_Order(temp->Left\_Node);

Pre\_Order(temp->Right\_Node);

}

}

void Post\_Order(Node \*C) // Post Order traversal

{

if (Root == NULL)

{

cout << endl << "Tree is Empty !" << endl;

return;

}

else

{

Node \*temp = C;

if (temp == NULL)

return;

Post\_Order(temp->Left\_Node);

Post\_Order(temp->Right\_Node);

cout << " " << C->Data;

}

}

};

int main()

{

Bst\_Tree Tree;

int choice = 0, value = 0, i = 1;

while (i != 0)

{

system("cls");

cout << "=================================================" << endl;

cout << " Press 1 to Enter a Node !" << endl;

cout << " Press 2 to Perform In-Order Traversal !" << endl;

cout << " Press 3 to Perform Pre-Order Traversal !" << endl;

cout << " Press 4 to Perform Post-Order Traversal !" << endl;

cout << " Press 0 To Exit From The System !" << endl;

cout << "=================================================" << endl;

cout << " Enter Choice : ";

cin >> choice;

switch (choice)

{

case 1:

{

cout << endl << "Enter Value for the Node :";

cin >> value;

Tree.Root = Tree.Insertion(Tree.Root, value);

system("pause");

break;

}

case 2:

{

cout << endl << "In-Order Traversal : ";

Tree.In\_Order(Tree.Root);

cout << endl;

system("pause");

break;

}

case 3:

{

cout << endl<<"Pre-Order Traversal : ";

Tree.Pre\_Order(Tree.Root);

cout << endl;

system("pause");

break;

}

case 4:

{

cout << endl << "Post-Order Traversal : ";

Tree.Post\_Order(Tree.Root);

cout << endl;

system("pause");

break;

}

case 0:

{

cout << endl << "You have exited from the system !" << endl;

i = 0;

break;

}

default:

cout << endl << "Invalid Entry !" << endl;

system("pause");

break;

}

}

cout << endl << endl;

system("pause");

}

**IN ORDER TRAVERSAL**

**A picture containing text, monitor, electronics, indoor

Description automatically generated**

**Pre ORDER TRAVERSAL**

**A picture containing text, monitor, electronics, screenshot

Description automatically generated**

**Post ORDER TRAVERSAL**

**A picture containing text, monitor, electronics, screenshot

Description automatically generated**

**Question # 6:**

**PROGRAM**

#include <iostream>

using namespace std; // Not Sure About Splittion, Tried but....

struct Node // Node Construction

{

int Data;

Node \*Left\_Node;

Node \*Right\_Node;

};

class Bst\_Tree // Class for Tree Formation

{

public:

Bst\_Tree()

{

Root = NULL;

}

Node \*Root;

Node \*Insertion(Node \*Current, int data) // Insertion of Nodes

{

if (Current == NULL)

{

Current = new Node;

Current->Data = data;

Current->Left\_Node = NULL;

Current->Right\_Node = NULL;

return Current;

}

else if (data < Current->Data) // If Data is lesser than Root

{

Current->Left\_Node = Insertion(Current->Left\_Node, data);

}

else if (data >= Current->Data) // If Data is greater than Root

{

Current->Right\_Node = Insertion(Current->Right\_Node, data);

}

return Current;

}

void In\_Order(Node \*C) // In order traversal

{

if (Root == NULL)

{

cout << endl << "Tree is Empty !" << endl;

return;

}

else

{

if (C == NULL)

return;

In\_Order(C->Left\_Node);

cout << " " << C->Data;

In\_Order(C->Right\_Node);

}

}

void Pre\_Order(Node \*C) // Pre Order traversal

{

if (Root == NULL)

{

cout << endl << "Tree is Empty !" << endl;

return;

}

else

{

Node \*temp = C;

if (temp == NULL)

return;

cout << " " << C->Data;

Pre\_Order(temp->Left\_Node);

Pre\_Order(temp->Right\_Node);

}

}

void Post\_Order(Node \*C) // Post Order traversal

{

if (Root == NULL)

{

cout << endl << "Tree is Empty !" << endl;

return;

}

else

{

Node \*temp = C;

if (temp == NULL)

return;

Post\_Order(temp->Left\_Node);

Post\_Order(temp->Right\_Node);

cout << " " << C->Data;

}

}

int Height\_of\_Tree(Node \*Current) // Checking Height of the Tree

{

int Height = 0;

if (Current != NULL)

{

int Left\_SubTree = Height\_of\_Tree(Current->Left\_Node);

int Right\_SubTree = Height\_of\_Tree(Current->Right\_Node);

if (Left\_SubTree >= Right\_SubTree)

return (Left\_SubTree + 1); // Returning Left Sub Tree if Greater

else

return (Right\_SubTree + 1); // Returning Right Sub Tree if Greater

}

else

return 0;

}

int Difference(Node \*Current) // Checking Difference of Left and Right Sub Tree

{

if (Root != NULL)

{

int Left\_SubTree = Height\_of\_Tree(Current->Left\_Node);

int Right\_SubTree = Height\_of\_Tree(Current->Right\_Node);

int Balance\_Factor = Left\_SubTree - Right\_SubTree;

return Balance\_Factor; // Returing BF

}

else

cout << endl << "Tree is Empty !" << endl;

}

Node \*Balance\_Tree(Node \*Current) // Checking Balance of the Tree

{

if (Root != NULL)

{

int Balance\_Factor = Difference(Current);

if (Balance\_Factor == -1 || Balance\_Factor == 0 || Balance\_Factor == 1)

{

cout << endl << "Yes, Tree is Balanced !" << endl;

}

else

{

cout << endl << "No, Tree is not Balanced !" << endl;

}

}

else

cout << endl << "Tree is Empty !" << endl;

return Current;

}

};

int main()

{

Bst\_Tree Tree;

int choice = 0, value = 0, i = 1;

while (i != 0)

{

system("cls");

cout << "=================================================" << endl;

cout << " Press 1 to Enter a Value in Tree !" << endl << endl;

cout << " Press 2 to Perform In-Order Traversal !" << endl;

cout << " Press 3 to Perform Pre-Order Traversal !" << endl;

cout << " Press 4 to Perform Post-Order Traversal !" << endl << endl;

cout << " Press 5 to Check Balance of Tree !" << endl;

cout << " Press 0 To Exit From The System !" << endl;

cout << "=================================================" << endl;

cout << " Enter Choice : ";

cin >> choice;

switch (choice)

{

case 1:

{

cout << endl << "Enter Value to Enter it in Tree : ";

cin >> value;

Tree.Root = Tree.Insertion(Tree.Root, value);

system("pause");

break;

}

case 2:

{

cout << endl << "In-Order Traversal : ";

Tree.In\_Order(Tree.Root);

cout << endl;

system("pause");

break;

}

case 3:

{

cout << endl<<"Pre-Order Traversal : ";

Tree.Pre\_Order(Tree.Root);

cout << endl;

system("pause");

break;

}

case 4:

{

cout << endl << "Post-Order Traversal : ";

Tree.Post\_Order(Tree.Root);

cout << endl;

system("pause");

break;

}

case 5:

{

cout << endl;

Tree.Balance\_Tree(Tree.Root);

cout << endl;

system("pause");

break;

}

case 0:

{

cout << endl << "You have exited from the system !" << endl;

i = 0;

break;

}

default:

cout << endl << "Invalid Entry !" << endl;

system("pause");

break;

}

}

cout << endl << endl;

system("pause");

}

**A picture containing text, monitor, electronics, screenshot

Description automatically generated**

**Question # 7:**

**PROGRAM**

#include <iostream>

using namespace std;

struct Node // Node Construction

{

int Data;

Node \*Left\_Node;

Node \*Right\_Node;

};

class Bst\_Tree // Class for Tree Formation

{

public:

Bst\_Tree()

{

Root = NULL;

}

Node \*Root;

Node \*Insertion(Node \*Current, int data) // Insertion of Nodes

{

if (Current == NULL)

{

Current = new Node;

Current->Data = data;

Current->Left\_Node = NULL;

Current->Right\_Node = NULL;

return Current;

}

else if (data < Current->Data) // If Data is lesser than Root

{

Current->Left\_Node = Insertion(Current->Left\_Node, data);

}

else if (data >= Current->Data) // If Data is greater than Root

{

Current->Right\_Node = Insertion(Current->Right\_Node, data);

}

return Current;

}

void In\_Order(Node \*C) // In order traversal

{

if (Root == NULL)

{

cout << endl << "Tree is Empty !" << endl;

return;

}

else

{

if (C == NULL)

return;

In\_Order(C->Left\_Node);

cout << " " << C->Data;

In\_Order(C->Right\_Node);

}

}

void Pre\_Order(Node \*C) // Pre Order traversal

{

if (Root == NULL)

{

cout << endl << "Tree is Empty !" << endl;

return;

}

else

{

Node \*temp = C;

if (temp == NULL)

return;

cout << " " << C->Data;

Pre\_Order(temp->Left\_Node);

Pre\_Order(temp->Right\_Node);

}

}

void Post\_Order(Node \*C) // Post Order traversal

{

if (Root == NULL)

{

cout << endl << "Tree is Empty !" << endl;

return;

}

else

{

Node \*temp = C;

if (temp == NULL)

return;

Post\_Order(temp->Left\_Node);

Post\_Order(temp->Right\_Node);

cout << " " << C->Data;

}

}

};

int main()

{

Bst\_Tree Tree;

int choice = 0, value = 0, i = 1;

while (i != 0)

{

system("cls");

cout << "=================================================" << endl;

cout << " Press 1 to Enter a Values Based on Array !" << endl << endl;

cout << " Press 2 to Perform In-Order Traversal !" << endl;

cout << " Press 3 to Perform Pre-Order Traversal !" << endl;

cout << " Press 4 to Perform Post-Order Traversal !" << endl << endl;

cout << " Press 0 To Exit From The System !" << endl;

cout << "=================================================" << endl;

cout << " Enter Choice : ";

cin >> choice;

switch (choice)

{

case 1: // Array Based Implementation

{

int Aray[10] = { 1,3,5,7,9,2,4,6,8,10 };

for (int i = 0; i < 10; i++)

{

Tree.Root = Tree.Insertion(Tree.Root, Aray[i]);

}

cout << endl << "Values Added from Array !" << endl;

system("pause");

break;

}

case 2:

{

cout << endl << "In-Order Traversal : ";

Tree.In\_Order(Tree.Root);

cout << endl;

system("pause");

break;

}

case 3:

{

cout << endl<<"Pre-Order Traversal : ";

Tree.Pre\_Order(Tree.Root);

cout << endl;

system("pause");

break;

}

case 4:

{

cout << endl << "Post-Order Traversal : ";

Tree.Post\_Order(Tree.Root);

cout << endl;

system("pause");

break;

}

case 0:

{

cout << endl << "You have exited from the system !" << endl;

i = 0;

break;

}

default:

cout << endl << "Invalid Entry !" << endl;

system("pause");

break;

}

}

cout << endl << endl;

system("pause");

}

**INSERTION**

**A picture containing text, electronics, monitor, indoor

Description automatically generated**

**SHOWING VALUES BY IN ORDER TRAVERSAL**

**A picture containing text, electronics, monitor, screenshot

Description automatically generated**

**Question # 8:**

**PROGRAM**

**I am not able to understand what is required in the question. Sorry for this!**

**Question # 9:**

**PROGRAM**

#include <iostream>

using namespace std; // Not Sure About Splittion, Tried but....

struct Node // Node Construction

{

int Data;

Node \*Left\_Node;

Node \*Right\_Node;

};

class Bst\_Tree // Class for Tree Formation

{

public:

Bst\_Tree()

{

Root = NULL;

}

Node \*Root;

Node \*Insertion(Node \*Current, int data) // Insertion of Nodes

{

if (Current == NULL)

{

Current = new Node;

Current->Data = data;

Current->Left\_Node = NULL;

Current->Right\_Node = NULL;

return Current;

}

else if (data < Current->Data) // If Data is lesser than Root

{

Current->Left\_Node = Insertion(Current->Left\_Node, data);

}

else if (data >= Current->Data) // If Data is greater than Root

{

Current->Right\_Node = Insertion(Current->Right\_Node, data);

}

return Current;

}

void In\_Order(Node \*C) // In order traversal

{

if (Root == NULL)

{

cout << endl << "Tree is Empty !" << endl;

return;

}

else

{

if (C == NULL)

return;

In\_Order(C->Left\_Node);

cout << " " << C->Data;

In\_Order(C->Right\_Node);

}

}

void Pre\_Order(Node \*C) // Pre Order traversal

{

if (Root == NULL)

{

cout << endl << "Tree is Empty !" << endl;

return;

}

else

{

Node \*temp = C;

if (temp == NULL)

return;

cout << " " << C->Data;

Pre\_Order(temp->Left\_Node);

Pre\_Order(temp->Right\_Node);

}

}

void Post\_Order(Node \*C) // Post Order traversal

{

if (Root == NULL)

{

cout << endl << "Tree is Empty !" << endl;

return;

}

else

{

Node \*temp = C;

if (temp == NULL)

return;

Post\_Order(temp->Left\_Node);

Post\_Order(temp->Right\_Node);

cout << " " << C->Data;

}

}

int BST\_OR\_NOT(Node\* root, Node\* Left = NULL, Node\* Right = NULL)

{

if (root == NULL)

return 1;

if (Left != NULL && root->Data <= Left->Data) // Checking Left Data

return 0;

if (Right != NULL && root->Data >= Right->Data) // Checking Right Data

return 0;

return BST\_OR\_NOT(root->Left\_Node, Left, root) && BST\_OR\_NOT(root->Right\_Node, root, Right); // Traversing in Tree

}

};

int main()

{

Bst\_Tree Tree;

int choice = 0, value = 0, i = 1;

while (i != 0)

{

system("cls");

cout << "=================================================" << endl;

cout << " Press 1 to Enter a Value in Tree !" << endl << endl;

cout << " Press 2 to Perform In-Order Traversal !" << endl;

cout << " Press 3 to Perform Pre-Order Traversal !" << endl;

cout << " Press 4 to Perform Post-Order Traversal !" << endl << endl;

cout << " Press 5 to Check That Tree is BST or NOT !" << endl;

cout << " Press 0 To Exit From The System !" << endl;

cout << "=================================================" << endl;

cout << " Enter Choice : ";

cin >> choice;

switch (choice)

{

case 1:

{

cout << endl << "Enter Value to Enter it in Tree : ";

cin >> value;

Tree.Root = Tree.Insertion(Tree.Root, value);

system("pause");

break;

}

case 2:

{

cout << endl << "In-Order Traversal : ";

Tree.In\_Order(Tree.Root);

cout << endl;

system("pause");

break;

}

case 3:

{

cout << endl<<"Pre-Order Traversal : ";

Tree.Pre\_Order(Tree.Root);

cout << endl;

system("pause");

break;

}

case 4:

{

cout << endl << "Post-Order Traversal : ";

Tree.Post\_Order(Tree.Root);

cout << endl;

system("pause");

break;

}

case 5:

{

cout << endl;

value = Tree.BST\_OR\_NOT(Tree.Root);

if (value == 1)

cout << endl << "Tree is Binary Search Tree !" << endl;

else

cout << endl << "Tree is Not Binary Search Tree !" << endl;

cout << endl;

system("pause");

break;

}

case 0:

{

cout << endl << "You have exited from the system !" << endl;

i = 0;

break;

}

default:

cout << endl << "Invalid Entry !" << endl;

system("pause");

break;

}

}

cout << endl << endl;

system("pause");

}

**INORDER TRAVERSAL OF TREE TO SHOW VALUES**

**A picture containing text, electronics, screenshot, computer

Description automatically generated**

**CHECKING IF IT IS BST OR NOT**

**A picture containing text, electronics, screenshot, computer

Description automatically generated**

**Question # 10:**

**PROGRAM**

#include <iostream>

using namespace std; // Not Sure About Splittion, Tried but....

struct Node // Node Construction

{

int Data;

Node \*Left\_Node;

Node \*Right\_Node;

};

class Bst\_Tree // Class for Tree Formation

{

public:

Bst\_Tree()

{

Root = NULL;

}

Node \*Root;

Node \*Insertion(Node \*Current, int data) // Insertion of Nodes

{

if (Current == NULL)

{

Current = new Node;

Current->Data = data;

Current->Left\_Node = NULL;

Current->Right\_Node = NULL;

return Current;

}

else if (data < Current->Data) // If Data is lesser than Root

{

Current->Left\_Node = Insertion(Current->Left\_Node, data);

}

else if (data >= Current->Data) // If Data is greater than Root

{

Current->Right\_Node = Insertion(Current->Right\_Node, data);

}

return Current;

}

void In\_Order(Node\* temp)

{

if (temp == NULL)

return;

In\_Order(temp->Left\_Node);

In\_Order(temp->Right\_Node);

cout << " " << temp->Data;

}

};

int main()

{

Bst\_Tree Tree;

int choice = 0, value = 0, i = 1;

while (i != 0)

{

system("cls");

cout << "=================================================" << endl;

cout << " Press 1 to Enter Values in Tree by an Array !" << endl << endl;

cout << " Press 2 to Check Height of the Tree for Monk !" << endl << endl;

cout << " Press 0 To Exit From The System !" << endl;

cout << "=================================================" << endl;

cout << " Enter Choice : ";

cin >> choice;

switch (choice)

{

case 1:

{

int Aray[10] = { 7,3,8,2,9,5,4,1,6,10 }; // Array Based Values Input

for (int i = 0; i < 10; i++)

{

Tree.Root = Tree.Insertion(Tree.Root, Aray[i]);

}

cout << endl << "Array Based Values Are Inputted !" << endl;

system("pause");

break;

}

case 2:

{

cout << endl << "Enter Key to Reverse Tree : ";

cin >> value;

cout << endl << "After Reversing, We Get In-Order as : ";

Tree.In\_Order(Tree.Root);

cout << endl;

system("pause");

break;

}

case 0:

{

cout << endl << "You have exited from the system !" << endl;

i = 0;

break;

}

default:

cout << endl << "Invalid Entry !" << endl;

system("pause");

break;

}

}

cout << endl << endl;

system("pause");

}

**A picture containing text, monitor, electronics, indoor

Description automatically generated**

**Question # 11:**

**PROGRAM**

#include <iostream>

using namespace std; // Not Sure About Splittion, Tried but....

struct Node // Node Construction

{

int Data;

Node \*Left\_Node;

Node \*Right\_Node;

};

class Bst\_Tree // Class for Tree Formation

{

public:

Bst\_Tree()

{

Root = NULL;

}

Node \*Root;

Node \*Insertion(Node \*Current, int data) // Insertion of Nodes

{

if (Current == NULL)

{

Current = new Node;

Current->Data = data;

Current->Left\_Node = NULL;

Current->Right\_Node = NULL;

return Current;

}

else if (data < Current->Data) // If Data is lesser than Root

{

Current->Left\_Node = Insertion(Current->Left\_Node, data);

}

else if (data >= Current->Data) // If Data is greater than Root

{

Current->Right\_Node = Insertion(Current->Right\_Node, data);

}

return Current;

}

int Height\_For\_Monk(Node\* temp, int Key)

{

if (temp != NULL)

{

Key++;

int Left\_Height = Height\_For\_Monk(temp->Left\_Node, Key);

int Right\_Height = Height\_For\_Monk(temp->Right\_Node, Key);

if (Right\_Height > Left\_Height)

return Right\_Height;

else

return Left\_Height;

}

return Key;

}

};

int main()

{

Bst\_Tree Tree;

int choice = 0, value = 0, i = 1;

while (i != 0)

{

system("cls");

cout << "=================================================" << endl;

cout << " Press 1 to Enter Values in Tree by an Array !" << endl << endl;

cout << " Press 2 to Check Height of the Tree for Monk !" << endl << endl;

cout << " Press 0 To Exit From The System !" << endl;

cout << "=================================================" << endl;

cout << " Enter Choice : ";

cin >> choice;

switch (choice)

{

case 1:

{

int Aray[10] = { 7,3,8,2,9,5,4,1,6,10 }; // Array Based Values Input

for (int i = 0; i < 10; i++)

{

Tree.Root = Tree.Insertion(Tree.Root, Aray[i]);

}

cout << endl << "Array Based Values Are Inputted !" << endl;

system("pause");

break;

}

case 2:

{

cout << endl << "Height of the Tree is : ";

cout << Tree.Height\_For\_Monk(Tree.Root, 0) << endl;

cout << "\nWhile Constraints are :\n 1 <= N <= 10^3\n 1 <= A[i] <= 10^6\n" << endl;

system("pause");

break;

}

case 0:

{

cout << endl << "You have exited from the system !" << endl;

i = 0;

break;

}

default:

cout << endl << "Invalid Entry !" << endl;

system("pause");

break;

}

}

cout << endl << endl;

system("pause");

}

**A picture containing text, monitor, screenshot, electronics

Description automatically generated**