**Muhammad Abdullah**

**SE (3a) | 19f-0916**

DS LAB 6 AVL

formation of avl tree

**Question # 1: AVL TREE FORMATION**

**PROGRAM:**

#include <iostream>

using namespace std; // AVL Tree Formation

struct Node // Node Construction

{

int Data;

Node \*Left\_Node;

Node \*Right\_Node;

};

class AVL\_Tree // Class for Tree Formation

{

public:

AVL\_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);

Current = Balance\_Tree(Current);

}

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

{

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

Current = Balance\_Tree(Current);

}

return Current;

}

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 Tre

{

if (Root != NULL)

{

int Balance\_Factor = Difference(Current);

if (Balance\_Factor > 1)

{

if (Difference(Current->Left\_Node) > 0)

Current = LL\_Rotation(Current); // Criteria for Left Left Rotation

else

Current = LR\_Rotation(Current); // Criteria for Left Right Rotation

}

else if (Balance\_Factor < -1)

{

if (Difference(Current->Right\_Node) < 0)

Current = RR\_Rotation(Current); // Criteria for Right Right Rotation

else

Current = RL\_Rotation(Current); // Criteria for Right Left Rotation

}

}

else

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

return Current;

}

void Pre\_Order(Node \*Current) // Pre-Order Traversal of the AVL Tree

{

if (Root != NULL)

{

if (Current == NULL)

return;

cout << Current->Data << " ";

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

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

}

else

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

}

void In\_Order(Node \*Current) // In-Order Traversal of the AVL Tree

{

if (Root != NULL)

{

if (Current == NULL)

return;

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

cout << Current->Data << " ";

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

}

else

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

}

void Post\_Order(Node \*Current) // Post-Order Traversal of the AVL Tree

{

if (Root != NULL)

{

if (Current == NULL)

return;

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

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

cout << Current->Data << " ";

}

else

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

}

Node \*LL\_Rotation(Node \*Current) // Left Left Imbalance Rotation

{

Node \*Temp = NULL;

Temp = Current->Left\_Node;

Current->Left\_Node = Current->Right\_Node;

Current->Right\_Node = Current;

return Temp;

}

Node \*LR\_Rotation(Node \*Current) // Left Right Imbalance Rotation

{

Node \*Temp = NULL;

Temp = Current->Left\_Node;

Current->Left\_Node = RR\_Rotation(Current);

return LL\_Rotation(Current);

}

Node \*RR\_Rotation(Node \*Current) // Right Right Imbalance Rotation

{

Node \*Temp = NULL;

Temp = Current->Right\_Node;

Current->Right\_Node = Current->Left\_Node;

Current->Left\_Node = Current;

return Temp;

}

Node \*RL\_Rotation(Node \*Current) // Right Left Imbalance Rotation

{

Node \*Temp = NULL;

Temp = Current->Right\_Node;

Current->Right\_Node = LL\_Rotation(Current);

return RR\_Rotation(Current);

}

};

int main() // Main Code

{

AVL\_Tree AVL;

int Choice = 0, Data = 0, Check = 1;

while (Check == 1) // Menu Driven Code

{

system("cls");

cout << endl << " ================= AVL Tree Formation ==================" << endl;

cout << endl << " Press 1 to Make a Node in AVL Tree !" << endl << endl;

cout << " Press 2 to Traverse in Tree by In-Order Traversal !" << endl;

cout << " Press 3 to Traverse in Tree by Pre-Order Traversal !" << endl;

cout << " Press 4 to Traverse in Tree by Post-Order Traversal !" << endl << endl;

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

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

cout << endl << " Enter Choice : ";

cin >> Choice;

switch (Choice)

{

case 1: // For Insertion

{

cout << endl<<"Enter Data to Push it in AVL Tree :";

cin >> Data;

AVL.Root = AVL.Insertion(AVL.Root, Data);

cout << endl << endl;

system("pause");

break;

}

case 2: // For In-Order Traversal

{

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

AVL.In\_Order(AVL.Root);

cout << endl << endl;

system("pause");

break;

}

case 3: // For Pre-Order Traversal

{

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

AVL.Pre\_Order(AVL.Root);

cout << endl << endl;

system("pause");

break;

}

case 4: // For Post-Order Traversal

{

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

AVL.Post\_Order(AVL.Root);

cout << endl << endl;

system("pause");

break;

}

case 0: // Exiting from Loop

{

cout << endl << "You Have Exited From The System !" << endl;

Check = 0;

break;

}

default:

{

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

break;

}

}

}

cout << endl << endl;

system("pause");

}

**A screenshot of a computer screen

Description automatically generated**