**M. Ali. Arslan**

**19F-0348**

**Lab-07**

**Task # 01**

#include<iostream>

using namespace std;

class Node

{

public:

int data;

Node\* left;

Node\* right;

int height;

Node()

{

data = 0;

left = NULL;

right = NULL;

height = 0;

}

};

class AVL

{

const int s = 5;

public:

Node\* root;

AVL()

{

root = NULL;

}

Node\* get\_root()

{

return root;

}

int height(Node\* N)

{

if (N == NULL)

{

return 0;

}

return N->height;

}

int max(int a, int b)

{

return a > b ? a : b;

}

Node\* newNode(int d)

{

Node\* ptr = new Node; ptr->data = d;

ptr->left = NULL; ptr->right = NULL; ptr->height = 1;

return ptr;

}

Node\* rightRotate(Node\* y)

{

Node\* x = y->left;

Node\* T2 = x->right; x->right = y;

y->left = T2;

y->height = max(height(y->left),

height(y->right)) + 1; x->height =

max(height(x->left),

height(x->right)) + 1;

return x;

}

Node\* leftRotate(Node\* x)

{

Node\* y = x->right; Node\* T2 = y->left; y->left = x;

x->right = T2;

x->height = max(height(x->left),

height(x->right)) + 1;

y->height = max(height(y->left),

height(y->right)) + 1;

return y;

}

int getBalance(Node\* N)

{

if (N == NULL)

{

return 0;

}

return height(N->left) - height(N->right);

}

Node\* insert(Node\* r, int d)

{

if (r == NULL)

{

return newNode(d);

}

if (d < r->data)

{

system("cls");

displayTree(root, s);

cout << endl;

cout << " Comparing " << d << " with " << r->data << endl;

system("pause");

r->left = insert(r->left, d);

}

else if (d > r->data)

{

system("cls");

displayTree(root, s);

cout << endl;

cout << " Comparing " << d << " with " << r->data << endl;

system("pause");

r->right = insert(r->right, d);

}

else

{

system("cls"); displayTree(root, s);

cout << endl;

cout << " Data already Exists! " << endl;

system("pause");

return r;

}

r->height = 1 + max(height(r->left), height(r->right));

int balance = getBalance(r);

if (balance > 1 && d < r->left->data)

{

return rightRotate(r);

}

if (balance < -1 && d > r->right->data)

{

return leftRotate(r);

}

if (balance > 1 && d > r->left->data)

{

r->left = leftRotate(r->left); return rightRotate(r);

}

if (balance < -1 && d < r->right->data)

{

r->right = rightRotate(r->right); return leftRotate(r);

}

return r;

}

void displayTree(Node\* r, int space)

{

if (r == NULL)

{

return;

}

space = space + s; displayTree(r->right, space); cout << endl;

for (int i = s; i < space; i++)

{

cout << " ";

}

cout << r->data << endl; displayTree(r->left, space);

}

};

int main()

{

AVL a1;

int val = 0;

const int SPACE = 5;

do

{

cout << "INPUT VALUE TO INSERT IN AVL TREE (-1 to exit): ";

cin >> val;

if (val == -1)

break;

a1.root = a1.insert(a1.get\_root(), val);

system("cls");

cout << val << " BALANCED " << endl;

a1.displayTree(a1.get\_root(), SPACE);

cout << endl;

} while (val != -1);

cout << endl;

cout << " AVL TREE " << endl;

a1.displayTree(a1.get\_root(), SPACE);

return 0;

}



**Task # 02**

#include<iostream>

using namespace std;

class Node

{

public:

int data;

Node\* left;

Node\* right;

int height;

Node()

{

data = 0;

left = NULL;

right = NULL;

height = 0;

}

};

class AVL

{

const int s = 5;

int count;

public:

Node\* root;

AVL()

{

root = NULL;

}

Node\* get\_root()

{

return root;

}

int get\_count()

{

return count;

}

int height(Node\* N)

{

if (N == NULL)

{

return 0;

}

return N->height;

}

int max(int a, int b)

{

return a > b ? a : b;

}

Node\* newNode(int d)

{

Node\* ptr = new Node; ptr->data = d;

ptr->left = NULL; ptr->right = NULL; ptr->height = 1;

return ptr;

}

Node\* rightRotate(Node\* y)

{

Node\* x = y->left;

Node\* T2 = x->right; x->right = y;

y->left = T2;

y->height = max(height(y->left),

height(y->right)) + 1; x->height =

max(height(x->left),

height(x->right)) + 1;

return x;

}

Node\* leftRotate(Node\* x)

{

Node\* y = x->right; Node\* T2 = y->left; y->left = x;

x->right = T2;

x->height = max(height(x->left),

height(x->right)) + 1;

y->height = max(height(y->left),

height(y->right)) + 1;

return y;

}

int getBalance(Node\* N)

{

if (N == NULL)

{

return 0;

}

return height(N->left) - height(N->right);

}

Node\* insert(Node\* r, int d)

{

if (r == NULL)

{

return newNode(d);

}

if (d < r->data)

{

system("cls");

displayTree(root, s);

cout << endl;

cout << " Comparing " << d << " with " << r->data << endl;

system("pause");

r->left = insert(r->left, d);

}

else if (d > r->data)

{

system("cls");

displayTree(root, s);

cout << endl;

cout << " Comparing " << d << " with " << r->data << endl;

system("pause");

r->right = insert(r->right, d);

}

else

{

system("cls"); displayTree(root, s);

cout << endl;

cout << " Data already Exists! " << endl;

system("pause");

return r;

}

r->height = 1 + max(height(r->left), height(r->right));

int balance = getBalance(r);

if (balance > 1 && d < r->left->data)

{

return rightRotate(r);

}

if (balance < -1 && d > r->right->data)

{

return leftRotate(r);

}

if (balance > 1 && d > r->left->data)

{

r->left = leftRotate(r->left); return rightRotate(r);

}

if (balance < -1 && d < r->right->data)

{

r->right = rightRotate(r->right); return leftRotate(r);

}

return r;

}

void displayTree(Node\* r, int space)

{

if (r == NULL)

{

return;

}

space = space + s; displayTree(r->right, space); cout << endl;

for (int i = s; i < space; i++)

{

cout << " ";

}

cout << r->data << endl; displayTree(r->left, space);

}

void count\_nodes(Node\* r)

{

if (r == NULL)

{

return;

}

count\_nodes(r->left);

if (r->left == NULL && r->right == NULL)

{

count++;

}

count\_nodes(r->right);

}

};

int main()

{

AVL a1;

int val = 0;

const int SPACE = 5;

do

{

cout << "INPUT VALUE TO INSERT IN AVL TREE (-1 to exit): ";

cin >> val;

if (val == -1)

break;

a1.root = a1.insert(a1.get\_root(), val);

system("cls");

cout << val << " BALANCED " << endl;

a1.displayTree(a1.get\_root(), SPACE);

cout << endl;

} while (val != -1);

cout << endl;

cout << " AVL TREE " << endl;

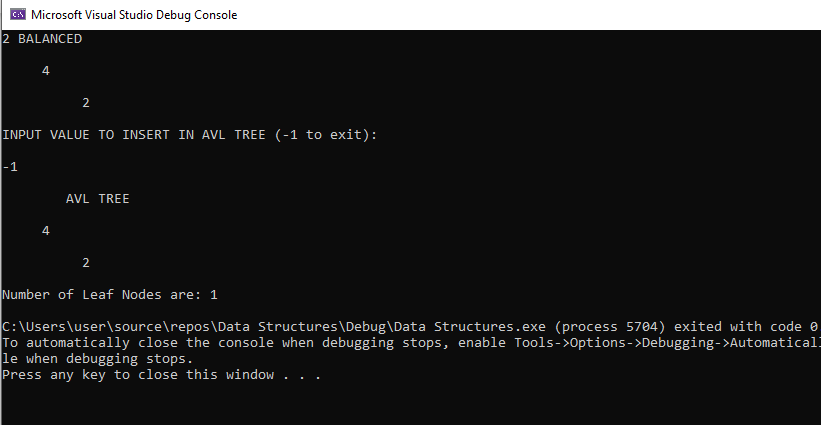
a1.displayTree(a1.get\_root(), SPACE);

a1.count\_nodes(a1.get\_root());

cout << "\nNumber of Leaf Nodes are: " << a1.get\_count() << endl;

return 0;

}



**Task # 03**

#include<iostream>

#include<cstdio>

#include<sstream>

#include<algorithm>

#define pow(n) ( << (n))

using namespace std;

struct avl\_node {

int data;

struct avl\_node\* left;

struct avl\_node\* right;

}

\*root;

class avlTree

{

public:

int height(avl\_node\*);

int diff(avl\_node\*);

avl\_node\* rr\_rotation(avl\_node\*);

avl\_node\* ll\_rotation(avl\_node\*);

avl\_node\* lr\_rotation(avl\_node\*);

avl\_node\* rl\_rotation(avl\_node\*);

avl\_node\* balance(avl\_node\*);

avl\_node\* insert(avl\_node\*, int);

void display(avl\_node\*, int);

void inorder(avl\_node\*);

void preorder(avl\_node\*);

void postorder(avl\_node\*);

avlTree()

{

root = NULL;

}

};

int main()

{

int choice, item;

avlTree avl;

while (1)

{

cout << "\n------------------------- " << endl;

cout << "AVL Tree Implementation" << endl;

cout << "\n--------------------------" << endl;

cout << "l. Insert Element into the tree" << endl;

cout << "2. Display Balanced AVL Tree" << endl;

cout << "3. InOrder traversal" << endl;

cout << "4. PreOrder traversal" << endl;

cout << "5. PostOrder traversal" << endl;

cout << "6. Exit" << endl;

cout << "Enter your Choice: ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter value to be inserted: ";

cin >> item;

root = avl.insert(root, item);

break;

case 2:

if (root == NULL) {

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

continue;

}

cout << "Balanced AVL Tree:" << endl;

avl.display(root, 1);

break;

case 3:

cout << "lnorder Traversal:" << endl;

avl.inorder(root);

cout << endl;

break;

case 4:

cout << "Preorder Traversal:" << endl;

avl.preorder(root);

cout << endl;

break;

case 5:

cout << "Postorder Traversal:" << endl;

avl.postorder(root);

cout << endl;

break;

case 6:

exit(1);

break;

default:

cout << "Wrong Choice" << endl;

}

}

return 0;

}

int avlTree::height(avl\_node\* temp)

{

int h = 0;

if (temp != NULL)

{

int l\_height = height(temp->left);

int r\_height = height(temp->right);

int max\_height = max(l\_height, r\_height); h =

max\_height + 1;

}

return h;

}

int avlTree::diff(avl\_node\* temp)

{

int l\_height = height(temp->left);

int r\_height = height(temp->right);

int b\_factor = l\_height - r\_height;

return b\_factor;

}

avl\_node\* avlTree::rr\_rotation(avl\_node\* parent)

{

avl\_node\* temp;

temp = parent->right;

parent->right = temp->left;

temp->left = parent;

return temp;

}

avl\_node\* avlTree::ll\_rotation(avl\_node\* parent)

{

avl\_node\* temp;

temp = parent->left;

parent->left = temp->right;

temp->right = parent;

return temp;

}

avl\_node\* avlTree::lr\_rotation(avl\_node\* parent)

{

avl\_node \*temp;

temp = parent->left;

parent->left = rr\_rotation(temp);

return ll\_rotation(parent);

}

avl\_node\* avlTree::rl\_rotation(avl\_node\* parent)

{

avl\_node\* temp;

temp = parent->right;

parent->right = ll\_rotation(temp);

return rr\_rotation(parent);

}

avl\_node\* avlTree::balance(avl\_node\* temp)

{

int bal\_factor = diff(temp);

if (bal\_factor > 1)

{

if (diff(temp->left) > 0)

temp = ll\_rotation(temp);

else

temp = lr\_rotation(temp);

}

else if (bal\_factor < -1)

{

if (diff(temp->right) > 0) temp = rl\_rotation(temp);

else

temp = rr\_rotation(temp);

}

return temp;

}

avl\_node\* avlTree::insert(avl\_node\* root, int value)

{

if (root == NULL)

{

root = new avl\_node;

root->data = value;

root->left = NULL;

root->right = NULL;

return root;

}

else if (value < root->data)

{

root->left = insert(root->left, value);

root = balance(root);

}

else if (value >= root->data)

{

root->right = insert(root->right, value); root =

balance(root);

}

return root;

}

void avlTree::display(avl\_node\* ptr, int level)

{

int i;

if (ptr != NULL)

{

display(ptr->right, level + 1);

printf("\n");

if (ptr == root)

cout << "Root ->";

for (i = 0; i < level && ptr != root; i++)

cout << " ";

cout << ptr->data;

display(ptr->left, level + 1);

}

}

void avlTree::inorder(avl\_node\* tree)

{

if (tree == NULL)

return;

inorder(tree->left);

cout << tree->data << " ";

inorder(tree->right);

}

void avlTree::preorder(avl\_node\* tree)

{

if (tree == NULL)

return;

cout << tree->data << " ";

preorder(tree->left);

preorder(tree->right);

}

/\* \* Postorder Traversal of AVL Tree \*/

void avlTree::postorder(avl\_node\* tree)

{

if (tree == NULL)

return;

postorder(tree->left);

postorder(tree->right);

cout << tree->data << " ";

}

