**Question#1**

**Code:**

#include<iostream>

#include<string>

using namespace std;

struct Node {

string data;

Node\* left;

Node\* right;

Node(string val) :data(val), left(nullptr), right(nullptr) {}

};

class Base {

protected:

Node\* root;

public:

Base() {

root = nullptr;

}

virtual void insert(string val) = 0;

virtual void printInOrder() = 0;

virtual void deleteNode(string val) = 0;

virtual bool find(string val) = 0;

};

class Bst :public Base {

public:

Bst() :Base() {}

virtual void insert(string val) {

insertDataHelper(val, root);

}

void printInOrder() {

inOrderHelper(root);

cout << endl;

}

void deleteNode(string val) {

root = deleteNodeHelper(root, val);

}

bool find(string val) {

return findHelper(root, val);

}

private:

int max(int a, int b) {

if (a > b) {

return a;

}

return b;

}

int getHeight(Node\* temp) {

if (temp == nullptr) {

return -1;

}

return 1 + max(getHeight(temp->left), getHeight(temp->right));

}

bool findHelper(Node\* temp, string val) {

if (temp!=nullptr) {

if (temp->data == val) {

return true;

}

return findHelper(temp->left, val);

return findHelper(temp->right, val);

}

return false;

}

Node\* rotateLeft(Node\* temp) {

cout << "Rotating Left" << endl;

Node\* ptr = temp->right;

temp->right = ptr->left;

ptr->left = temp;

return ptr;

}

Node\* rotateRight(Node\* temp) {

cout << "Rotating Right" << endl;

Node\* ptr = temp->left;

temp->left = ptr->right;

ptr->right = temp;

return ptr;

}

int calculateBF(Node\* temp) {

int left = getHeight(temp->left);

int right = getHeight(temp->right);

return left - right;

}

void insertDataHelper(string val, Node\*& temp) {

if (temp == nullptr) {

temp = new Node(val);

return;

}

if (val < temp->data) {

insertDataHelper(val, temp->left);

}

else if (val > temp->data) {

insertDataHelper(val, temp->right);

}

else {

return;

}

int balance = calculateBF(temp);

if (balance > 1 && val < temp->left->data) {

temp = rotateRight(temp);

}

else if (balance < -1 && val > temp->right->data) {

temp = rotateLeft(temp);

}

else if (balance > 1 && val > temp->left->data) {

temp->left = rotateLeft(temp->left);

temp = rotateRight(temp);

}

else if (balance < -1 && val < temp->right->data) {

temp->right = rotateRight(temp->right);

temp = rotateLeft(temp);

}

}

Node\* minValueNode(Node\* temp) {

Node\* current = temp;

while (current && current->left != nullptr)

current = current->left;

return current;

}

Node\* balance(Node\* temp) {

int balance = calculateBF(temp);

if (balance > 1 && calculateBF(temp->left) >= 0) {

return rotateRight(temp);

}

if (balance < -1 && calculateBF(temp->right) <= 0) {

return rotateLeft(temp);

}

if (balance > 1 && calculateBF(temp->left) < 0) {

temp->left = rotateLeft(temp->left);

return rotateRight(temp);

}

if (balance < -1 && calculateBF(temp->right) > 0) {

temp->right = rotateRight(temp->right);

return rotateLeft(temp);

}

return temp;

}

Node\* deleteNodeHelper(Node\* temp, string val) {

if (temp == nullptr) return temp;

if (val < temp->data) {

temp->left = deleteNodeHelper(temp->left, val);

}

else if (val > temp->data) {

temp->right = deleteNodeHelper(temp->right, val);

}

else {

if (temp->left == nullptr || temp->right == nullptr) {

Node\* tempChild = temp->left ? temp->left : temp->right;

if (tempChild == nullptr) {

tempChild = temp;

temp = nullptr;

}

else {

\*temp = \*tempChild;

}

delete tempChild;

}

else {

Node\* tempSuccessor = minValueNode(temp->right);

temp->data = tempSuccessor->data;

temp->right = deleteNodeHelper(temp->right, tempSuccessor->data);

}

}

if (temp == nullptr) return temp;

return balance(temp);

}

void inOrderHelper(Node\* temp) {

if (temp != nullptr) {

inOrderHelper(temp->left);

cout << temp->data << " ";

inOrderHelper(temp->right);

}

}

};

int main() {

Bst obj;

string val[] = { "Apple", "Banana", "Orange", "Mango", "Grapes" };

for (int i = 0; i < 5; i++) {

cout << "Input : " << val[i] << endl;

obj.insert(val[i]);

}

cout << "InOrder : ";

obj.printInOrder();

obj.deleteNode("Orange");

cout << "After Deletion : ";

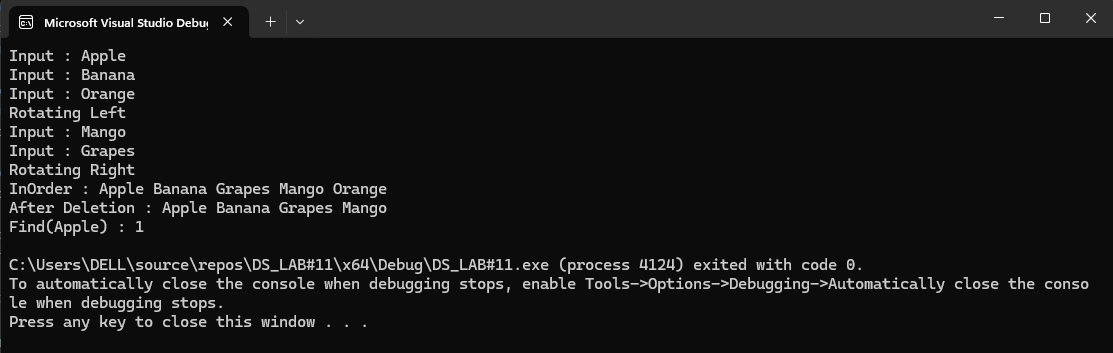
obj.printInOrder();

cout << "Find(Apple) : " << obj.find("Apple") << endl;

return 0;

}

**Output:**

****

**Question#2**

**Code:**

#include<iostream>

#include<cstdlib>

#include<ctime>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int val) :data(val), left(nullptr), right(nullptr) {}

};

class Base {

protected:

Node\* root;

public:

Base() {

root = nullptr;

}

virtual void insert(int val) = 0;

virtual void printInOrder() = 0;

};

class Bst :public Base {

public:

Bst() :Base() {}

virtual void insert(int val) {

insertDataHelper(val, root);

}

void printInOrder() {

inOrderHelper(root);

cout << endl;

}

private:

void insertDataHelper(int val, Node\*& temp) {

if (temp == nullptr) {

temp = new Node(val);

return;

}

else {

if (val < temp->data) {

insertDataHelper(val, temp->left);

}

else if (val > temp->data) {

insertDataHelper(val, temp->right);

}

else if (val == temp->data) {

return;

}

}

}

void inOrderHelper(Node\* temp) {

if (temp != nullptr) {

inOrderHelper(temp->left);

cout << temp->data << " ";

inOrderHelper(temp->right);

}

}

};

class Avl :public Bst {

public:

Node\* avlRoot;

public:

Avl() :Bst() {

avlRoot = root;

}

void ConvertBstToAvlHelper() {

avlRoot = ConvertBstToAvlHelper(root);

}

void printInOrderAvl() {

inOrderAvlHelper(avlRoot);

cout << endl;

}

private:

void inOrderAvlHelper(Node\* temp) {

if (temp != nullptr) {

inOrderAvlHelper(temp->left);

cout << temp->data << " ";

inOrderAvlHelper(temp->right);

}

}

int getHeight(Node\* temp) {

if (temp == nullptr) return -1;

return 1 + max(getHeight(temp->left), getHeight(temp->right));

}

int calculateBF(Node\* temp) {

return getHeight(temp->left) - getHeight(temp->right);

}

Node\* rotateRight(Node\* y) {

cout << "Rotating Right" << endl;

Node\* x = y->left;

y->left = x->right;

x->right = y;

return x;

}

Node\* rotateLeft(Node\* x) {

cout << "Rotating Left" << endl;

Node\* y = x->right;

x->right = y->left;

y->left = x;

return y;

}

Node\* balance(Node\* temp) {

int bf = calculateBF(temp);

if (bf > 1 && calculateBF(temp->left) >= 0) {

return rotateRight(temp);

}

if (bf < -1 && calculateBF(temp->right) <= 0) {

return rotateLeft(temp);

}

if (bf > 1 && calculateBF(temp->left) < 0) {

temp->left = rotateLeft(temp->left);

return rotateRight(temp);

}

if (bf < -1 && calculateBF(temp->right) > 0) {

temp->right = rotateRight(temp->right);

return rotateLeft(temp);

}

return temp;

}

Node\* ConvertBstToAvlHelper(Node\* temp) {

if (!temp) return nullptr;

temp->left = ConvertBstToAvlHelper(temp->left);

temp->right = ConvertBstToAvlHelper(temp->right);

return balance(temp);

}

};

int main() {

srand(time(0));

Bst obj;

int val = 0;

for (int i = 0; i < 7; i++) {

val = rand() % 10;

cout << "Input : " << val << endl;

obj.insert(val);

}

cout << "InOrder : ";

obj.printInOrder();

Avl avl;

avl.ConvertBstToAvlHelper();

cout << "AVL InOrder: ";

obj.printInOrder();

return 0;

}

**Output:**

**A screen shot of a computer

Description automatically generated**

**Question#3**

**Code:**

**Output:**

**Question#4**

**Code:**

**Output:**