**Data Structure**

**Lab 8**

**Name :** Faizan Tariq

**Roll No:** 22F-3858

**TASK 1:**

**CODE :**

#include<iostream>

#include<string>

using namespace std;

struct Node {

int id;

string playerName;

int score;

Node\* left, \* right;

Node() {

left = NULL;

right = NULL;

id = 0;

score = 0;

playerName = "";

}

};

class BST {

Node\* root;

public:

BST() {

root = NULL;

}

void addPlayer(int id, string name, int score) {

Node\* newNode = new Node;

newNode->id = id;

newNode->playerName = name;

newNode->score = score;

newNode->left = NULL;

newNode->right = NULL;

if (!root)

root = newNode; // if the tree is not created

else {

Node\* cur = root;

while (true) {

if (score < cur->score) {

if (cur->left) {

cur = cur->left;

}

else {

cur->left = newNode;

return;

}

}

else if (score > cur->score) {

if (cur->right) {

cur = cur->right;

}

else {

cur->right = newNode;

return;

}

}

else {

cur->left = newNode; // for the same score, store on the left

break;

}

}

}

}

void deleteNode(int id, Node\*& newNode) {

if (newNode == NULL) {

cout << id << " Player not found\n";

}

else if (id < newNode->id) {

deleteNode(id, newNode->left);

}

else if (id > newNode->id) {

deleteNode(id, newNode->right);

}

else {

makeDelete(newNode);

}

}

void makeDelete(Node\*& newNode) {

Node\* deleteNode;

if (newNode->right == NULL) {

deleteNode = newNode;

newNode = newNode->left;

delete deleteNode;

}

else if (newNode->left == NULL) {

deleteNode = newNode;

newNode = newNode->right;

delete deleteNode;

}

else {

deleteNode = newNode->right;

while (deleteNode->left) {

deleteNode = deleteNode->left;

}

deleteNode->left = newNode->left;

deleteNode = newNode;

newNode = newNode->right;

delete deleteNode;

}

}

void updateScore(int playerID, int updatedScore) {

Node\* cur = root;

Node\* prev = NULL;

while (cur) {

if (playerID == cur->id) {

cout << "Player found .\n";

cur->score = updatedScore;

break;

}

prev = cur;

if (playerID < cur->id) {

cur = cur->left;

}

else {

cur = cur->right;

}

}

if (cur == nullptr) {

cout << "Player with ID " << playerID << " not found." << endl;

return;

}

if (prev) {

if (prev->left == cur) {

prev->left = nullptr;

}

else {

prev->right = nullptr;

}

}

else {

root = nullptr;

}

addPlayer(playerID, cur->playerName, updatedScore);

}

void getTopPlayers(Node\* pointerNode, int& num) {

if (pointerNode != NULL && num > 0) {

getTopPlayers(pointerNode->right, num);

cout << "\nPlayer ID : " << pointerNode->id << endl;

cout << "Player Name : " << pointerNode->playerName << endl;

cout << "Player Score : " << pointerNode->score << endl;

num--;

getTopPlayers(pointerNode->left, num);

}

}

Node\* getRootNode() {

return root;

}

};

int main() {

BST RankingSystem;

RankingSystem.addPlayer(1, "Player1", 100);

RankingSystem.addPlayer(2, "Player2", 30);

RankingSystem.addPlayer(3, "Player3", 190);

RankingSystem.addPlayer(4, "Player4", 95);

cout << "Enter Number of top players to display: ";

int num;

cin >> num;

cout << "Top Players:" << endl;

RankingSystem.getTopPlayers(RankingSystem.getRootNode(), num);

int playerId, updatedScore;

cout << "Enter Player Id to update: ";

cin >> playerId;

cout << "Enter Score to update: ";

cin >> updatedScore;

RankingSystem.updateScore(playerId, updatedScore);

cout << "Player " << playerId << " updated to score " << updatedScore << endl;

cout << "Enter Number of top players to display: ";

int newnum;

cin >> newnum;

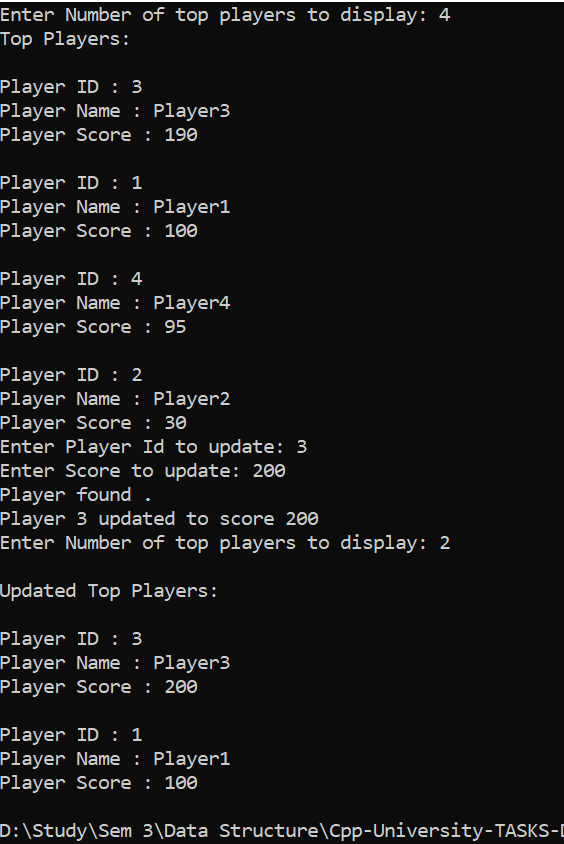
cout << "\nUpdated Top Players:" << endl;

RankingSystem.getTopPlayers(RankingSystem.getRootNode(), newnum);

return 0;

}

OUTPUT:



TASK 2 :

CODE :

#include<iostream>

#include<string>

using namespace std;

int counter = 0;// to calculate number of leaf nodes

struct Node {

int id;

string playerName;

int score;

Node\* left, \* right;

Node() {

left = NULL;

right = NULL;

id = 0;

score = 0;

playerName = "";

}

};

class BST {

Node\* root;

public:

BST() {

root = NULL;

}

void addPlayer(int id, string name, int score) {

Node\* newNode = new Node;

newNode->id = id;

newNode->playerName = name;

newNode->score = score;

newNode->left = NULL;

newNode->right = NULL;

if (!root)

root = newNode; // if the tree is not created

else {

Node\* cur = root;

while (true) {

if (score < cur->score) {

if (cur->left) {

cur = cur->left;

}

else {

cur->left = newNode;

return;

}

}

else if (score > cur->score) {

if (cur->right) {

cur = cur->right;

}

else {

cur->right = newNode;

return;

}

}

else {

cur->left = newNode; // for the same score, store on the left

break;

}

}

}

}

void deleteNode(int id, Node\*& newNode) {

if (newNode == NULL) {

cout << id << " Player not found\n";

}

else if (id < newNode->id) {

deleteNode(id, newNode->left);

}

else if (id > newNode->id) {

deleteNode(id, newNode->right);

}

else {

makeDelete(newNode);

}

}

void makeDelete(Node\*& newNode) {

Node\* deleteNode;

if (newNode->right == NULL) {

deleteNode = newNode;

newNode = newNode->left;

delete deleteNode;

}

else if (newNode->left == NULL) {

deleteNode = newNode;

newNode = newNode->right;

delete deleteNode;

}

else {

deleteNode = newNode->right;

while (deleteNode->left) {

deleteNode = deleteNode->left;

}

deleteNode->left = newNode->left;

deleteNode = newNode;

newNode = newNode->right;

delete deleteNode;

}

}

void updateScore(int playerID, int updatedScore) {

Node\* cur = root;

Node\* prev = NULL;

while (cur) {

if (playerID == cur->id) {

cout << "Player found .\n";

cur->score = updatedScore;

break;

}

prev = cur;

if (playerID < cur->id) {

cur = cur->left;

}

else {

cur = cur->right;

}

}

if (cur == nullptr) {

cout << "Player with ID " << playerID << " not found." << endl;

return;

}

if (prev) {

if (prev->left == cur) {

prev->left = nullptr;

}

else {

prev->right = nullptr;

}

}

else {

root = nullptr;

}

addPlayer(playerID, cur->playerName, updatedScore);

}

void getTopPlayers(Node\* pointerNode, int& num) {

if (pointerNode != NULL && num > 0) {

getTopPlayers(pointerNode->right, num);

cout << "\nPlayer ID : " << pointerNode->id << endl;

cout << "Player Name : " << pointerNode->playerName << endl;

cout << "Player Score : " << pointerNode->score << endl;

num--;

getTopPlayers(pointerNode->left, num);

}

}

Node\* getRootNode() {

return root;

}

void countLeafNodes(Node\* node) {

if (node == nullptr) {

return ;

}

if (node->left == nullptr && node->right == nullptr) {

counter++;

}

countLeafNodes(node->left);

countLeafNodes(node->right);

}

};

int main() {

BST RankingSystem;

RankingSystem.addPlayer(1, "Player1", 100);

RankingSystem.addPlayer(2, "Player2", 30);

RankingSystem.addPlayer(3, "Player3", 190);

RankingSystem.addPlayer(4, "Player4", 95);

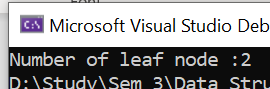
RankingSystem.countLeafNodes(RankingSystem.getRootNode());

cout <<"Number of leaf node :"<< counter;

return 0;

}

OUTPUT :



**TASK 3:**

**CODE :** #include <iostream>

#include <string>

using namespace std;

struct Node {

string courseCode;

Node\* left;

Node\* right;

Node(string code) : courseCode(code), left(NULL), right(NULL) {}

};

class CourseScheduler {

public:

Node\* root;

CourseScheduler() : root(NULL) {}

Node\* EnrollCourse(Node\* node, string code) {

if (node == NULL) {

return new Node(code);

}

if (code < node->courseCode) {

node->left = EnrollCourse(node->left, code);

}

else if (code > node->courseCode) {

node->right = EnrollCourse(node->right, code);

}

return node;

}

Node\* WithdrawCourse(Node\* node, string code) {

if (node == NULL) {

return node;

}

if (code > node->courseCode) {

node->right = WithdrawCourse(node->right, code);

}

else if (code < node->courseCode) {

node->left = WithdrawCourse(node->left, code);

}

else {

if (node->left == NULL) {

Node\* temp = node->right;

delete node;

return temp;

}

else if (node->right == NULL) {

Node\* temp = node->left;

delete node;

return temp;

}

Node\* minRight = findMinimum(node->right);

node->courseCode = minRight->courseCode;

node->right = WithdrawCourse(node->right, minRight->courseCode);

}

return node;

}

Node\* search(Node\* node, string code) {

if (node == NULL || node->courseCode == code) {

return node;

}

if (code < node->courseCode) {

return search(node->left, code);

}

else {

return search(node->right, code);

}

}

void inOrderTraversal(Node\* node) {

if (node != NULL) {

inOrderTraversal(node->left);

cout << node->courseCode << endl;

inOrderTraversal(node->right);

}

}

Node\* findMinimum(Node\* node) {

while (node->left != NULL) {

node = node->left;

}

return node;

}

Node\* getRoot() {

return root;

}

};

int main() {

CourseScheduler scheduler;

scheduler.root= scheduler.EnrollCourse(scheduler.getRoot(), "Data");

scheduler.root = scheduler.EnrollCourse(scheduler.getRoot(), "COAL");

scheduler.root = scheduler.EnrollCourse(scheduler.getRoot(), "Discrete");

cout << "Enrolled Courses :\n";

scheduler.inOrderTraversal(scheduler.getRoot());

cout << "\nWithdraw Course COAL.\n";

scheduler.root = scheduler.WithdrawCourse(scheduler.getRoot(), "COAL");

cout << "\nFinding DATA COURSE.\n";

Node\* course = scheduler.search(scheduler.getRoot(), "Data");

if (course != NULL) {

cout << "Found: " << course->courseCode << endl;

}

else {

cout << "Course not found." << endl;

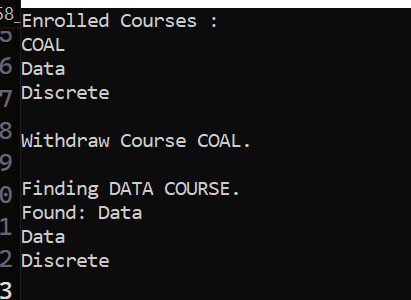
}

scheduler.inOrderTraversal(scheduler.getRoot());

return 0;

}

**OUTPUT :**

****