**Muhammad Abdullah**

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

DS Assignment 3

Student management system with bst

**Question # 1: BST**

**PROGRAM**

#include<iostream>

#include <string> //Management Using BST Formation

using namespace std;

struct Node

{

string Name;

int Roll\_no; // Student Data

float CGPA;

string Department;

Node \*Left\_Node;

Node \*Right\_Node;

};

class Management

{

public:

Management()

{

Head = NULL, current = NULL;

}

Node \*Head, \*current;

void insertion(string name, int roll, float cgpa, string department)

{

Node \*temp = new Node; //Insertion on CGPA

current = Head;

temp->CGPA = cgpa;

temp->Department = department;

temp->Name = name;

temp->Roll\_no = roll;

if (Head == NULL)

{

Head = temp; // If there is no tree

temp->Left\_Node = NULL;

temp->Right\_Node = NULL;

current = Head;

}

else

{

int i;

while (i = 1)

{

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

{

if (cgpa >= current->CGPA)

{

current = current->Right\_Node;

} // If tree exist then go to leaf node (Multiple nodes)

else if (cgpa < current->CGPA)

{

current = current->Left\_Node;

}

if (current->Right\_Node == NULL && current->CGPA <= cgpa)

{

i = 0;

break;

}

if (current->Left\_Node == NULL&&current->CGPA >= cgpa)

{

i = 2;

break;

}

}

else

{

if (cgpa >= current->CGPA)

{

current->Right\_Node = temp;

temp->Right\_Node = NULL;

temp->Left\_Node = NULL;

break;

}

if (cgpa < current->CGPA) // if there is only one node

{

current->Left\_Node = temp;

temp->Right\_Node = NULL;

temp->Left\_Node = NULL;

break;

}

}

}

if (i == 0)

{

current->Right\_Node = temp;

temp->Left\_Node = NULL;

temp->Right\_Node = NULL;

}

if (i == 2)

{

current->Left\_Node = temp;

temp->Right\_Node = NULL;

temp->Left\_Node = NULL;

}

current = Head;

}

}

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

{

if (Head == NULL)

{

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

return;

}

else

{

if (C == NULL)

return;

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

cout << " Name = " << C->Name << "\_\_\_\_ CGPA = " << C->CGPA << "\_\_\_\_ ROLL NO. = " << C->Roll\_no << "\_\_\_\_ Department = " << C->Department << endl;

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

}

}

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

{

if (Head == NULL)

{

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

return;

}

else

{

Node \*temp = C;

if (temp == NULL)

return;

cout << " Name = " << C->Name << "\_\_\_\_ CGPA = " << C->CGPA << "\_\_\_\_ ROLL NO. = " << C->Roll\_no << "\_\_\_\_ Department = " << C->Department << endl;

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

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

}

}

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

{

if (Head == 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 << " Name = " << C->Name << "\_\_\_\_ CGPA = " << C->CGPA << "\_\_\_\_ ROLL NO. = " << C->Roll\_no << "\_\_\_\_ Department = " << C->Department << endl;

}

}

void Max\_CGPA() // Showing Maximum CGPA

{

if (Head != NULL)

{

current = Head;

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

{

current = current->Right\_Node;

}

cout << endl << "Student with Maximum CGPA is : " << " Name = " << current->Name << "\_\_\_\_ CGPA = " << current->CGPA << "\_\_\_\_ ROLL NO. = " << current->Roll\_no << "\_\_\_\_ Department = " << current->Department << endl;

current = Head;

}

else

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

}

void Min\_CGPA() //Showing Minimum CGPA

{

if (Head != NULL)

{

current = Head;

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

{

current = current->Left\_Node;

}

cout << endl << "Student with Manimum CGPA is : " << " Name = " << current->Name << ".... CGPA = " << current->CGPA << ".... ROLL NO. = " << current->Roll\_no << ".... Department = " << current->Department << endl;

current = Head;

}

else

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

}

int Height\_Tree(Node\* C) // Showing Height of Tree

{

if (Head == NULL)

{

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

return 0;

}

else

{

Node \*temp = C;

if (temp == NULL)

return 0;

int Left\_Height = Height\_Tree(temp->Left\_Node);

int Right\_Height = Height\_Tree(temp->Right\_Node);

if (Left\_Height >= Right\_Height)

return(Left\_Height + 1);

else return(Right\_Height + 1);

}

}

void Edges\_From\_Root() // Showing Edges of Tress from Main Root Node

{

if (Head != NULL)

{

current = Head;

int x = Height\_Tree(current);

cout << "Edges of Tree to Maximum Height Are : " << x - 1 << endl;

current = Head;

}

else

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

}

void Search\_Roll\_No(int roll, Node \*C) // Searching with Roll Number

{

if (Head == NULL)

{

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

return;

}

else

{

Node \*temp = C;

if (temp == NULL)

return;

Search\_Roll\_No(roll, temp->Left\_Node);

if (temp->Roll\_no == roll)

{

cout << endl << "Rquired Student is : " << " Name = " << temp->Name << ".... CGPA = " << temp->CGPA << ".... ROLL NO. = " << temp->Roll\_no << ".... Department = " << temp->Department << endl;

return;

}

Search\_Roll\_No(roll, temp->Right\_Node);

}

}

void Tree\_Mirror(Node\* C) //Making Mirror of BST

{

if (Head == NULL)

{

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

return;

}

else

{

Node \*temp = NULL, \*temp1 = C;

if (C == NULL)

return;

Tree\_Mirror(temp1->Left\_Node);

Tree\_Mirror(temp1->Right\_Node);

temp = temp1->Left\_Node;

temp1->Left\_Node = temp1->Right\_Node;

temp1->Right\_Node = temp;

}

}

};

int main() // Main Code with Menu Driven Approach

{

Management BST;

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

float CGPA = 0.0;

string name = "0", department = "0";

while (i == 1)

{

system("cls");

cout << "================Student Managment with BST=================" << endl;

cout << endl << " Press 1 to Enter a Student !" << 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 5 to Check Student with Maximum CGPA !" << endl;

cout << " Press 6 to Check Student with Minimum CGPA !" << endl << endl;

cout << " Press 7 to Check Height of Tree from Root Node !" << endl;

cout << " Press 8 to Check Edges of Tree from Root Node !" << endl << endl;

cout << " Press 9 to Search a Student with Roll Number !" << endl << endl;

cout << " Press 10 to Convert Tree into its Mirror Tree !" << endl << endl;

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

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

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

cin >> choice;

switch (choice)

{

case 1:

{

cout << endl << "Enter the Name of Student : ";

cin >> name;

cout << "Enter the Roll Number of Student : "; //Insertion

cin >> Roll;

cout << "Enter the CGPA of Student : ";

cin >> CGPA;

cout << "Enter the Department of Student : ";

cin >> department;

BST.insertion(name, Roll, CGPA, department);

cout << endl;

system("pause");

break;

}

case 2:

{

cout << endl;

BST.In\_Order(BST.current); //InOrder

cout << endl;

system("pause");

break;

}

case 3:

{

cout << endl;

BST.Pre\_Order(BST.current); //PreOrder

cout << endl;

system("pause");

break;

}

case 4:

{

cout << endl;

BST.Post\_Order(BST.current); //PostOrder

cout << endl;

system("pause");

break;

}

case 5:

{

cout << endl;

BST.Max\_CGPA(); //Max Cgpa

cout << endl;

system("pause");

break;

}

case 6:

{

cout << endl;

BST.Min\_CGPA(); //Min Cgpa

cout << endl;

system("pause");

break;

}

case 7:

{

cout << endl;

cout << " Height of Tree is : " << BST.Height\_Tree(BST.current); //Max Height

cout << endl;

system("pause");

break;

}

case 8:

{

cout << endl;

BST.Edges\_From\_Root(); //Max Roots from root to leave

cout << endl;

system("pause");

break;

}

case 9:

{

int RN = 0;

cout << endl << "Enter Roll Number to Find Student : "; //Finding using Roll No.

cin >> RN;

BST.Search\_Roll\_No(RN, BST.current);

cout << endl;

system("pause");

break;

}

case 10: // Mirror Of Tree

{

BST.Tree\_Mirror(BST.current);

cout << endl;

system("pause");

break;

}

case 0:

{

i = 0;

cout << "You Have Exited, Thank You !" << endl << endl; // Exited

system("pause");

break;

}

default:

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

system("pause");

break;

}

}

cout << endl << endl;

system("pause");

}

**MAIN SYSTEM**

A screen shot of a computer

Description automatically generated

**IN ORDER**

A screen shot of a computer

Description automatically generated

**PREORDER**

A screen shot of a computer

Description automatically generated

**MAX NUMBER**

A screenshot of a computer

Description automatically generated

**Height**

A screenshot of a computer screen

Description automatically generated

**SEARCH WITH ROLL #**  
A screen shot of a computer

Description automatically generated

**MIRROR INORDER**

A screen shot of a computer

Description automatically generated