ASSIGNMENT NO 05

#include <iostream>

#include <queue>

using namespace std;

struct Node

{

int data;

Node \*left, \*right;

};

class BST

{

Node\* root;

Node\* mirror;

public:

BST();

Node\* getroot();

Node\* getmirror();

void insert();

void insert\_rec(Node \*);

void del(int);

Node\* delete\_Node(Node \*, int);

//void mir();

void mirror\_tree(Node \*);

void search(int);

void display(Node \*);

void display\_level();

int height\_tree(Node\*);

};

BST::BST()

{

root = NULL;

mirror = NULL;

}

Node\* BST::getroot()

{

return root;

}

Node\* BST::getmirror()

{

return mirror;

}

void BST::insert()

{

Node\* temp, \*newNode;

char ch = 'y';

while(ch == 'y' || ch == 'Y')

{

newNode = new Node();

cout<<"\nEnter data for new Node: ";

cin>>newNode->data;

newNode->left = newNode->right = NULL;

if(root == NULL)

root = newNode;

else

{

temp = root;

check:

if(newNode->data < temp->data)

{

if(temp->left != NULL)

{

temp = temp->left;

goto check;

}

else

{

temp->left = newNode;

}

}

else if(newNode->data > temp->data)

{

if(temp->right != NULL)

{

temp = temp->right;

goto check;

                 }

else

{

temp->right = newNode;

}

             }

        }

        cout<<"\nEnter y or Y to add more Nodes: ";

        cin>>ch;

    }

}

void BST::del(int x)

{

root = delete\_Node(root, x);

}

Node\* BST::delete\_Node(Node \*t, int x)

{

Node \*temp;

if(t == NULL)

{

cout<<"\nElement not found!";

return t;

     }

if(x < t->data)

{

t->left = delete\_Node(t->left, x);

return t;

}

else if(x > t->data)

{

t->right = delete\_Node(t->right, x);

return t;

}

else

{

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

{

temp = t;

delete temp;

return NULL;

}

if(t->left == NULL)

{

temp = t;

t = t->right;

delete temp;

return t;

}

if(t->right == NULL)

{

temp = t;

t = t->left;

delete temp;

return t;

}

temp = t->right;

while(temp->left != NULL)

temp = temp->left;

t->data = temp->data;

t->right = delete\_Node(t->right, temp->data);

return t;

}

return NULL;

}

void BST::mirror\_tree(Node\* t)

{

Node\* temp;

if(t!=NULL)

{

temp = t->left;

t->left = t->right;

t->right = temp;

mirror\_tree(t->left);

mirror\_tree(t->right);

}

else

return;

}

void BST::search(int no)

{

Node \*temp;

temp = root;

check:

if(no == temp->data)

{

cout<<"\nData found!";

return;

}

if(no < temp->data)

{

if(temp->left != NULL)

{

temp = temp->left;

goto check;

}

else

{

cout<<"\nData doesnt exist.";

return;

}

}

else if(no > temp->data)

{

if(temp->right != NULL)

{

temp = temp->right;

goto check;

}

else

{

cout<<"\nData doesnt exist.";

return;

}

}

}

void BST::display(Node \*root)

{

if(root == NULL)

return;

else

{

display(root->left);

cout<<" "<<root->data<<" ";

display(root->right);

}

}

void BST::display\_level()

{

queue<Node\*> q, q2;

Node\* curr;

Node\* temp = root;

if(temp == NULL)

return;

q.push(temp);

q.push(NULL);

while (q.size() > 1)

{

curr = q.front();

q.pop();

if (curr == NULL)

{

q.push(NULL);

cout << "\n";

}

else

{

if(curr->left)

q.push(curr->left);

if(curr->right)

q.push(curr->right);

cout << curr->data << " ";

}

}

}

int BST::height\_tree(Node \*t)

{

int hlt, hrt;

if(t == NULL)

return 0;

else

{

hlt = height\_tree(t->left) + 1;

hrt = height\_tree(t->right) + 1;

}

if(hlt > hrt)

return hlt;

return hrt;

}

int main()

{

BST tree;

int x, choice;

char ch = 'y';

while(1)

{

cout<<"\n\nBinary Search Tree Operations ";

cout<<"\n1. Insert.";

cout<<"\n2. Delete.";

cout<<"\n3. Search.";

cout<<"\n4. Mirror Image.";

cout<<"\n5. Display.";

cout<<"\n6. Display Level-wise.";

cout<<"\n7. Height of the tree.";

cout<<"\n8. Exit.";

cout<<"\nEnter your choice: ";

cin>>choice;

switch(choice)

{

case 1:

tree.insert();

break;

case 2:

cout<<"\nEnter data to delete: ";

cin>>x;

tree.del(x);

break;

case 3:

cout<<"\nEnter number to search: ";

cin>>x;

tree.search(x);

break;

case 4:

cout<<"\nMirror Image of tree: ";

tree.mirror\_tree(tree.getroot());

tree.display(tree.getroot());

tree.mirror\_tree(tree.getroot());

break;

case 5:

cout<<"\nInorder traversal of tree: ";

tree.display(tree.getroot());

break;

case 6:

cout<<"\nLevel-Order traversal of tree: ";

tree.display\_level();

break;

case 7:

cout<<"\nHeight of the tree: "<<tree.height\_tree(tree.getroot());

break;

case 8:

return 0;

}

}

return 0;

}

OUTPUT :

Binary Search Tree Operations

1. Insert.

2. Delete.

3. Search.

4. Mirror Image.

5. Display.

6. Display Level-wise.

7. Height of the tree.

8. Exit.

Enter your choice: 1

Enter data for new Node: 10

Enter y or Y to add more Nodes: y

Enter data for new Node: 20

Enter y or Y to add more Nodes: y

Enter data for new Node: 30

Enter y or Y to add more Nodes: y

Enter data for new Node: 40

Enter y or Y to add more Nodes: y

Enter data for new Node: 50

Enter y or Y to add more Nodes: 2

Binary Search Tree Operations

1. Insert.

2. Delete.

3. Search.

4. Mirror Image.

5. Display.

6. Display Level-wise.

7. Height of the tree.

8. Exit.

Enter your choice: 2

Enter data to delete: 20

Binary Search Tree Operations

1. Insert.

2. Delete.

3. Search.

4. Mirror Image.

5. Display.

6. Display Level-wise.

7. Height of the tree.

8. Exit.

Enter your choice: 5

Inorder traversal of tree:  10  30  40  50

Binary Search Tree Operations

1. Insert.

2. Delete.

3. Search.

4. Mirror Image.

5. Display.

6. Display Level-wise.

7. Height of the tree.

8. Exit.

Enter your choice: 3

Enter number to search: 40

Data found!

Binary Search Tree Operations

1. Insert.

2. Delete.

3. Search.

4. Mirror Image.

5. Display.

6. Display Level-wise.

7. Height of the tree.

8. Exit.

Enter your choice: 4

Mirror Image of tree:  50  40  30  10

Binary Search Tree Operations

1. Insert.

2. Delete.

3. Search.

4. Mirror Image.

5. Display.

6. Display Level-wise.

7. Height of the tree.

8. Exit.

Enter your choice: 6

Level-Order traversal of tree: 10

30

40

50

Binary Search Tree Operations

1. Insert.

2. Delete.

3. Search.

4. Mirror Image.

5. Display.

6. Display Level-wise.

7. Height of the tree.

8. Exit.

Enter your choice: 7

Height of the tree: 4

Binary Search Tree Operations

1. Insert.

2. Delete.

3. Search.

4. Mirror Image.

5. Display.

6. Display Level-wise.

7. Height of the tree.

8. Exit.

Enter your choice: 8