#include <iostream>

using namespace std;

struct Node

{

int key, height;

Node\* L, \* R;

};

class Tree

{

private:

Node\* root;

public:

Tree() {

root = NULL;

};

Node\* getRoot()

{

return root;

}

Node\* insert(int num, Node\* temp)

{

if (temp == NULL)

{

temp = new Node;

temp->key = num;

temp->L = temp->R = NULL;

}

else if (num < temp->key)

{

temp->L = insert(num, temp->R);

}

else if (num > temp->key)

{

temp->R = insert(num, temp->R);

}

return temp;

}

//print display of tree

void display(Node\* r, int space)

{

if (r == NULL)

{

return;

}

display(r->R, space);

cout << endl;

cout << r->key << "\n";

display(r->L, space);

}

//print inorder of tree

void inorderPrint(Node\* T)

{

if (T == NULL)

return;

inorderPrint(T->L);

cout << T->key << " ";

inorderPrint(T->R);

}

//print preorder of tree

void print\_preorder(Node\* node)

{

if (node == NULL)

{

return;

}

else

{

cout << node->key << " ";

inorderPrint(node->L);

inorderPrint(node->R);

}

}

//print postorder of tree

void print\_postorder(Node\* node)

{

if (node == NULL)

{

return;

}

else

{

inorderPrint(node->L);

inorderPrint(node->R);

cout << node->key << " ";

}

}

Node\* minval(struct Node\* node)

{

Node\* temp = node;

while (temp && temp->L != NULL)

{

temp = temp->L;

}

return temp;

}

Node\* deleteNode( Node\* temp1, int key)

{

if (temp1 == NULL)

{

return temp1;

}

if (key < temp1->key)

{

temp1->L = deleteNode(temp1->L, key);

}

else if (key > temp1->key)

{

temp1->R = deleteNode(temp1->R, key);

}

else

{

if (temp1->L == NULL)

{

Node\* temp = temp1->R;

delete temp1;

return temp;

}

else if (temp1->R == NULL)

{

Node\* temp = temp1->L;

delete temp1;

return temp;

}

Node\* temp = minval(temp1->R);

temp1->key = temp->key;

temp1->R = deleteNode(temp1->R, temp->key);

}

return temp1;

}

};

int main()

{

Tree tree;

Node\* T = tree.getRoot();

int opt, num;

bool exit = false;

while (!exit)

{

cout << "Press 1 for Inserting values" << endl;

cout << "Press 2 for display" << endl;

cout << "Press 3 for Inorder display" << endl;

cout << "Press 4 for preorder display" << endl;

cout << "Press 5 for postorder display" << endl;

cout << "press 6 to delete element " << endl;

cout << "Press 7 to Exit" << endl;

cout << "Your Option is: ";

cin >> opt;

switch (opt)

{

case 1:

cout << "Enter element: ";

cin >> num;

T = tree.insert( num,T);

break;

case 2:

cout << "Display of Tree: ";

tree.display(T,5);

cout << endl;

break;

case 3:

cout << "Inorder print of Tree: ";

tree.inorderPrint(T);

cout << endl;

break;

case 4:

cout << "Preorder print of Tree: ";

tree.print\_preorder(T);

cout << endl;

break;

case 5:

cout << "postorder print of Tree: ";

tree.print\_postorder(T);

cout << endl;

break;

case 6:

int a;

cout << "Enter element to delete :";

cin >> a;

tree.deleteNode(T,a);

cout << endl;

break;

case 7:

exit = true;

break;

default:

cout << "Option is invalid!" << endl;

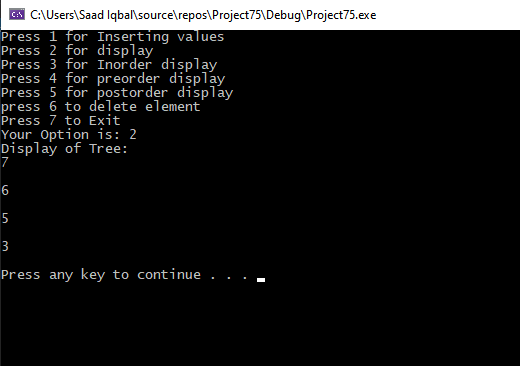
}

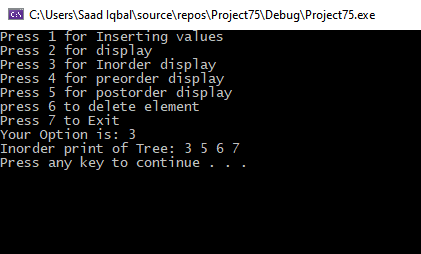
system("pause");

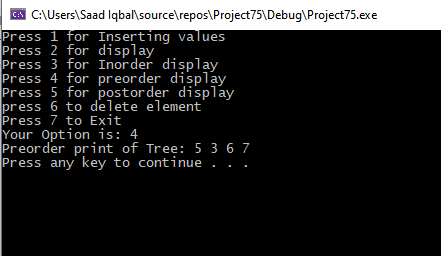
system("cls");

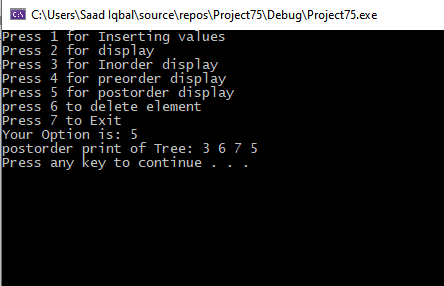
}

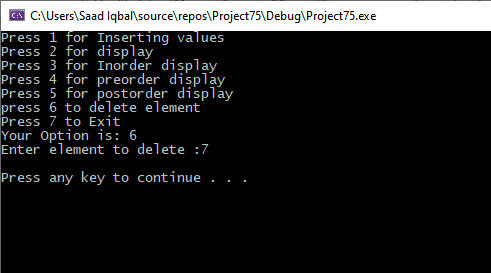
}

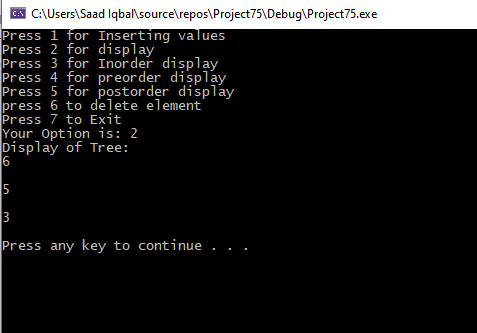












**TASK 2:**

#include <iostream>

using namespace std;

struct Node

{

int key, height;

Node\* L, \* R;

};

class Tree

{

private:

Node\* root;

public:

Tree() {

root = NULL;

};

Node\* getRoot()

{

return root;

}

Node\* insert(int num, Node\* temp)

{

if (temp == NULL)

{

temp = new Node;

temp->key = num;

temp->L = temp->R = NULL;

}

else if (num < temp->key)

{

temp->L = insert(num, temp->R);

}

else if (num > temp->key)

{

temp->R = insert(num, temp->R);

}

return temp;

}

//print display of tree

void display(Node\* r, int space)

{

if (r == NULL)

{

return;

}

display(r->R, space);

cout << endl;

cout << r->key << "\n";

display(r->L, space);

}

//print inorder of tree

void inorderPrint(Node\* T)

{

if (T == NULL)

return;

inorderPrint(T->L);

cout << T->key << " ";

inorderPrint(T->R);

}

//print preorder of tree

void print\_preorder(Node\* node)

{

if (node == NULL)

{

return;

}

else

{

cout << node->key << " ";

inorderPrint(node->L);

inorderPrint(node->R);

}

}

//print postorder of tree

void print\_postorder(Node\* node)

{

if (node == NULL)

{

return;

}

else

{

inorderPrint(node->L);

inorderPrint(node->R);

cout << node->key << " ";

}

}

Node\* minval(struct Node\* node)

{

Node\* temp = node;

while (temp && temp->L != NULL)

{

temp = temp->L;

}

return temp;

}

Node\* deleteNode( Node\* temp1, int key)

{

if (temp1 == NULL)

{

return temp1;

}

if (key < temp1->key)

{

temp1->L = deleteNode(temp1->L, key);

}

else if (key > temp1->key)

{

temp1->R = deleteNode(temp1->R, key);

}

else

{

if (temp1->L == NULL)

{

Node\* temp = temp1->R;

delete temp1;

return temp;

}

else if (temp1->R == NULL)

{

Node\* temp = temp1->L;

delete temp1;

return temp;

}

Node\* temp = minval(temp1->R);

temp1->key = temp->key;

temp1->R = deleteNode(temp1->R, temp->key);

}

return temp1;

}

int power(int base, int power)

{

int res = 1;

while (power > 0)

{

if (power & 1) res \*= base;

base \*= base;

power >>= 1;

}

return res;

}

int countNodes(Node\* root)

{

int height\_of\_right = 0;

int height\_of\_left = 0;

Node\* l = root;

Node\* r = root;

while (l) {

height\_of\_left++;

l = l->L;

}

while (r) {

r = r->R;

height\_of\_right++;

}

if (height\_of\_left == height\_of\_right)

{

return power(2, height\_of\_left) - 1;

}

else

{

return 1 + countNodes(root->L) + countNodes(root->R);

}

}

};

int main()

{

Tree tree;

Node\* T = tree.getRoot();

int opt, num;

bool exit = false;

while (!exit)

{

cout << "Press 1 for Inserting values" << endl;

cout << "Press 2 for display" << endl;

cout << "Press 3 for Inorder display" << endl;

cout << "Press 4 for preorder display" << endl;

cout << "Press 5 for postorder display" << endl;

cout << "press 6 to delete element " << endl;

cout << "press 7 to count nodes " << endl;

cout << "Press 8 to Exit" << endl;

cout << "Your Option is: ";

cin >> opt;

switch (opt)

{

case 1:

cout << "Enter element: ";

cin >> num;

T = tree.insert( num,T);

break;

case 2:

cout << "Display of Tree: ";

tree.display(T,5);

cout << endl;

break;

case 3:

cout << "Inorder print of Tree: ";

tree.inorderPrint(T);

cout << endl;

break;

case 4:

cout << "Preorder print of Tree: ";

tree.print\_preorder(T);

cout << endl;

break;

case 5:

cout << "postorder print of Tree: ";

tree.print\_postorder(T);

cout << endl;

break;

case 6:

int a;

cout << "Enter element to delete :";

cin >> a;

tree.deleteNode(T,a);

cout << endl;

break;

case 7:

cout << "Total number of nodes in a Tree is: ";

cout<<tree.countNodes(T);

cout << endl;

case 8:

exit = true;

break;

default:

cout << "Option is invalid!" << endl;

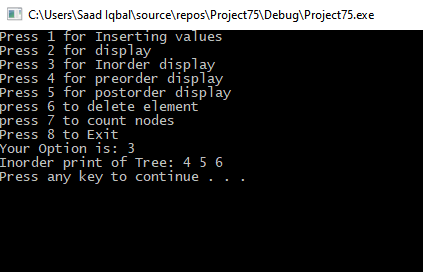
}

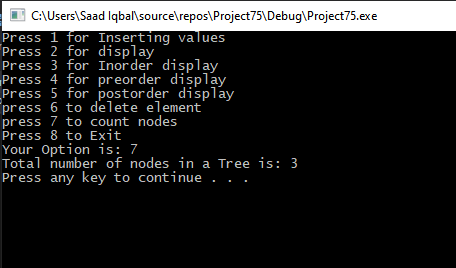
system("pause");

system("cls");

}

}

****

****

**TASK 3:**

#include <iostream>

using namespace std;

struct Node

{

int key, height;

Node\* L, \* R;

};

class Tree

{

private:

Node\* root;

public:

Tree() {

root = NULL;

};

Node\* getRoot()

{

return root;

}

Node\* insert(int num, Node\* temp)

{

if (temp == NULL)

{

temp = new Node;

temp->key = num;

temp->L = temp->R = NULL;

}

else if (num < temp->key)

{

temp->L = insert(num, temp->R);

}

else if (num > temp->key)

{

temp->R = insert(num, temp->R);

}

return temp;

}

//print display of tree

void display(Node\* r, int space)

{

if (r == NULL)

{

return;

}

display(r->R, space);

cout << endl;

cout << r->key << "\n";

display(r->L, space);

}

//print inorder of tree

void inorderPrint(Node\* T)

{

if (T == NULL)

return;

inorderPrint(T->L);

cout << T->key << " ";

inorderPrint(T->R);

}

//print preorder of tree

void print\_preorder(Node\* node)

{

if (node == NULL)

{

return;

}

else

{

cout << node->key << " ";

inorderPrint(node->L);

inorderPrint(node->R);

}

}

//print postorder of tree

void print\_postorder(Node\* node)

{

if (node == NULL)

{

return;

}

else

{

inorderPrint(node->L);

inorderPrint(node->R);

cout << node->key << " ";

}

}

Node\* minval(struct Node\* node)

{

Node\* temp = node;

while (temp && temp->L != NULL)

{

temp = temp->L;

}

return temp;

}

Node\* deleteNode( Node\* temp1, int key)

{

if (temp1 == NULL)

{

return temp1;

}

if (key < temp1->key)

{

temp1->L = deleteNode(temp1->L, key);

}

else if (key > temp1->key)

{

temp1->R = deleteNode(temp1->R, key);

}

else

{

if (temp1->L == NULL)

{

Node\* temp = temp1->R;

delete temp1;

return temp;

}

else if (temp1->R == NULL)

{

Node\* temp = temp1->L;

delete temp1;

return temp;

}

Node\* temp = minval(temp1->R);

temp1->key = temp->key;

temp1->R = deleteNode(temp1->R, temp->key);

}

return temp1;

}

int getLeafCount(Node\* Node)

{

if (Node == NULL)

{

return 0;

}

else if (Node->L == NULL && Node->R == NULL)

{

return 1;

}

else

{

return (getLeafCount(Node->L) + getLeafCount(Node->R));

}

}

};

int main()

{

Tree tree;

Node\* T = tree.getRoot();

int opt, num;

bool exit = false;

while (!exit)

{

cout << "Press 1 for Inserting values" << endl;

cout << "Press 2 for display" << endl;

cout << "Press 3 for Inorder display" << endl;

cout << "Press 4 for preorder display" << endl;

cout << "Press 5 for postorder display" << endl;

cout << "press 6 to delete element " << endl;

cout << "press 7 to count nodes " << endl;

cout << "Press 8 to Exit" << endl;

cout << "Your Option is: ";

cin >> opt;

switch (opt)

{

case 1:

cout << "Enter element: ";

cin >> num;

T = tree.insert( num,T);

break;

case 2:

cout << "Display of Tree: ";

tree.display(T,5);

cout << endl;

break;

case 3:

cout << "Inorder print of Tree: ";

tree.inorderPrint(T);

cout << endl;

break;

case 4:

cout << "Preorder print of Tree: ";

tree.print\_preorder(T);

cout << endl;

break;

case 5:

cout << "postorder print of Tree: ";

tree.print\_postorder(T);

cout << endl;

break;

case 6:

int a;

cout << "Enter element to delete :";

cin >> a;

tree.deleteNode(T,a);

cout << endl;

break;

case 7:

cout << "Total number of nodes in a Tree is: ";

cout<<tree.getLeafCount(T);

cout << endl;

case 8:

exit = true;

break;

default:

cout << "Option is invalid!" << endl;

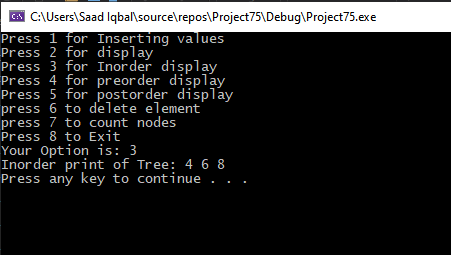
}

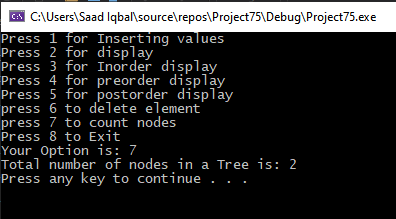
system("pause");

system("cls");

}

}

****

****

**TASK 4:**

#include <iostream>

using namespace std;

struct Node

{

int key, height;

Node\* L, \* R;

};

class Tree

{

private:

Node\* root;

public:

Tree() {

root = NULL;

};

Node\* getRoot()

{

return root;

}

Node\* insert(int num, Node\* temp)

{

if (temp == NULL)

{

temp = new Node;

temp->key = num;

temp->L = temp->R = NULL;

}

else if (num < temp->key)

{

temp->L = insert(num, temp->R);

}

else if (num > temp->key)

{

temp->R = insert(num, temp->R);

}

return temp;

}

//print display of tree

void display(Node\* r, int space)

{

if (r == NULL)

{

return;

}

display(r->L, space);

cout << r->key << "\n";

display(r->R, space);

return;

}

//print inorder of tree

void inorderPrint(Node\* T)

{

if (T == NULL)

return;

inorderPrint(T->L);

cout << T->key << " ";

inorderPrint(T->R);

}

//print preorder of tree

void print\_preorder(Node\* node)

{

if (node == NULL)

{

return;

}

else

{

cout << node->key << " ";

inorderPrint(node->L);

inorderPrint(node->R);

}

}

//print postorder of tree

void print\_postorder(Node\* node)

{

if (node == NULL)

{

return;

}

else

{

inorderPrint(node->L);

inorderPrint(node->R);

cout << node->key << " ";

}

}

Node\* minval(struct Node\* node)

{

Node\* temp = node;

while (temp && temp->L != NULL)

{

temp = temp->L;

}

return temp;

}

Node\* deleteNode(Node\* temp1, int key)

{

if (temp1 == NULL)

{

return temp1;

}

if (key < temp1->key)

{

temp1->L = deleteNode(temp1->L, key);

}

else if (key > temp1->key)

{

temp1->R = deleteNode(temp1->R, key);

}

else

{

if (temp1->L == NULL)

{

Node\* temp = temp1->R;

temp1->R = temp1->L;

delete temp1->L;

return temp;

}

else if (temp1->R == NULL)

{

Node\* temp = temp1->L;

delete temp1;

return temp;

}

Node\* temp = minval(temp1->R);

temp1->key = temp->key;

temp1->R = deleteNode(temp1->R, temp->key);

}

return temp1;

}

};

int main()

{

Tree tree;

Node\* T = tree.getRoot();

int opt, num;

bool exit = false;

while (!exit)

{

cout << "Press 1 for Inserting values" << endl;

cout << "Press 2 for display" << endl;

cout << "Press 3 for Inorder display" << endl;

cout << "Press 4 for preorder display" << endl;

cout << "Press 5 for postorder display" << endl;

cout << "press 6 to delete element " << endl;

cout << "Press 7 to Exit" << endl;

cout << "Your Option is: ";

cin >> opt;

switch (opt)

{

case 1:

cout << "Enter element: ";

cin >> num;

T = tree.insert(num, T);

break;

case 2:

cout << "Display of Tree: ";

tree.display(T, 5);

cout << endl;

break;

case 3:

cout << "Inorder print of Tree: ";

tree.inorderPrint(T);

cout << endl;

break;

case 4:

cout << "Preorder print of Tree: ";

tree.print\_preorder(T);

cout << endl;

break;

case 5:

cout << "postorder print of Tree: ";

tree.print\_postorder(T);

cout << endl;

break;

case 6:

int a;

cout << "Enter element to delete :";

cin >> a;

tree.deleteNode(T, a);

cout << endl;

break;

case 7:

exit = true;

break;

default:

cout << "Option is invalid!" << endl;

}

system("pause");

system("cls");

}

}

**TASK 5:**

#include <iostream>

using namespace std;

struct Node

{

int key, height;

Node\* L, \* R;

};

class Tree

{

private:

Node\* root;

public:

Tree() {

root = NULL;

};

Node\* getRoot()

{

return root;

}

Node\* insert(int num, Node\* temp)

{

if (temp == NULL)

{

temp = new Node;

temp->key = num;

temp->L = temp->R = NULL;

}

else if (num > temp->key)

{

temp->L = insert(num, temp->L);

}

else if (num < temp->key)

{

temp->R = insert(num, temp->L);

}

return temp;

}

//print display of tree

void display(Node\* r, int space)

{

if (r == NULL)

{

return;

}

display(r->L, space);

cout << r->key << "\n";

display(r->R, space);

return;

}

//print inorder of tree

void inorderPrint(Node\* T)

{

if (T == NULL)

return;

inorderPrint(T->L);

cout << T->key << " ";

inorderPrint(T->R);

}

//print preorder of tree

void print\_preorder(Node\* node)

{

if (node == NULL)

{

return;

}

else

{

cout << node->key << " ";

inorderPrint(node->L);

inorderPrint(node->R);

}

}

//print postorder of tree

void print\_postorder(Node\* node)

{

if (node == NULL)

{

return;

}

else

{

inorderPrint(node->L);

inorderPrint(node->R);

cout << node->key << " ";

}

}

Node\* minval(struct Node\* node)

{

Node\* temp = node;

while (temp && temp->L != NULL)

{

temp = temp->L;

}

return temp;

}

Node\* deleteNode( Node\* temp1, int key)

{

if (temp1 == NULL)

{

return temp1;

}

if (key < temp1->key)

{

temp1->L = deleteNode(temp1->L, key);

}

else if (key > temp1->key)

{

temp1->R = deleteNode(temp1->R, key);

}

else

{

if (temp1->L == NULL)

{

Node\* temp = temp1->R;

delete temp1;

return temp;

}

else if (temp1->R == NULL)

{

Node\* temp = temp1->L;

delete temp1;

return temp;

}

Node\* temp = minval(temp1->R);

temp1->key = temp->key;

temp1->R = deleteNode(temp1->R, temp->key);

}

return temp1;

}

int getLeafCount(Node\* Node)

{

if (Node == NULL)

{

return 0;

}

else if (Node->L == NULL && Node->R == NULL)

{

return 1;

}

else

{

return (getLeafCount(Node->L) + getLeafCount(Node->R));

}

}

};

int main()

{

Tree tree;

Node\* T = tree.getRoot();

int opt, num;

bool exit = false;

while (!exit)

{

cout << "Press 1 for Inserting values" << endl;

cout << "Press 2 for display" << endl;

cout << "Press 3 for Inorder display" << endl;

cout << "Press 4 for preorder display" << endl;

cout << "Press 5 for postorder display" << endl;

cout << "press 6 to delete element " << endl;

cout << "Press 7 to Exit" << endl;

cout << "Your Option is: ";

cin >> opt;

switch (opt)

{

case 1:

cout << "Enter element: ";

cin >> num;

T = tree.insert( num,T);

break;

case 2:

cout << "Display of Tree: ";

tree.display(T,5);

cout << endl;

break;

case 3:

cout << "Inorder print of Tree: ";

tree.inorderPrint(T);

cout << endl;

break;

case 4:

cout << "Preorder print of Tree: ";

tree.print\_preorder(T);

cout << endl;

break;

case 5:

cout << "postorder print of Tree: ";

tree.print\_postorder(T);

cout << endl;

break;

case 6:

int a;

cout << "Enter element to delete :";

cin >> a;

tree.deleteNode(T,a);

cout << endl;

break;

case 7:

exit = true;

break;

default:

cout << "Option is invalid!" << endl;

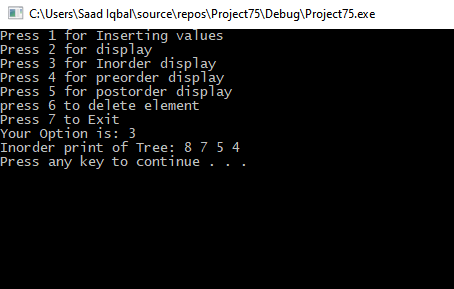
}

system("pause");

system("cls");

}

}

****

**TASK 6:**

#include <iostream>

using namespace std;

struct Node

{

int key, height;

Node\* L, \* R;

};

class Tree

{

private:

Node\* root;

public:

Tree() {

root = NULL;

};

Node\* getRoot()

{

return root;

}

Node\* insert(int num, Node\* temp)

{

if (temp == NULL)

{

temp = new Node;

temp->key = num;

temp->L = temp->R = NULL;

}

else if (num < temp->key)

{

temp->L = insert(num, temp->R);

}

else if (num > temp->key)

{

temp->R = insert(num, temp->R);

}

return temp;

}

//print display of tree

void display(Node\* r, int space)

{

if (r == NULL)

{

return;

}

display(r->L, space);

cout << r->key << "\n";

display(r->R, space);

return;

}

//calculating level and then level order traversal

bool printLevel(Node\* temp, int level)

{

if (temp == NULL)

{

return 0;

}

if (level == 1)

{

cout << temp->key << " ";

return 1;

}

bool left = printLevel(temp->L, level - 1);

bool right = printLevel(temp->R, level - 1);

return left || right;

}

void levelOrderTraversal(Node\* temp)

{

int level = 1;

while (printLevel(temp, level))

{

level++;

}

}

//print inorder of tree

void inorderPrint(Node\* T)

{

if (T == NULL)

return;

inorderPrint(T->L);

cout << T->key << " ";

inorderPrint(T->R);

}

//print preorder of tree

void print\_preorder(Node\* node)

{

if (node == NULL)

{

return;

}

else

{

cout << node->key << " ";

inorderPrint(node->L);

inorderPrint(node->R);

}

}

//print postorder of tree

void print\_postorder(Node\* node)

{

if (node == NULL)

{

return;

}

else

{

inorderPrint(node->L);

inorderPrint(node->R);

cout << node->key << " ";

}

}

Node\* minval(struct Node\* node)

{

Node\* temp = node;

while (temp && temp->L != NULL)

{

temp = temp->L;

}

return temp;

}

Node\* deleteNode( Node\* temp1, int key)

{

if (temp1 == NULL)

{

return temp1;

}

if (key < temp1->key)

{

temp1->L = deleteNode(temp1->L, key);

}

else if (key > temp1->key)

{

temp1->R = deleteNode(temp1->R, key);

}

else

{

if (temp1->L == NULL)

{

Node\* temp = temp1->R;

delete temp1;

return temp;

}

else if (temp1->R == NULL)

{

Node\* temp = temp1->L;

delete temp1;

return temp;

}

Node\* temp = minval(temp1->R);

temp1->key = temp->key;

temp1->R = deleteNode(temp1->R, temp->key);

}

return temp1;

}

int getLeafCount(Node\* Node)

{

if (Node == NULL)

{

return 0;

}

else if (Node->L == NULL && Node->R == NULL)

{

return 1;

}

else

{

return (getLeafCount(Node->L) + getLeafCount(Node->R));

}

}

};

int main()

{

Tree tree;

Node\* T = tree.getRoot();

int opt, num;

bool exit = false;

while (!exit)

{

cout << "Press 1 for Inserting values" << endl;

cout << "Press 2 for display" << endl;

cout << "Press 3 for Inorder display" << endl;

cout << "Press 4 for preorder display" << endl;

cout << "Press 5 for postorder display" << endl;

cout << "press 6 to delete element " << endl;

cout << "press 7 for level order traversal " << endl;

cout << "Press 8 to Exit" << endl;

cout << "Your Option is: ";

cin >> opt;

switch (opt)

{

case 1:

cout << "Enter element: ";

cin >> num;

T = tree.insert( num,T);

break;

case 2:

cout << "Display of Tree: ";

tree.display(T,5);

cout << endl;

break;

case 3:

cout << "Inorder print of Tree: ";

tree.inorderPrint(T);

cout << endl;

break;

case 4:

cout << "Preorder print of Tree: ";

tree.print\_preorder(T);

cout << endl;

break;

case 5:

cout << "postorder print of Tree: ";

tree.print\_postorder(T);

cout << endl;

break;

case 6:

int a;

cout << "Enter element to delete :";

cin >> a;

tree.deleteNode(T,a);

cout << endl;

break;

case 7:

cout << "Level order traversal is :";

tree.levelOrderTraversal(T);

break;

case 8:

exit = true;

break;

default:

cout << "Option is invalid!" << endl;

}

system("pause");

system("cls");

}

}

