# Department of Computing

# School of Electrical Engineering and Computer Science

**CS - 250: Data Structure and Algorithms**

**Class: BSCS 10AB**

**Lab 10 : Binary Search Tree – Part B**

**Date: 14th December, 2021**

**Time: 10:00 am – 12:50 pm   
&  
 02:00 pm – 4:50 pm**

# Instructor: Dr. Yasir Faheem

# Lab Engineer: Aftab Farooq

# Mashal Ashfaque

# 337203

# Lab 10 : Implementation of Binary Search Trees – Part B

**Introduction**

This lab is based on the implementation of Binary Search tree and its functions.

**Objectives**

The objectives of this lab are the following:

* Become familiar with implementation of binary search trees
* Study some statistics of binary search trees
* Write simple applications using binary search tree

**Tools/Software Requirement**

Visual Studio 2012 or gcc or g++

**Description**

In computer science, a binary search tree (BST), which may sometimes also be called an ordered or sorted binary tree, is a node-based binary tree data structure which has the following properties:

* The left sub-tree of a node contains only nodes with keys less than the node's key.
* The right sub-tree of a node contains only nodes with keys greater than the node's key.
* Both the left and right sub-trees must also be binary search trees.
* There must be no duplicate nodes.

In this lab, you will expand implement binary search tree, study some statistical properties of BST and write a simple application using the BST.

Here is a template of how your class/structure looks like.

class BST\_Node{

Template data;

BST\_Node \*LeftChild;

BST\_Node \*RightChild;

};

**Lab Task**

**Tasks**

**Previous Week’s Tasks:** In the previous lab you implemented the following operation of a binary search tree:

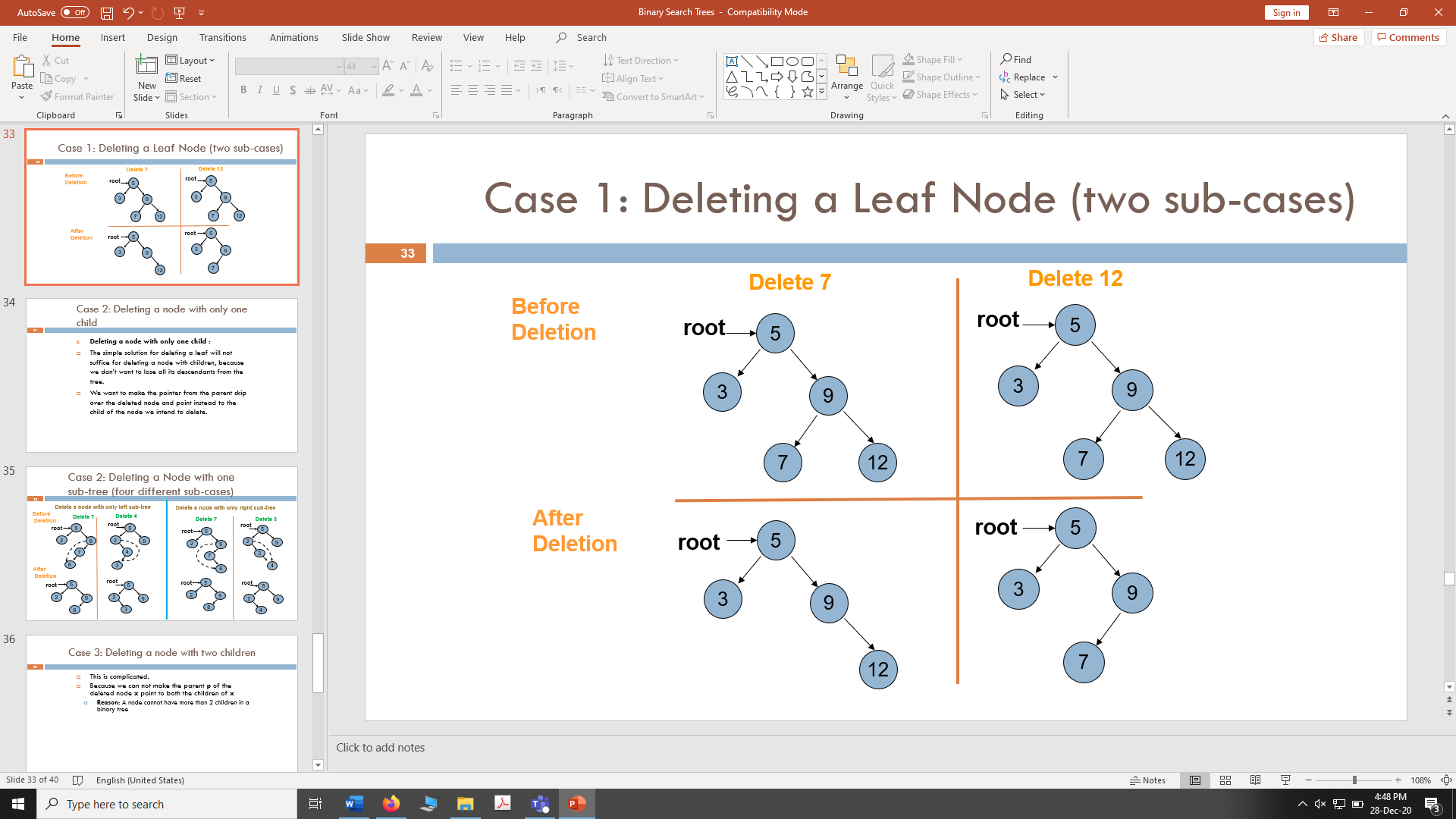
* bool IsEmpty();
* void Search(template value)
* Void InsertWithoutDuplication(template value)
* Void InsertWithDuplication(template value)
* Traversing a binary tree in pre-order, in-order and post-order.
* Implement a function that prints the smallest value of a BST.
* Implement a function that prints the largest value of a BST.
* Implement a function to calculate the height of a BST.
* Implement a function that calculates the depth of a BST.

**In today’s lab,** your task is to implement the following operations of a binary search tree:

Deleting a node. You may call your search(value) function. You should implement all the three cases:

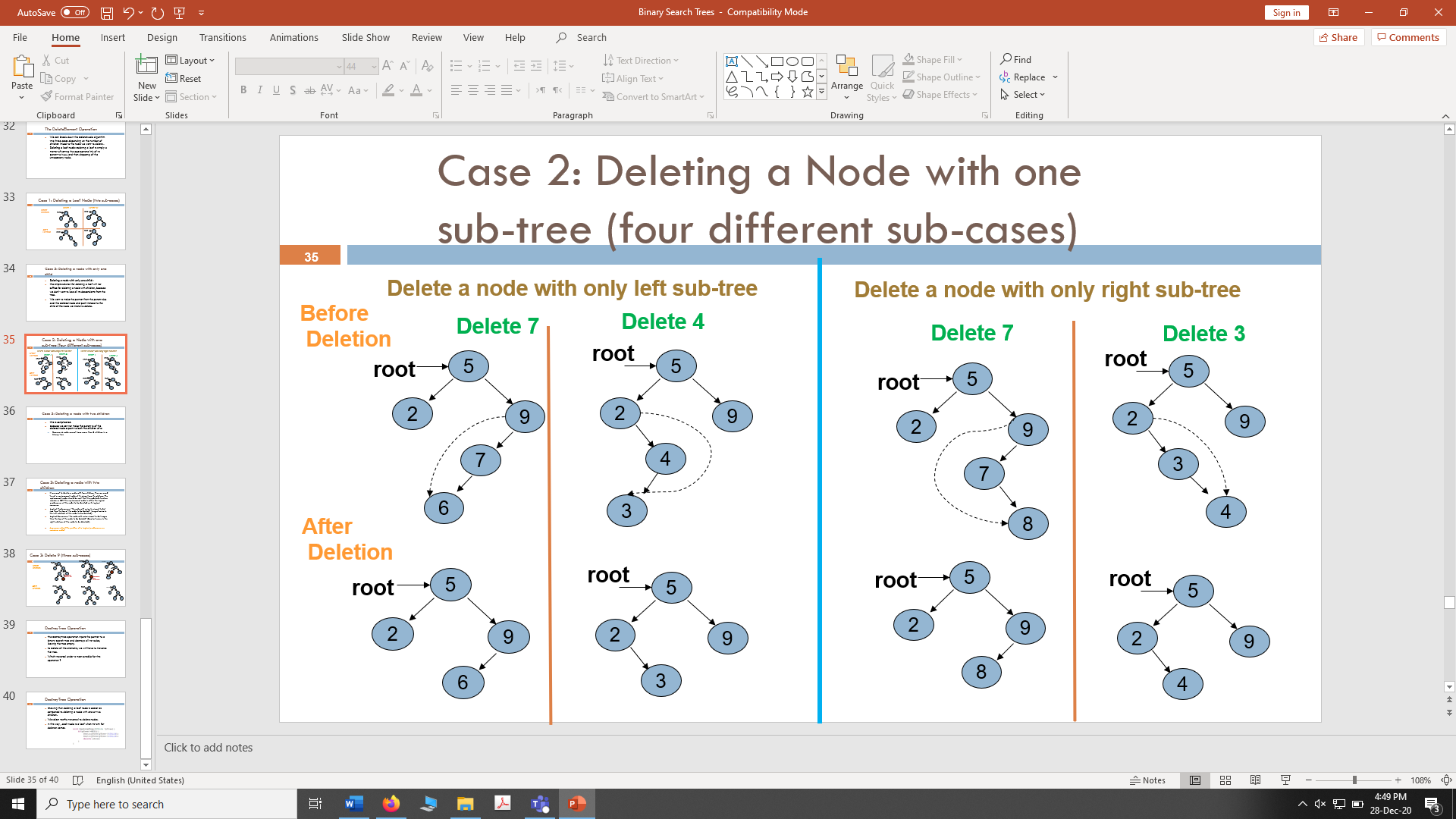
**a).** **Deleting a leaf node.**

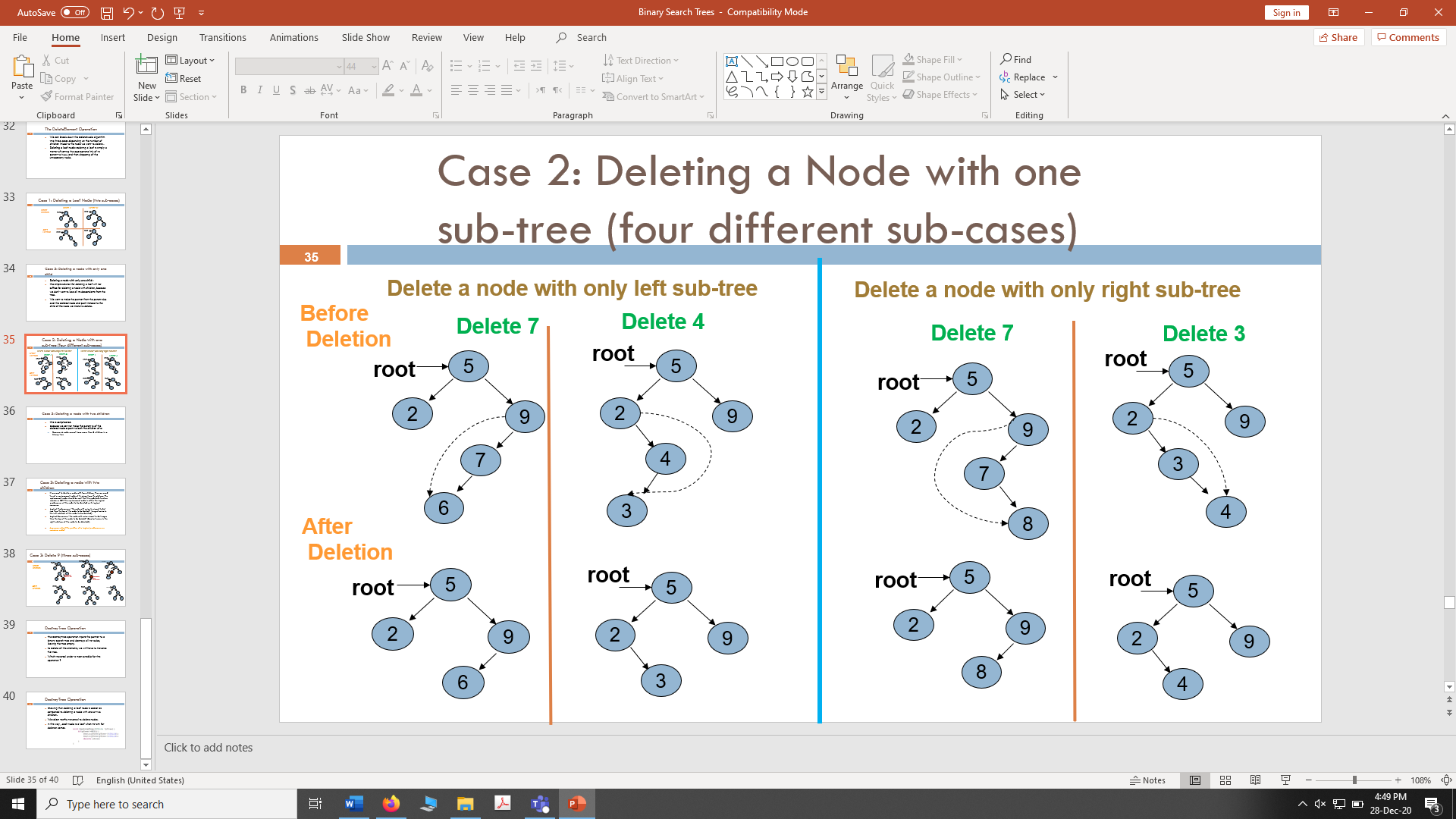
Include the case in which the node being deleted is a root node.



**b). Deleting a node with only one sub-tree.**

Include the case in which the node being deleted is a root node.



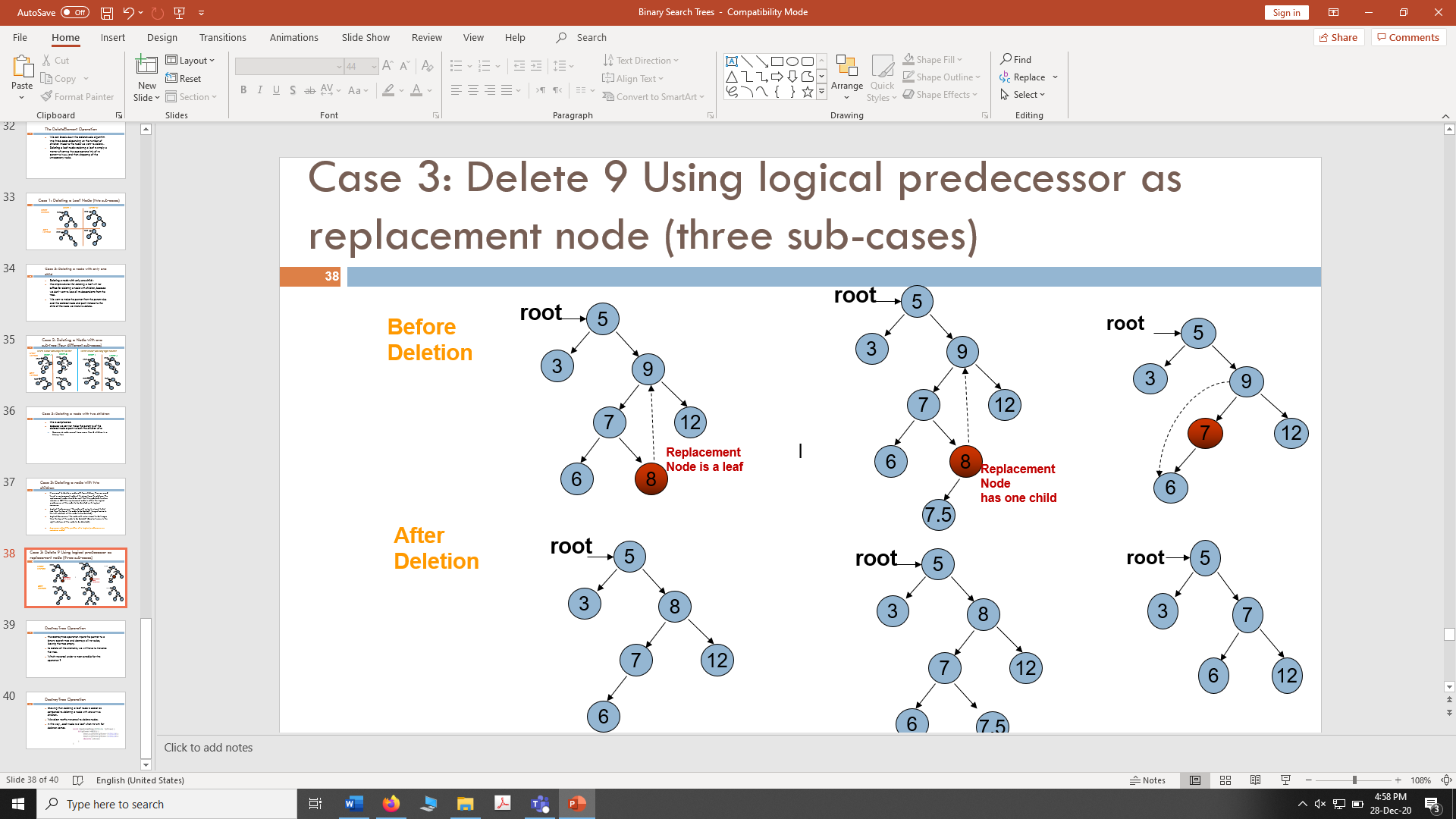


**c).** **Deleting a node with two children:** If we want to delete a node with two children, then we need to put a replacement node at its place from its sub-tree. The replacement node should be such that the updated structure remains a BST. The replacement node is either the logical predecessor of the node to be deleted or its logical successor.

**Logical Predecessor:** ”*the node with value to closest to but less than the key of the node to be deleted*” (largest value in the left sub-tree of the node to be deleted).

**Logical Successor:** ”*the node with value closest to but larger than the key of the node to be deleted*” (Smallest value in the right sub-tree of the node to be deleted).

In the below example, the node containing value 9 is deleted using its logical predecessor as replacement node. Include the case in which the node being deleted is a root node.



**Having all 3 delete cases:**

#include<iostream>

using namespace std;

class BTNode

{

public:

int data;

BTNode\* lchild;

BTNode\* rchild;

};

class BST

{

public:

BTNode\* root;

BTNode\* loc = NULL;

BTNode\* ploc = NULL;

BST()

{

root = NULL;

}

bool isEmpty()

{

return root == NULL;

}

void Search(int value)

{

ploc = NULL;

loc = root;

if (!isEmpty())

{

while (loc != NULL && loc->data != value)

{

ploc = loc;

if (value < loc->data)

loc = loc->lchild;

else

loc = loc->rchild;

} // end while

}//end if

}//end search

void InsertValue(int value)

{

if (isEmpty())

{

root = new BTNode();

root->data = value;

}

else

{

Search(value);

if (loc != NULL)

cout << "Duplication not allowed";

else

{

BTNode\* nn = new BTNode();

nn->data = value;

if (value < ploc->data)

ploc->lchild = nn;

else

ploc->rchild = nn;

} //end else

} //end outer else

} //end insertvalue

void PreOrder(BTNode\* ptree)

{

if (ptree != NULL)

{

cout << ptree->data << " ";

PreOrder(ptree->lchild);

PreOrder(ptree->rchild);

}

}

void DeleteValue(int value)

{

Search(value);

if (loc != NULL)

{

//check if loc is a leaf node

if (loc->lchild == NULL && loc->rchild == NULL)

{

if (ploc == NULL)

root = NULL;

else if (ploc->lchild == loc)

ploc->lchild = NULL;

else

ploc->rchild = NULL;

delete loc;

}

//check if deleting a node with left branch

else if (loc->lchild != NULL && loc->rchild == NULL)

{

if (ploc == NULL)

root = loc->lchild;

else if (ploc->lchild == loc)

ploc->lchild = loc->lchild;

else

ploc->rchild = loc->lchild;

delete loc;

}

//check if deleting a node with right branch

else if (loc->lchild == NULL && loc->rchild != NULL)

{

if (ploc == NULL)

root = loc->rchild;

else if (ploc->rchild == loc)

ploc->rchild = loc->rchild;

else

ploc->lchild = loc->rchild;

delete loc;

}

//check if deleting a node with 2 children

else

{

BTNode\* tploc = ploc;

BTNode\* tloc = loc;

ploc = loc; loc = loc->lchild;

while (loc->rchild != NULL)

{

ploc = loc; loc = loc->rchild;

}

if (loc == tloc->lchild)

tloc->lchild = loc->lchild;

else if (loc->lchild == NULL)

ploc->rchild = NULL;

else if (loc->lchild != NULL)

ploc->rchild = loc->lchild;

loc->lchild = tloc->lchild;

loc->rchild = tloc->rchild;

if (tploc == NULL)

root = NULL;

else if (tploc->rchild == tloc)

tploc->rchild = loc;

else

tploc->lchild = loc;

delete tloc;

}

}

else

{

cout << "Value not found. Deletion not possible!\n";

}

}

};

int main()

{

BST\* tree = new BST();

//int value;

int last;

cout << "Insert some elements for tree. Press any 0 to stop inserting." << endl;

cin >> last;

while (last != 0)

{

tree->InsertValue(last);

cin >> last;

}

int value;

cout << "Before deleting value: ";

tree->PreOrder(tree->root);

cout << "\nEnter value to be deleted.";

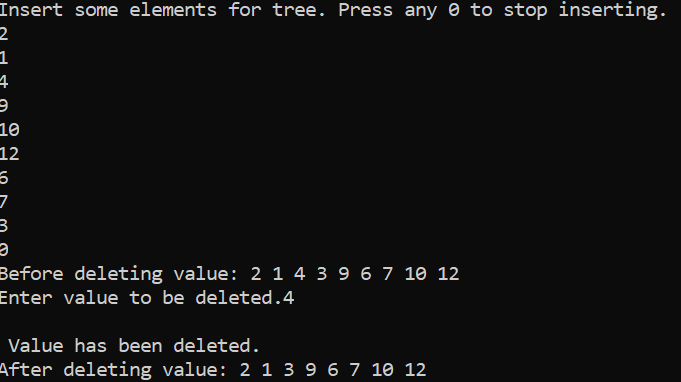
cin >> value;

tree->DeleteValue(value);

cout<<"\n Value has been deleted.\nAfter deleting value: ";

tree->PreOrder(tree->root);

}



1. **Destroy tree:** Implement a function that destroys all nodes of a binary tree leaving the tree empty.

#include<iostream>

using namespace std;

class BTNode

{

public:

int data;

BTNode\* lchild;

BTNode\* rchild;

};

class BST

{

public:

BTNode\* root;

BTNode\* loc = NULL;

BTNode\* ploc = NULL;

BST()

{

root = NULL;

}

bool isEmpty()

{

return root == NULL;

}

void Search(int value)

{

ploc = NULL;

loc = root;

if (!isEmpty())

{

while (loc != NULL && loc->data != value)

{

ploc = loc;

if (value < loc->data)

loc = loc->lchild;

else

loc = loc->rchild;

} // end while

}//end if

}//end search

void InsertValue(int value)

{

if (isEmpty())

{

root = new BTNode();

root->data = value;

}

else

{

Search(value);

if (loc != NULL)

cout << "Duplication not allowed";

else

{

BTNode\* nn = new BTNode();

nn->data = value;

if (value < ploc->data)

ploc->lchild = nn;

else

ploc->rchild = nn;

} //end else

} //end outer else

} //end insertvalue

void PreOrder(BTNode\* ptree)

{

if (ptree != NULL)

{

cout << ptree->data << " ";

PreOrder(ptree->lchild);

PreOrder(ptree->rchild);

}

}

void DestroyTree(BTNode\* pTree)

{

if (pTree != NULL)

{

DestroyTree(pTree->lchild);

DestroyTree(pTree->rchild);

delete pTree;

}

}

};

int main()

{

BST\* tree = new BST();

//int value;

int last;

cout << "Insert some elements for tree. Press any 0 to stop inserting." << endl;

cin >> last;

while (last != 0)

{

tree->InsertValue(last);

cin >> last;

}

tree->DestroyTree(tree->root);

tree->root = NULL;

cout << "Tree has been destroyed.";

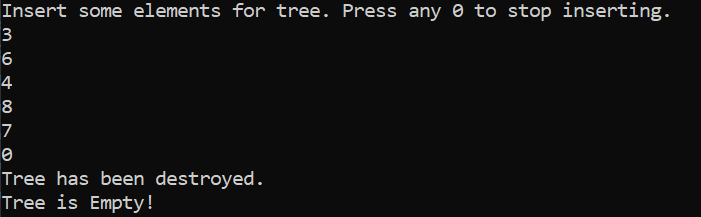
if (tree->isEmpty())

cout << "\nTree is Empty!";

else

cout << "Tree is Not empty!";

}



1. Implement a function that traverses a BST and **prints** only its **leaf nodes.**

#include<iostream>

using namespace std;

class BTNode

{

public:

int data;

BTNode\* lchild;

BTNode\* rchild;

};

class BST

{

public:

BTNode\* root;

BTNode\* loc = NULL;

BTNode\* ploc = NULL;

BST()

{

root = NULL;

}

bool isEmpty()

{

return root == NULL;

}

void Search(int value)

{

ploc = NULL;

loc = root;

if (!isEmpty())

{

while (loc != NULL && loc->data != value)

{

ploc = loc;

if (value < loc->data)

loc = loc->lchild;

else

loc = loc->rchild;

} // end while

}//end if

}//end search

void InsertValue(int value)

{

if (isEmpty())

{

root = new BTNode();

root->data = value;

}

else

{

Search(value);

if (loc != NULL)

cout << "Duplication not allowed";

else

{

BTNode\* nn = new BTNode();

nn->data = value;

if (value < ploc->data)

ploc->lchild = nn;

else

ploc->rchild = nn;

} //end else

} //end outer else

} //end insertvalue

void PreOrder(BTNode\* ptree)

{

if (ptree != NULL)

{

cout << ptree->data << " ";

PreOrder(ptree->lchild);

PreOrder(ptree->rchild);

}

}

void printleafnodes(BTNode\* loc)

{

if (loc->lchild == NULL && loc->rchild == NULL)

cout << loc->data << " ";

else

{

if (loc->lchild != NULL)

printleafnodes(loc->lchild);

if (loc->rchild != NULL)

printleafnodes(loc->rchild);

}

}

};

int main()

{

BST\* tree = new BST();

//int value;

int last;

cout << "Insert some elements for tree. Press any 0 to stop inserting." << endl;

cin >> last;

while (last != 0)

{

tree->InsertValue(last);

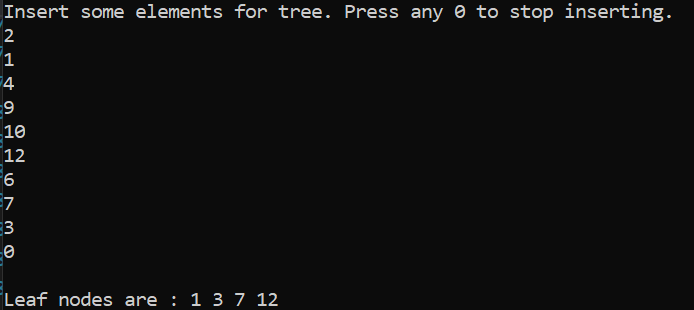
cin >> last;

}

cout << "\nLeaf nodes are : ";

tree->printleafnodes(tree->root);

}



1. Implement a function that traverses all nodes of a BST once, and counts the number of leaves, nodes with only left child, nodes with only right child and nodes with two children in it. You should maintain a separate counter variable for each of the four types of nodes.

#include<iostream>

using namespace std;

class BTNode

{

public:

int data;

BTNode\* lchild;

BTNode\* rchild;

};

class BST

{

public:

BTNode\* root;

BTNode\* loc = NULL;

BTNode\* ploc = NULL;

BST()

{

root = NULL;

}

bool isEmpty()

{

return root == NULL;

}

void Search(int value)

{

ploc = NULL;

loc = root;

if (!isEmpty())

{

while (loc != NULL && loc->data != value)

{

ploc = loc;

if (value < loc->data)

loc = loc->lchild;

else

loc = loc->rchild;

} // end while

}//end if

}//end search

void InsertValue(int value)

{

if (isEmpty())

{

root = new BTNode();

root->data = value;

}

else

{

Search(value);

if (loc != NULL)

cout << "Duplication not allowed";

else

{

BTNode\* nn = new BTNode();

nn->data = value;

if (value < ploc->data)

ploc->lchild = nn;

else

ploc->rchild = nn;

} //end else

} //end outer else

} //end insertvalue

void PreOrder(BTNode\* ptree)

{

if (ptree != NULL)

{

cout << ptree->data << " ";

PreOrder(ptree->lchild);

PreOrder(ptree->rchild);

}

}

int CountLeafNodes(BTNode\* loc)

{

if (loc == NULL)

return 0;

if (loc->lchild == NULL && loc->rchild == NULL)

return 1;

else

return CountLeafNodes(loc->lchild) +

CountLeafNodes(loc->rchild);

}

int countLeftNodes(BTNode\* left)

{

if (left == NULL)

return 0;

// left-child count of current node.

int cnt = 0;

// does the current node have a left-child

if (left->lchild != NULL)

cnt = 1;

// return left-child count of current node +

// left-child count of left and right subtrees

return cnt + countLeftNodes(left->lchild) + countLeftNodes(left->rchild);

}

int countRightNodes(BTNode\* right)

{

if (right == NULL)

return 0;

// right-child count of current node.

int cnt1 = 0;

// does the current node have a right-child

if (right->rchild != NULL)

cnt1 = 1;

// return right-child count of current node +

// left-child count of left and right subtrees

return cnt1 + countRightNodes(right->lchild) + countRightNodes(right->rchild);

}

int Nodeswith2children(BTNode\* Node)

{

if (Node == NULL)

return 0;

int cnt = 0;

// does the current node have a left-child and the right child

if (Node->lchild != NULL && Node->rchild != NULL)

cnt = 1;

// return both-child count of current node +

// both-child count of left and right subtrees

return cnt + Nodeswith2children(Node->lchild) + Nodeswith2children(Node->rchild);

}

};

int main()

{

BST\* tree = new BST();

//int value;

int last;

cout << "Insert some elements for tree. Press any 0 to stop inserting." << endl;

cin >> last;

while (last != 0)

{

tree->InsertValue(last);

cin >> last;

}

int leafcount = tree->CountLeafNodes(tree->root);

cout << "\nTotal leaf nodes are: " << leafcount;

cout << "\nNumber of nodes with only left child are : " ;

cout<< tree->countLeftNodes(tree->root);

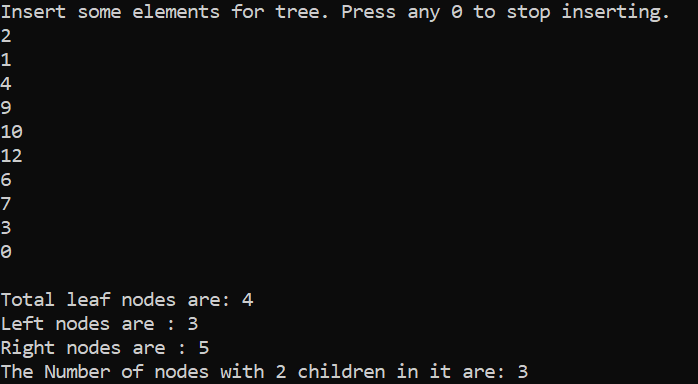
cout << "\nNumber of nodes with only right child are : ";

cout << tree->countRightNodes(tree->root);

cout << "\nThe Number of nodes with 2 children in it are: ";

cout << tree->Nodeswith2children(tree->root);

}



1. Implement a function that **deletes** all **leaf** nodes of a **given BST only.**

#include<iostream>

using namespace std;

class BTNode

{

public:

int data;

BTNode\* lchild;

BTNode\* rchild;

};

class BST

{

public:

BTNode\* root;

BTNode\* loc = NULL;

BTNode\* ploc = NULL;

BST()

{

root = NULL;

}

bool isEmpty()

{

return root == NULL;

}

void Search(int value)

{

ploc = NULL;

loc = root;

if (!isEmpty())

{

while (loc != NULL && loc->data != value)

{

ploc = loc;

if (value < loc->data)

loc = loc->lchild;

else

loc = loc->rchild;

} // end while

}//end if

}//end search

void InsertValue(int value)

{

if (isEmpty())

{

root = new BTNode();

root->data = value;

}

else

{

Search(value);

if (loc != NULL)

cout << "Duplication not allowed";

else

{

BTNode\* nn = new BTNode();

nn->data = value;

if (value < ploc->data)

ploc->lchild = nn;

else

ploc->rchild = nn;

} //end else

} //end outer else

} //end insertvalue

void PreOrder(BTNode\* ptree)

{

if (ptree != NULL)

{

cout << ptree->data << " ";

PreOrder(ptree->lchild);

PreOrder(ptree->rchild);

}

}

void DeleteValue(int value)

{

Search(value);

if (loc != NULL)

{

//check if loc is a leaf node

if (loc->lchild == NULL && loc->rchild == NULL)

{

if (ploc == NULL)

root = NULL;

else if (ploc->lchild == loc)

ploc->lchild = NULL;

else

ploc->rchild = NULL;

delete loc;

}

//check if deleting a node with left branch

else if (loc->lchild != NULL && loc->rchild == NULL)

{

if (ploc == NULL)

root = loc->lchild;

else if (ploc->lchild == loc)

ploc->lchild = loc->lchild;

else

ploc->rchild = loc->lchild;

delete loc;

}

//check if deleting a node with right branch

else if (loc->lchild == NULL && loc->rchild != NULL)

{

if (ploc == NULL)

root = loc->rchild;

else if (ploc->rchild == loc)

ploc->rchild = loc->rchild;

else

ploc->lchild = loc->rchild;

delete loc;

}

//check if deleting a node with 2 children

else

{

BTNode\* tploc = ploc;

BTNode\* tloc = loc;

ploc = loc; loc = loc->lchild;

while (loc->rchild != NULL)

{

ploc = loc; loc = loc->rchild;

}

if (loc == tloc->lchild)

tloc->lchild = loc->lchild;

else if (loc->lchild == NULL)

ploc->rchild = NULL;

else if (loc->lchild != NULL)

ploc->rchild = loc->lchild;

loc->lchild = tloc->lchild;

loc->rchild = tloc->rchild;

if (tploc == NULL)

root = NULL;

else if (tploc->rchild == tloc)

tploc->rchild = loc;

else

tploc->lchild = loc;

delete loc;

}

}

else

{

cout << "Value not found. Deletion not possible!\n";

}

}

void DeleteLeafNodes(BTNode\* Node) {

if (Node != NULL)

{

if (Node->lchild == NULL && Node->rchild == NULL)

DeleteValue(Node->data);

else //recursive case

{

DeleteLeafNodes(Node->lchild);

DeleteLeafNodes(Node->rchild);

}

}

}

};

int main()

{

BST\* tree = new BST();

//int value;

int last;

cout << "Insert some elements for tree. Press any 0 to stop inserting." << endl;

cin >> last;

while (last != 0)

{

tree->InsertValue(last);

cin >> last;

}

cout << "Nodes before deleting leaf nodes: ";

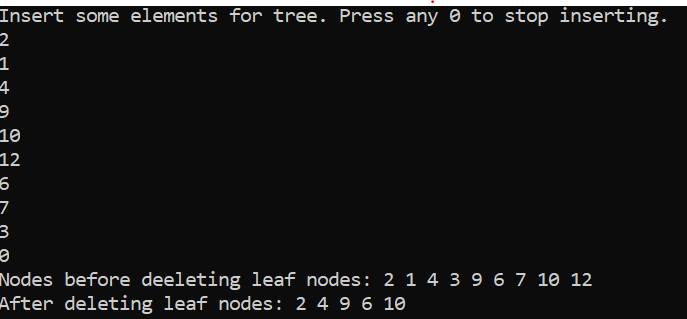
tree->PreOrder(tree->root);

tree->DeleteLeafNodes(tree->root);

cout << "\nAfter deleting leaf nodes: ";

tree->PreOrder(tree->root);

}



1. Implement a function that **deletes** only those nodes from a **given** **BST** that have only **left sub-tree.** Only those nodes should be deleted that have left branch in the original tree.

Code:

#include<iostream>

using namespace std;

class BTNode

{

public:

int data;

BTNode\* lchild;

BTNode\* rchild;

};

class BST

{

public:

BTNode\* root;

BTNode\* loc = NULL;

BTNode\* ploc = NULL;

BST()

{

root = NULL;

}

bool isEmpty()

{

return root == NULL;

}

void Search(int value)

{

ploc = NULL;

loc = root;

if (!isEmpty())

{

while (loc != NULL && loc->data != value)

{

ploc = loc;

if (value < loc->data)

loc = loc->lchild;

else

loc = loc->rchild;

} // end while

}//end if

}//end search

void InsertValue(int value)

{

if (isEmpty())

{

root = new BTNode();

root->data = value;

}

else

{

Search(value);

if (loc != NULL)

cout << "Duplication not allowed";

else

{

BTNode\* nn = new BTNode();

nn->data = value;

if (value < ploc->data)

ploc->lchild = nn;

else

ploc->rchild = nn;

} //end else

} //end outer else

} //end insertvalue

void PreOrder(BTNode\* ptree)

{

if (ptree != NULL)

{

cout << ptree->data << " ";

PreOrder(ptree->lchild);

PreOrder(ptree->rchild);

}

}

void DeleteValue(int value)

{

Search(value);

if (loc != NULL)

{

//check if loc is a leaf node

if (loc->lchild == NULL && loc->rchild == NULL)

{

if (ploc == NULL)

root = NULL;

else if (ploc->lchild == loc)

ploc->lchild = NULL;

else

ploc->rchild = NULL;

delete loc;

}

//check if deleting a node with left branch

else if (loc->lchild != NULL && loc->rchild == NULL)

{

if (ploc == NULL)

root = loc->lchild;

else if (ploc->lchild == loc)

ploc->lchild = loc->lchild;

else

ploc->rchild = loc->lchild;

delete loc;

}

//check if deleting a node with right branch

else if (loc->lchild == NULL && loc->rchild != NULL)

{

if (ploc == NULL)

root = loc->rchild;

else if (ploc->rchild == loc)

ploc->rchild = loc->rchild;

else

ploc->lchild = loc->rchild;

delete loc;

}

//check if deleting a node with 2 children

else

{

BTNode\* tploc = ploc;

BTNode\* tloc = loc;

ploc = loc; loc = loc->lchild;

while (loc->rchild != NULL)

{

ploc = loc; loc = loc->rchild;

}

if (loc == tloc->lchild)

tloc->lchild = loc->lchild;

else if (loc->lchild == NULL)

ploc->rchild = NULL;

else if (loc->lchild != NULL)

ploc->rchild = loc->lchild;

loc->lchild = tloc->lchild;

loc->rchild = tloc->rchild;

if (tploc == NULL)

root = NULL;

else if (tploc->rchild == tloc)

tploc->rchild = loc;

else

tploc->lchild = loc;

delete tloc;

}

}

else

{

cout << "Value not found. Deletion not possible!\n";

}

}

void DeleteLeftBranchNodes(BTNode\* loc)

{

if (loc != NULL)

{

if (loc->lchild != NULL && loc->rchild == NULL)

DeleteValue(loc->data);

else

{

DeleteLeftBranchNodes(loc->lchild);

DeleteLeftBranchNodes(loc->rchild);

}

}

}

};

int main()

{

BST\* tree = new BST();

//int value;

int last;

cout << "Insert some elements for tree. Press any 0 to stop inserting." << endl;

cin >> last;

while (last != 0)

{

tree->InsertValue(last);

cin >> last;

}

cout << "Nodes before deleting nodes with left branch: ";

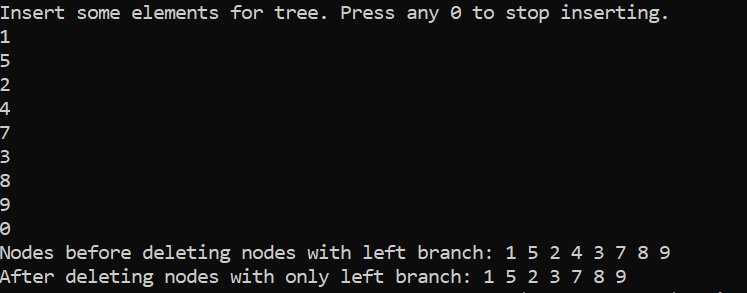
tree->PreOrder(tree->root);

tree->DeleteLeftBranchNodes(tree->root);

cout << "\nAfter deleting nodes with only left branch: ";

tree->PreOrder(tree->root);

}



**Deliverables:**

Compile a single word document by filling in the solution part and submit this Word file on LMS. The name of word document should follow this format. i.e. **YourFullName(reg)\_Lab#.** This lab grading policy is as follows: The lab is graded between 0 to 10 marks. The submitted solution can get a maximum of 5 marks. At the end of each lab or in the next lab, there will be a viva related to the tasks. The viva has a weightage of 5 marks. Insert the solution/answer in this document. You must show the implementation of the tasks in the designing tool, along with your complete Word document to get your work graded. You must also submit this Word document on the LMS. In case of any problems discuss it by emailing it to [aftab.farooq@seecs.edu.pk](mailto:aftab.farooq@seecs.edu.pk).

**Note:** Students are required to upload the lab on LMS before deadline.

Use proper indentation and comments. Lack of comments and indentation will result in deduction of marks.