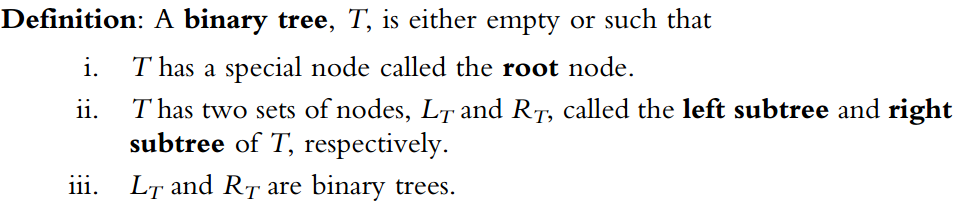
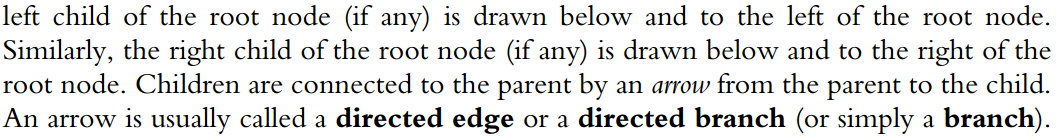
**Why Binary Trees?**

* Array -> Insertion and Deletion requires movement of data
* Linked List -> Must be processed sequentially

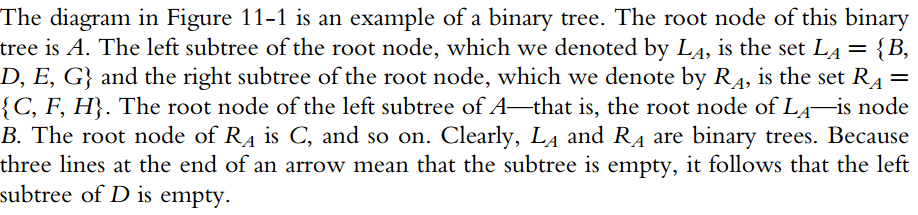
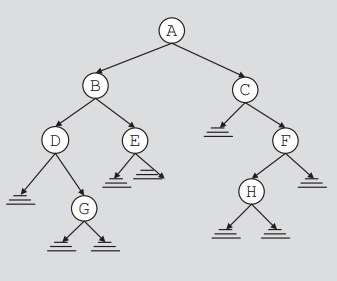


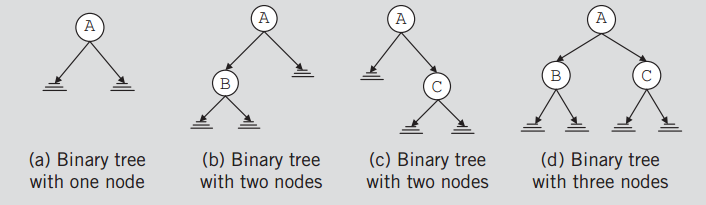


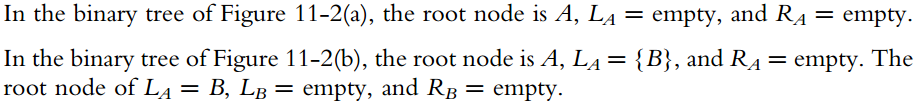


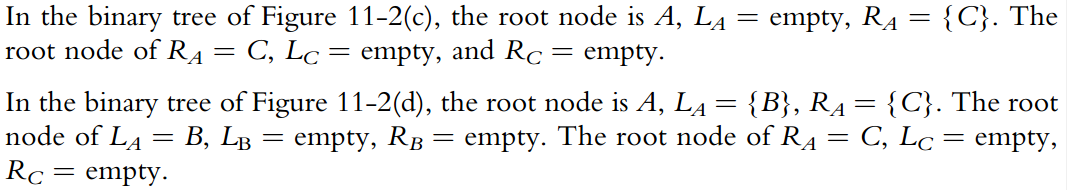


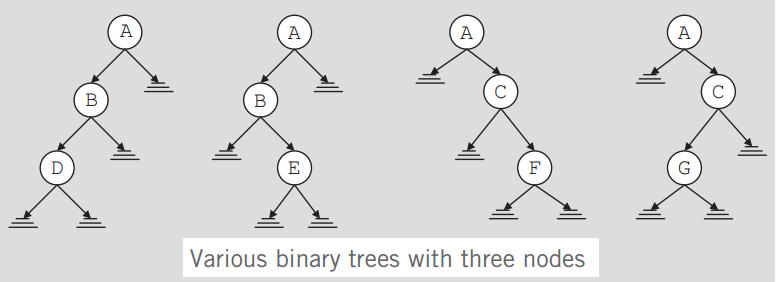








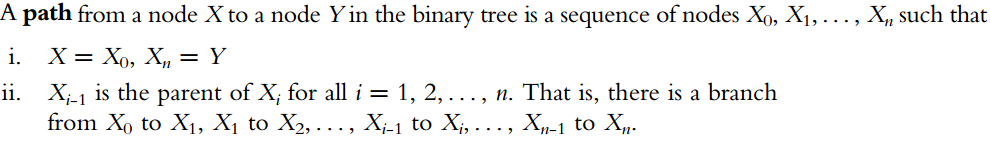


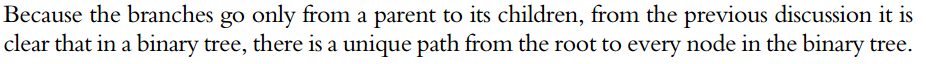


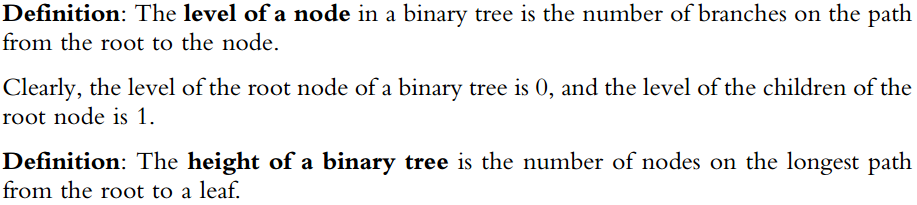
**IMPORTANT DISCUSSION**



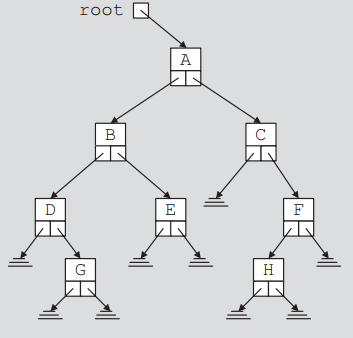








**Implementation:**

Each Node:

1. Stores its own information
2. llink points to the root of the left sub-tree
3. rlink points to the root of the right sub-tree

**Step 1: Create a class (struct) for the Node of the tree**

template <class T>

class BinaryTreeNode

{

public:

T info;

BinaryTreeNode \*llink;

BinaryTreeNode \*rlink;

};

**Step 2: Create the ADT [Binary Tree Type].**

#pragma once

#include "BinaryTreeNode.h"

#include <iostream>

#include <string>

using namespace std;

template <class T>

class binaryTreeType

{

public:

int height(BinaryTreeNode<T>\*p)

{

if (p == NULL)

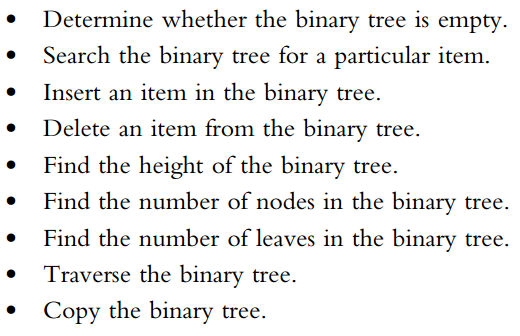
return 0;

else

return 1 + max(height(p->llink), height(p->rlink));

}

int max(int x, int y) const

 {

if (x >= y)

return x;

else

return y;

}

void inorder(BinaryTreeNode<T> \*p) const

{

if (p != NULL)

{

inorder(p->llink);

cout << p->info;

inorder(p->rlink);

}

}

void preorder(BinaryTreeNode<T> \*p) const

{

if (p != NULL)

{

cout << p->info;

preorder(p->llink);

preorder(p->rlink);

}

}

void postorder(BinaryTreeNode<T> \*p) const

{

if (p != NULL)

{

postorder(p->llink);

postorder(p->rlink);

cout << p->info;

}

}

int treeNodeCount(BinaryTreeNode<T>\*p)

{

static int count = 0;

if (p != NULL)

{

treeNodeCount(p->llink);

treeNodeCount(p->rlink);

count++;

}

return count;

}

int treeLeafCount(BinaryTreeNode<T>\*p)

{

static int count = 0;

if (p != NULL)

{

treeLeafCount(p->llink);

treeLeafCount(p->rlink);

if (p->llink == NULL && p->rlink == NULL)

count++;

}

return count;

}

//void destroy(BinaryTreeNode<T>\*&p);

protected:

BinaryTreeNode<T> \*root;

public:

bool isEmpty()

{

return (root == NULL)

}

binaryTreeType()

{

root = NULL;

}

~binaryTreeType(){}

/\*

int treeLeafCount()

{

}

void inorderTraversal() const;

void preorderTraversal() const;

void postorderTraversal() const;

int treeHeight();

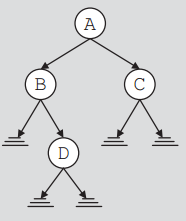
int treeNodeCount();

void destroyTree();

\*/

};

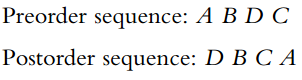
**Tree Traversal**

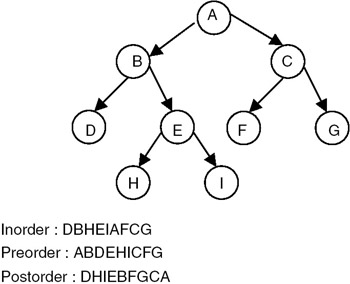
INORDER: LNR

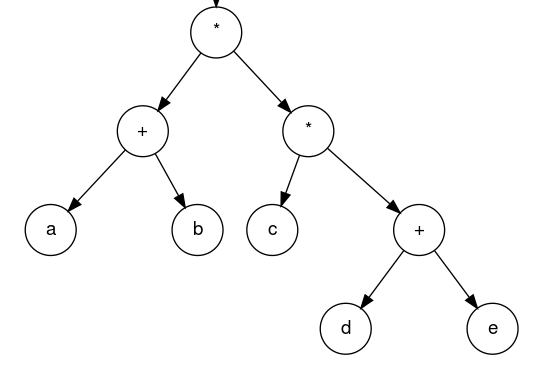
PREORDER: NLR

POSTORDER: LRN









**Postorder: LRN**

**ab+cde+\*\***

**Inorder: LNR**

**(a+b)\*c\*(d+e)**

**Preorder: NLR**

**\*+ab\*c+de**

**Complete Code:**

#pragma once

#include "BinaryTreeNode.h"

#include <iostream>

#include <string>

using namespace std;

template <class T>

class binaryTreeType

{

private:

int height(BinaryTreeNode<T>\*p)

{

if (p == NULL)

return 0;

else

return 1 + max(height(p->llink), height(p->rlink));

}

int max(int x, int y) const

{

if (x >= y)

return x;

else

return y;

}

void inorder(BinaryTreeNode<T> \*p) const

{

if (p != NULL)

{

inorder(p->llink);

cout << p->info << " ";

inorder(p->rlink);

}

}

void preorder(BinaryTreeNode<T> \*p) const

{

if (p != NULL)

{

cout << p->info << " ";

preorder(p->llink);

preorder(p->rlink);

}

}

void postorder(BinaryTreeNode<T> \*p) const

{

if (p != NULL)

{

postorder(p->llink);

postorder(p->rlink);

cout << p->info << " ";

}

}

int treeNodeCount(BinaryTreeNode<T>\*p)

{

static int count = 0;

if (p != NULL)

{

treeNodeCount(p->llink);

treeNodeCount(p->rlink);

count++;

}

return count;

}

int treeLeafCount(BinaryTreeNode<T>\*p)

{

static int count = 0;

if (p != NULL)

{

treeLeafCount(p->llink);

treeLeafCount(p->rlink);

if (p->llink == NULL && p->rlink == NULL)

count++;

}

return count;

}

protected:

BinaryTreeNode<T> \*root;

public:

bool isEmpty()

{

return (root == NULL)

}

binaryTreeType()

{

root = NULL;

}

~binaryTreeType(){}

int height()

{

return height(root);

}

void preorder()

{

preorder(root);

}

void postorder()

{

postorder(root);

}

void inorder()

{

inorder(root);

}

int treeNodeCount()

{

return treeNodeCount(root);

}

int treeLeafCount()

{

return treeLeafCount(root);

}

};