**Experiment Number: 6b**

**Title:** Implementation of Binary Tree traversal Using C++.

**Problem Statement:** Write a C++ program to implement the following Binary Tree traversal Using C++.

1. In order Traversal.
2. Post order Traversal.
3. Pre order Traversal.

**Algorithm:**

In order Traversal-

Until all nodes are traversed −

* **Step 1** − Recursively traverse left subtree.
* **Step 2** − Visit root node.
* **Step 3** − Recursively traverse right subtree.

Post order Traversal-

Until all nodes are traversed −

* **Step 1** − Recursively traverse left subtree.
* **Step 2** − Recursively traverse right subtree.
* **Step 3** − Visit root node.

Pre order Traversal -

Until all nodes are traversed −

* **Step 1** − Visit root node.
* **Step 2** − Recursively traverse left subtree.
* **Step 3** − Recursively traverse right subtree.

**Code:**

#include<iostream>

using namespace std;

//binary tree node declaration

struct bintree\_node{

bintree\_node \*left;

bintree\_node \*right;

char data;

} ;

class bintree\_class{

bintree\_node \*root;

public:

bintree\_class(){

root=NULL;

}

int isempty() {

return(root==NULL);

}

void insert\_node(int item);

void inorder\_seq();

void inorder(bintree\_node \*);

void postorder\_seq();

void postorder(bintree\_node \*);

void preorder\_seq();

void preorder(bintree\_node \*);

};

void bintree\_class::insert\_node(int item){

bintree\_node \*p=new bintree\_node;

bintree\_node \*parent;

p->data=item;

p->left=NULL;

p->right=NULL;

parent=NULL;

if(isempty())

root=p;

else{

bintree\_node \*ptr;

ptr=root;

while(ptr!=NULL) {

parent=ptr;

if(item>ptr->data)

ptr=ptr->right;

else

ptr=ptr->left;

}

if(item<parent->data)

parent->left=p;

else

parent->right=p;

}

}

void bintree\_class::inorder\_seq()

{

inorder(root);

}

void bintree\_class::inorder(bintree\_node \*ptr)

{

if(ptr!=NULL){

inorder(ptr->left);

cout<<" "<<ptr->data<<" ";

inorder(ptr->right);

}

}

void bintree\_class::postorder\_seq()

{

postorder(root);

}

void bintree\_class::postorder(bintree\_node \*ptr)

{

if(ptr!=NULL){

postorder(ptr->left);

postorder(ptr->right);

cout<<" "<<ptr->data<<" ";

}

}

void bintree\_class::preorder\_seq()

{

preorder(root);

}

void bintree\_class::preorder(bintree\_node \*ptr)

{

if(ptr!=NULL){

cout<<" "<<ptr->data<<" ";

preorder(ptr->left);

preorder(ptr->right);

}

}

int main()

{

bintree\_class bintree;

bintree.insert\_node('L');

bintree.insert\_node('M');

bintree.insert\_node('N);

bintree.insert\_node('O')

bintree.insert\_node('P');

bintree.insert\_node('Q');

bintree.insert\_node('R');

cout<<"Inorder traversal:"<<endl;

bintree.inorder\_seq();

cout<<endl<<"Postorder traversal:"<<endl;

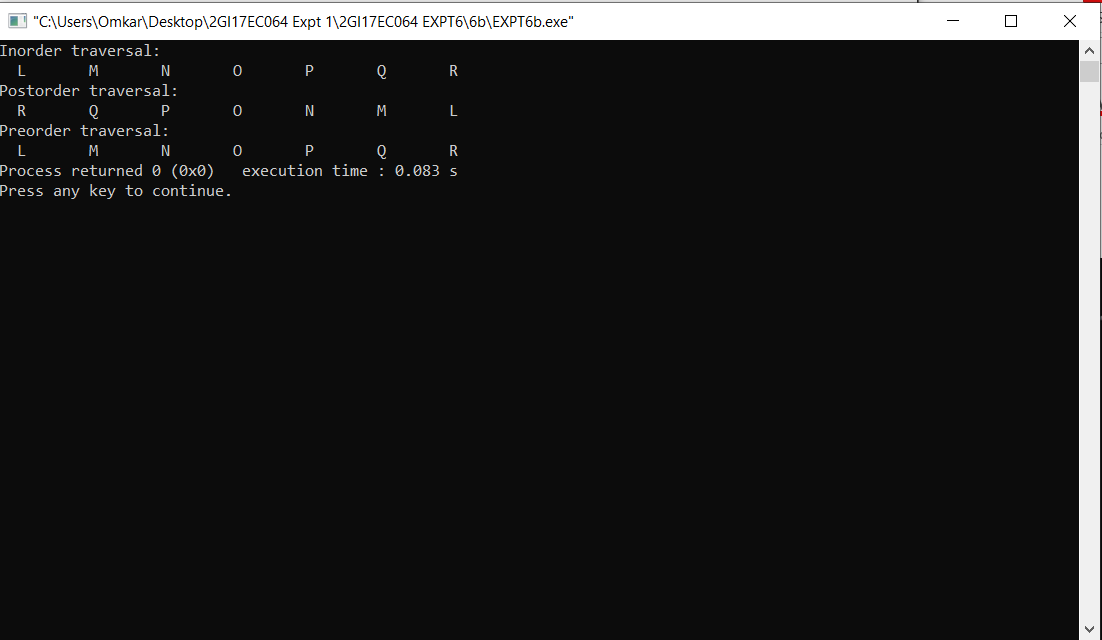
bintree.postorder\_seq();

cout<<endl<<"Preorder traversal:"<<endl;

bintree.preorder\_seq();

}

**Output:**



**6. ANALYSIS(LIMITATIONS):**

The main disadvantage is that we should always implement a balanced binary search tree - AVL tree, Red-Black tree, Splay tree. Otherwise the cost of operations may not be logarithmic and degenerate into a linear search on an array.