Experiment No:6b

Title: Implementation of Binary Tree Traversal in C.

Problem Statement: Implementation of Binary Tree Traversal in C.

Insert()

Inorder()

Preorder()

Postorder()

Algorithm:

1. Start.
2. Create a class with data and left, right pointers to object from class created as private variables.
3. Create isempty function which returns true if there is no root node.
4. In insert function check if tree is empty if yes then create it as root node else check if the data is less than or greater then parent node and insert in tree.
5. In preorder function print parent node then left node then right node.
6. In inorder function print left node then parent node then right node which gives ascending order of tree.
7. In postorder function print left node then right node then parent node
8. Stop.

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('A');

bintree.insert\_node('B');

bintree.insert\_node('C');

bintree.insert\_node('D');

bintree.insert\_node('E');

bintree.insert\_node('F');

bintree.insert\_node('G');

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

bintree.inorder\_seq();

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

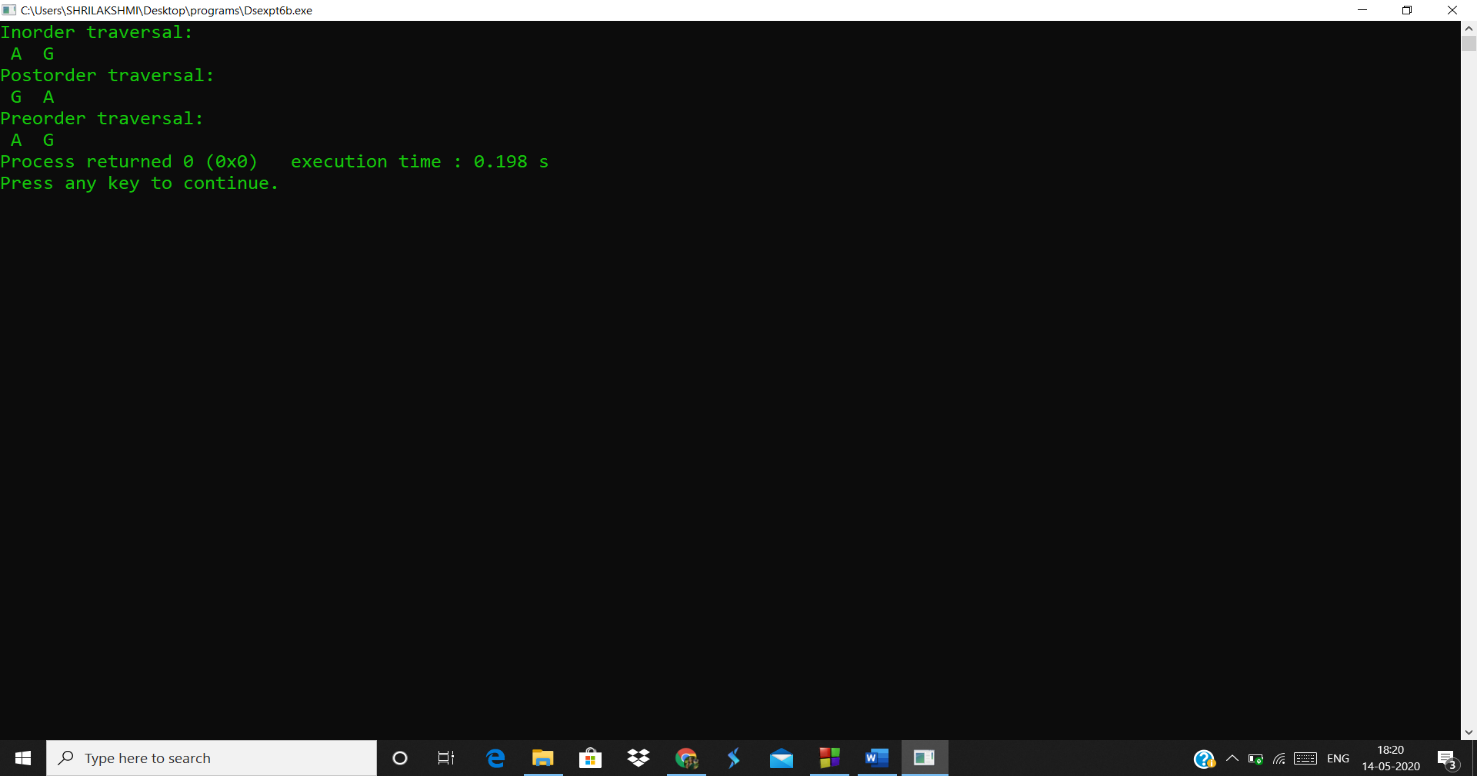
bintree.postorder\_seq();

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

bintree.preorder\_seq();

}

Output:



Analysis(Limitations):

* In inorder, postorder and preorder function if the values in the tree are similar both are analyzed twice while printing which is a repetitive work
* The time complexity increases as the nodes increase since to add a node we have to traverse until the leaf node on any one side.
* Only Inorder function returns ascending order of values but preorder and postorder do not but can be used according to applications as required.