#include<iostream>

using namespace std;

struct tree\_node //name of structure

{

tree\_node \*left; //left child

tree\_node \*right; // right child

int data; //data part

} ;

class bst //name of class

{

private:

tree\_node \*root; //name of root is root

public:

bst() //constructer

{

root=NULL; //root=null

}

/////////////////////////////////////////////////////

//int isempty() //root is empty if root=null //remove commit if u receive error ? why we call no need whole progrm

//{

// return(root==NULL);

//}

//////////////////////////////////////////////////

void insert(int item); //insert function declar

void inordertrav(); //in-order function declar L-R'-R

void inorder(tree\_node \*p); //insert value at in-order function declar L-R'-R ? pointer with p

void postordertrav(); //post-order function declar L-R-R'

void postorder(tree\_node \*); //insert value at post-order function declar L-R-R' ? only pointer

void preordertrav(); //pre-order function declar R'-L-R

void preorder(tree\_node \*); //insert value at pre-order function declar R'-L-R ?

};

////////////////////////////////////////////////////

////DEFINE ALL FUNTION AFTER THE CLASS NOW

void bst::insert(int item) //name of class :: function call to inser node

{

tree\_node \*p=new tree\_node; //create a new node

tree\_node \*parent; //make the node parent

p->data=item; //item into data part

p->left=NULL; //null the left

p->right=NULL; //null the right

parent=NULL; // why?

if(root==NULL)

root=p;

else

{

tree\_node \*ptr; //we can also write as struct tree\_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 bst::inordertrav() //name of class :: function call to inordertrav function

{

inorder(root);

}

void bst::inorder(tree\_node \*ptr)

{

if(ptr!=NULL)

{

inorder(ptr->left);

cout<<" "<<ptr->data<<" "; //these 3 funtion are same in all 3 type of traverse

inorder(ptr->right);

}

}

/////////////////////////////////////////////////////

void bst::postordertrav() //name of class :: function call to post-ordertrav function

{

postorder(root);

}

void bst::postorder(tree\_node \*ptr)

{

if(ptr!=NULL)

{

postorder(ptr->left);

postorder(ptr->right); //these 3 funtion are same in all 3 type of traverse

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

}

}

/////////////////////////////////////////////////////////

void bst::preordertrav() //name of class :: function call to pre-ordertrav function

{

preorder(root);

}

void bst::preorder(tree\_node \*ptr)

{

if(ptr!=NULL)

{

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

preorder(ptr->left); //these 3 funtion are same in all 3 type of traverse

preorder(ptr->right);

}

}

/////////////////////////////////////////////////////main body

void main()

{

bst b;

b.insert(52);

b.insert(25);

b.insert(50);

b.insert(15);

b.insert(40);

b.insert(45);

b.insert(20);

cout<<"inorder"<<endl;

b.inordertrav();

cout<<endl<<"postorder"<<endl;

b.postordertrav();

cout<<endl<<"preorder"<<endl;

b.preordertrav();

getchar();

getchar();

}