**/\* Write a menu driven Program for the following operations on Binary Search Tree (BST) of Integers**

**i. Create a BST of N Integers: 5, 10, 25, 2, 8, 15, 24, 14, 7, 8, 35, 2**

**ii. Traverse the BST in Inorder, Preorder and Post Order**

**iii. Search the BST for a given element (KEY) and print the appropriate message**

**iv. Exit \*/**

**#include<iostream>**

**#include<cstdlib>**

**using namespace std;**

**struct node**

**{**

**int info;**

**node \*left,\*right;**

**};**

**class BST**

**{**

**public:**

**node \*root;**

**BST():root(NULL) {}**

**node \* GetRoot();**

**void AddNewKey(node\* &,int&);**

**void InorderTransvere(node \*);**

**void PreorderTransvere(node \*);**

**void PostorderTransvere(node \*);**

**void SearchKey(node \*,int&);**

**void DeleteKey(node\* &,int&);**

**node \* FindLargestNode(node\* &);**

**};**

**node \* BST::GetRoot()**

**{**

**return root;**

**}**

**void BST::AddNewKey(node\* &r,int &val)**

**{**

**if(r==NULL)**

**{**

**node \*temp=new node;**

**if(temp==NULL)**

**cout<<"\nFailed to initialize memory for new node\n\n";**

**else**

**{**

**temp->info=val;**

**r=temp;**

**r->left=r->right=NULL;**

**}**

**}**

**else**

**{**

**if(val<r->info)**

**AddNewKey(r->left,val);**

**else**

**AddNewKey(r->right,val);**

**}**

**}**

**void BST::PreorderTransvere(node \*r)**

**{**

**if(r!=NULL)**

**{**

**cout<<r->info<<"\t";**

**PreorderTransvere(r->left);**

**PreorderTransvere(r->right);**

**}**

**}**

**void BST::InorderTransvere(node \*r)**

**{**

**if(r!=NULL)**

**{**

**InorderTransvere(r->left);**

**cout<<r->info<<"\t";**

**InorderTransvere(r->right);**

**}**

**}**

**void BST::PostorderTransvere(node \*r)**

**{**

**if(r!=NULL)**

**{**

**PostorderTransvere(r->left);**

**PostorderTransvere(r->right);**

**cout<<r->info<<"\t";**

**}**

**}**

**void BST::SearchKey(node \*r,int &val)**

**{**

**if(r==NULL)**

**cout<<"\n\nThere is no "<<val<<" in BST\n\n";**

**else if(r->info==val)**

**cout<<"\n\n"<<val<<" is present in BST.\n\n";**

**else**

**{**

**if(val<r->info)**

**SearchKey(r->left,val);**

**else**

**SearchKey(r->right,val);**

**}**

**}**

**void BST::DeleteKey(node\* &r,int &val)**

**{**

**if(r==NULL)**

**cout<<"\n\n"<<val<<" is not present in the BST.\n";**

**else if(val<r->info)**

**DeleteKey(r->left,val);**

**else if(val>r->info)**

**DeleteKey(r->right,val);**

**else if(r->left && r->right)**

**{**

**node \*t;**

**t=FindLargestNode(r->left);**

**r->info=t->info;**

**DeleteKey(r->left,t->info);**

**}**

**else**

**{**

**node \*t;**

**t=r;**

**if(r->left==NULL &&r->right==NULL)**

**r=NULL;**

**else if(r->left!=NULL)**

**r=r->left;**

**else**

**r=r->right;**

**delete t;**

**}**

**}**

**node \* BST::FindLargestNode(node\* &l)**

**{**

**if(l->right==NULL)**

**return l;**

**else**

**return FindLargestNode(l->right);**

**}**

**int main()**

**{**

**BST test;**

**int choice,num;**

**while(1)**

**{**

**cout<<"1. Create BST\n2. Add New Key\n3. Preorder Transverse\n4. Inorder Transvere\n5. Postorder Transverse\n6. Search Key\n7. Delete Key\n8. Exit\nEnter your choice : ";**

**cin>>choice;**

**switch(choice)**

**{**

**case 1:**

**{**

**cout<<"\nEnter -1 to end\nEnter the key value: ";**

**cin>>num;**

**while(num!=-1)**

**{**

**test.AddNewKey(test.root,num);**

**cin>>num;**

**}**

**cout<<"\n\n";**

**break;**

**}**

**case 2:**

**{**

**cout<<"\nEnter the key value: ";**

**cin>>num;**

**test.AddNewKey(test.root,num);**

**cout<<"\n\n";**

**break;**

**}**

**case 3:**

**{**

**cout<<"\n\nThe preorder transversal is as follows: \n";**

**test.PreorderTransvere(test.GetRoot());**

**cout<<"\n\n";**

**break;**

**}**

**case 4:**

**{**

**cout<<"\n\nThe inorder transversal is as follows: \n";**

**test.InorderTransvere(test.root);**

**cout<<"\n\n";**

**break;**

**}**

**case 5:**

**{**

**cout<<"\n\nThe postorder transversal is as follows: \n";**

**test.PostorderTransvere(test.GetRoot());**

**cout<<"\n\n";**

**break;**

**}**

**case 6:**

**{**

**cout<<"\n\nEnter the key value which you want to search : ";**

**cin>>num;**

**test.SearchKey(test.GetRoot(),num);**

**break;**

**}**

**case 7:**

**{**

**cout<<"\n\nThe preorder transversal is as follows: \n";**

**test.PreorderTransvere(test.GetRoot());**

**cout<<"\n\n\nEnter the key value which you want to delete : ";**

**cin>>num;**

**test.DeleteKey(test.root,num);**

**cout<<"\n\nThe preorder transversal is as follows: \n";**

**test.PreorderTransvere(test.GetRoot());**

**cout<<"\n\n\n";**

**break;**

**}**

**default :**

**exit(0);**

**}**

**}**

**return 0;**

**}**

**/\* Write a menu driven Program for the following operations on Binary Search Tree (BST) of Integers**

**i. Create a BST of N Integers: 5, 10, 25, 2, 8, 15, 24, 14, 7, 8, 35, 2**

**ii. Traverse the BST in Inorder, Preorder and Post Order**

**iii. Search the BST for a given element (KEY) and print the appropriate message**

**iv. Exit \*/**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**static string path = "";**

**struct node**

**{**

**int data;**

**node \*left;**

**node \*right;**

**};**

**class binTree**

**{**

**public:**

**node \*root;**

**binTree()**

**{**

**root = NULL;**

**}**

**void create\_binTree()**

**{**

**int val;**

**do**

**{**

**cout<<" Enter the value: ";**

**cin>>val;**

**if(val == -1)**

**{**

**break;**

**}**

**insert\_data(val, root);**

**}**

**while (val != -1);**

**}**

**void insert\_data(int val, node \*ptr)**

**{**

**if(root == NULL)**

**{**

**node \*newNode = new node;**

**newNode->data = val;**

**newNode->left = NULL;**

**newNode->right = NULL;**

**root = newNode;**

**}**

**else if(ptr->data <= val)**

**{**

**if(ptr->right == NULL)**

**{**

**node \*newNode = new node;**

**newNode->data = val;**

**newNode->left = NULL;**

**newNode->right = NULL;**

**ptr->right = newNode;**

**}**

**else**

**{**

**insert\_data(val, ptr->right);**

**}**

**}**

**else**

**{**

**if(ptr->left == NULL)**

**{**

**node \*newNode = new node;**

**newNode->data = val;**

**newNode->left = NULL;**

**newNode->right = NULL;**

**ptr->left = newNode;**

**}**

**else**

**{**

**insert\_data(val, ptr->left);**

**}**

**}**

**}**

**void search\_btree(int val, node \*ptr)**

**{**

**if(ptr->data == val)**

**{**

**cout<<"\n The number "<<val<<" is in the tree."<<endl;**

**cout<<" Path : "<<path<<endl;**

**}**

**else if(ptr == NULL)**

**{**

**cout<<" The number doesn't exist."<<endl;**

**}**

**else if(val < ptr->data)**

**{**

**path += "Left ";**

**search\_btree(val, ptr->left);**

**}**

**else**

**{**

**path += "Right ";**

**search\_btree(val, ptr->right);**

**}**

**}**

**void inOrderTrav(node \*ptr)**

**{**

**if(ptr != NULL)**

**{**

**inOrderTrav(ptr->left);**

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

**inOrderTrav(ptr->right);**

**}**

**}**

**void postOrderTrav(node \*ptr)**

**{**

**if(ptr != NULL)**

**{**

**postOrderTrav(ptr->left);**

**postOrderTrav(ptr->right);**

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

**}**

**}**

**void preOrderTrav(node \*ptr)**

**{**

**if(ptr != NULL)**

**{**

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

**preOrderTrav(ptr->left);**

**preOrderTrav(ptr->right);**

**}**

**}**

**void delete\_btree(node\* &r,int &val)**

**{**

**if(r==NULL)**

**cout<<"\n\n"<<val<<" is not present in the BST.\n";**

**else if(val<r->data)**

**delete\_btree(r->left,val);**

**else if(val>r->data)**

**delete\_btree(r->right,val);**

**else if(r->left && r->right)**

**{**

**node \*t;**

**t=LargestNode(r->left);**

**r->data=t->data;**

**delete\_btree(r->left,t->data);**

**}**

**else**

**{**

**node \*t;**

**t=r;**

**if(r->left==NULL &&r->right==NULL)**

**r=NULL;**

**else if(r->left!=NULL)**

**r=r->left;**

**else**

**r=r->right;**

**delete t;**

**}**

**}**

**node \* LargestNode(node\* &l)**

**{**

**if(l->right==NULL)**

**return l;**

**else**

**return LargestNode(l->right);**

**}**

**};**

**int main()**

**{**

**binTree tree1;**

**int choice;**

**do**

**{**

**cout<<"\n\n1. Create Binary Tree.\n";**

**cout<<"2. Insert a number.\n";**

**cout<<"3. Search a number.\n";**

**cout<<"4. In-order display.\n";**

**cout<<"5. Post-order display.\n";**

**cout<<"6. Pre-order display.\n";**

**cout<<"7. Delete a number.\n";**

**cout<<"8. Exit\n";**

**cout<<" Enter your choice: ";**

**cin>>choice;**

**switch(choice)**

**{**

**case 1:**

**{**

**tree1.create\_binTree();**

**break;**

**}**

**case 2:**

**{**

**int val;**

**cout<<endl<<" Enter the number to insert: ";**

**cin>>val;**

**tree1.insert\_data(val,tree1.root);**

**break;**

**}**

**case 3:**

**{**

**int val;**

**cout<<endl<<" Enter the number to search: ";**

**cin>>val;**

**path = "";**

**tree1.search\_btree(val, tree1.root);**

**break;**

**}**

**case 4:**

**{**

**tree1.inOrderTrav(tree1.root);**

**cout<<endl;**

**break;**

**}**

**case 5:**

**{**

**tree1.postOrderTrav(tree1.root);**

**cout<<endl;**

**break;**

**}**

**case 6:**

**{**

**tree1.preOrderTrav(tree1.root);**

**cout<<endl;**

**break;**

**}**

**case 7:**

**{**

**int val;**

**cout<<" Enter the number to delete: ";**

**cin>>val;**

**tree1.delete\_btree(tree1.root, val);**

**}**

**case 8:**

**{**

**break;**

**}**

**default:**

**{**

**cout<<endl<<" !!!Invalid input!!!\n\n";**

**break;**

**}**

**}**

**}**

**while(choice != 8);**

**return 0;**

**}**

**/\* Write a menu driven Program for the following operations on Binary Search Tree (BST) of Integers**

**i. Create a BST of N Integers: 5, 10, 25, 2, 8, 15, 24, 14, 7, 8, 35, 2**

**ii. Traverse the BST in Inorder, Preorder and Post Order**

**iii. Search the BST for a given element (KEY) and print the appropriate message**

**iv. Exit \*/**

**#include<iostream>**

**using namespace std;**

**class BST**

**{**

**struct Node**

**{**

**int data;**

**Node \*left;**

**Node \*right;**

**};**

**typedef struct Node\* nodeptr;**

**public:**

**nodeptr root;**

**BST() //constructor**

**{**

**root=NULL;**

**}**

**void create() // create BST having some data**

**{**

**int val=0;**

**cout<<"insert and end with -1"<<endl;**

**cin>>val;**

**while(val!=-1)**

**{**

**ins(root,val);**

**cin>>val;**

**}**

**}**

**void search\_ele(nodeptr ptr,int data) //search element**

**{**

**if(ptr==NULL || data==ptr->data)**

**{**

**if(ptr==NULL)**

**cout<<endl<<"DATA NOT FOUND"<<endl;**

**else**

**{**

**cout<<"= "<<ptr->data<<endl;**

**cout<<"sucessfully found"<<endl;**

**}**

**}**

**else**

**{**

**if(data<ptr->data)**

**{**

**cout<<"->L";**

**search\_ele(ptr->left,data);**

**}**

**else**

**{**

**cout<<"->R";**

**search\_ele(ptr->right,data);**

**}**

**}**

**}**

**void ins(nodeptr &ptr,int new\_data) //insert at the last**

**{**

**nodeptr p;**

**if(ptr==NULL)**

**{**

**p=new Node;**

**p->data=new\_data;**

**p->left=NULL;**

**p->right=NULL;**

**ptr=p;**

**}**

**else**

**{**

**if(new\_data < ptr->data)**

**ins(ptr->left,new\_data);**

**else**

**ins(ptr->right,new\_data);**

**}**

**}**

**nodeptr find\_largest(nodeptr &ptr) // largest in the tree**

**{**

**if(ptr->right==NULL)**

**return ptr;**

**else**

**return find\_largest(ptr->right);**

**}**

**void del\_data(nodeptr &ptr,int old\_data) // delete the specified data**

**{**

**if(ptr==NULL)**

**{**

**cout<<"Val not found"<<endl;**

**}**

**else if(old\_data<ptr->data)**

**del\_data(ptr->left,old\_data);**

**else if(old\_data>ptr->data)**

**del\_data(ptr->right,old\_data);**

**else if(ptr->left && ptr->right)**

**{**

**nodeptr p=find\_largest(ptr->left);**

**ptr->data=p->data;**

**del\_data(ptr->left,p->data);**

**}**

**else**

**{**

**nodeptr p=ptr;**

**if(ptr->left==NULL && ptr->right==NULL)**

**ptr=NULL;**

**else if(ptr->left!=NULL)**

**ptr=ptr->left;**

**else**

**ptr=ptr->right;**

**delete p;**

**}**

**}**

**void display\_pre(nodeptr ptr) // display Preorder**

**{**

**if(ptr!=NULL)**

**{**

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

**display\_pre(ptr->left);**

**display\_pre(ptr->right);**

**}**

**}**

**void display\_post(nodeptr ptr) // display postorder**

**{**

**if(ptr!=NULL)**

**{**

**display\_post(ptr->left);**

**display\_post(ptr->right);**

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

**}**

**}**

**void display\_in(nodeptr ptr) // display inorder**

**{**

**if(ptr!=NULL)**

**{**

**display\_in(ptr->left);**

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

**display\_in(ptr->right);**

**}**

**}**

**};**

**int main()**

**{**

**BST tree;**

**int x,a;**

**int choice=0;**

**while(choice!=7)**

**{**

**cout<<"\n\nyour Choice please: "<<endl;**

**cout<<"0-create "<<endl;**

**cout<<"1-inserting in Tree "<<endl;**

**cout<<"2-Search"<<endl;**

**cout<<"3-Display preorder"<<endl;**

**cout<<"4-Display postorder"<<endl;**

**cout<<"5-Display Inorder"<<endl;**

**cout<<"6-Delete element"<<endl;**

**cout<<"7-Exit\n"<<endl;**

**cout<<"\t\tyour choice: ";**

**cin>>choice;**

**system("CLS");**

**if(choice!=3)**

**{**

**cout<<"pre-order Display"<<endl;**

**if(tree.root!=NULL)**

**tree.display\_pre(tree.root);**

**else**

**cout<<"empty"<<endl;**

**}**

**cout<<"\n\n"<<endl;**

**switch (choice)**

**{**

**case 0:**

**tree.create();**

**cout<<"\n\npre-order Display"<<endl;**

**if(tree.root!=NULL)**

**tree.display\_pre(tree.root);**

**else**

**cout<<"empty"<<endl;**

**break;**

**case 1:**

**cout<<"enter data to insert: ";**

**cin>>x;**

**tree.ins(tree.root,x);**

**cout<<"\n\npre-order Display"<<endl;**

**if(tree.root!=NULL)**

**tree.display\_pre(tree.root);**

**else**

**cout<<"empty"<<endl;**

**break;**

**case 2:**

**cout<<"enter data to search: ";**

**cin>>x;**

**cout<<"Root";**

**tree.search\_ele(tree.root,x);**

**cout<<"\n\npre-order Display"<<endl;**

**if(tree.root!=NULL)**

**tree.display\_pre(tree.root);**

**else**

**cout<<"empty"<<endl;**

**break;**

**case 3:**

**cout<<"PreOrder Display"<<endl;**

**tree.display\_pre(tree.root);**

**break;**

**case 4:**

**cout<<"PostOrder Display"<<endl;**

**tree.display\_post(tree.root);**

**break;**

**case 5:**

**cout<<"InOrder Display"<<endl;**

**tree.display\_in(tree.root);**

**break;**

**case 6:**

**cout<<"enter data to delete: ";**

**cin>>x;**

**tree.del\_data(tree.root,x);**

**cout<<"\n\npre-order Display"<<endl;**

**if(tree.root!=NULL)**

**tree.display\_pre(tree.root);**

**else**

**cout<<"empty"<<endl;**

**break;**

**case 7:**

**break;**

**}**

**}**

**cout<<"\n============X==========="<<endl;**

**cout<<"\t THANK YOU "<<endl;**

**return 0;**

**}**

**/\* Write a menu driven Program for the following operations on Binary Search Tree (BST) of Integers**

**i. Create a BST of N Integers: 5, 10, 25, 2, 8, 15, 24, 14, 7, 8, 35, 2**

**ii. Traverse the BST in Inorder, Preorder and Post Order**

**iii. Search the BST for a given element (KEY) and print the appropriate message**

**iv. Exit \*/**

**#include<iostream>**

**using namespace std;**

**struct node**

**{**

**int data;**

**struct node \*left, \*right;**

**};**

**class BST**

**{**

**public:**

**node\* root;**

**BST()**

**{**

**root = NULL;**

**}**

**void insert\_node(node\* &tree, int val)**

**{**

**if(tree == NULL)**

**{**

**tree = new node;**

**tree -> data = val;**

**tree -> left = NULL;**

**tree -> right = NULL;**

**}**

**else**

**{**

**if(val < tree -> data)**

**{**

**insert\_node(tree -> left, val);**

**}**

**else**

**{**

**insert\_node(tree -> right, val);**

**}**

**}**

**}**

**void preorder(struct node\* tree)**

**{**

**if (tree != NULL)**

**{**

**cout << tree -> data << endl;**

**preorder(tree -> left);**

**preorder(tree -> right);**

**}**

**}**

**void postorder(struct node\* tree)**

**{**

**if (tree != NULL)**

**{**

**postorder(tree -> left);**

**postorder(tree -> right);**

**cout << tree -> data << endl;**

**}**

**}**

**void inorder(struct node\* tree)**

**{**

**if (tree != NULL)**

**{**

**inorder(tree -> left);**

**cout << tree -> data << endl;**

**inorder(tree -> right);**

**}**

**}**

**void search\_element(struct node\* tree, int val)**

**{**

**if(tree == NULL)**

**cout << "Data not present in tree." << endl;**

**else if(tree -> data == val)**

**cout << "Data found in tree." << endl;**

**else**

**{**

**if(val < tree->data)**

**search\_element(tree -> left, val);**

**else**

**search\_element(tree -> right, val);**

**}**

**}**

**};**

**int main()**

**{**

**int choice = 0, val = 0;**

**BST tree;**

**do**

**{**

**cout << "Main Menu" << endl**

**<< "1. Insert node" << endl**

**<< "2. Search node" << endl**

**<< "3. Preorder Traversal" << endl**

**<< "4. Inorder Traversal" << endl**

**<< "5. Postorder Traversal" << endl**

**<< "6. Exit" << endl**

**<< "Enter your choice: ";**

**cin >> choice;**

**switch(choice)**

**{**

**case 1:**

**cout << "Enter -1 to end." << endl;**

**while(val != -1)**

**{**

**cout << "Enter a number: ";**

**cin >> val;**

**if(val != -1)**

**tree.insert\_node(tree.root, val);**

**}**

**break;**

**case 2:**

**cout << "Enter the number to find: ";**

**cin >> val;**

**tree.search\_element(tree.root, val);**

**break;**

**case 3:**

**tree.preorder(tree.root);**

**break;**

**case 4:**

**tree.inorder(tree.root);**

**break;**

**case 5:**

**tree.postorder(tree.root);**

**break;**

**}**

**cout <<"\n\n\n";**

**}**

**while(choice > 0 && choice < 6);**

**return 0;**

**}**