**// Practical Assignment**

**// 1) Create a menu driven program to implement Binary Search Tree. The menu should have following options.**

**// A) Insert elements into Binary Search Tree**

**// B) Display Binary Search Tree elements.**

**// C) Traversal of Binary Search Tree (with recursion and without recursion)**

**// I) Preorder**

**// II) Postorder**

**// III) Inorder**

**// D) Delete elements from Binary Search Tree**

**// E) Search an element**

**// F) Exit**

**#include<iostream>**

**#include<stdlib.h>**

**#include<stack>**

**using namespace std;**

**struct node**

**{**

**int data;**

**struct node \*left;**

**struct node \*right;**

**}**

**\*list=NULL,\*p,\*s,\*q,\*r,\*d,\*x,\*temp,\*root,\*n;**

**class BinarySearchTree**

**{**

**public:**

**int choice,value,ch,ele,dele,count;**

**int arr[100];**

**BinarySearchTree()**

**{**

**count=0;**

**}**

**void get()**

**{**

**cout<<endl;**

**do**

**{**

**cout<<"0.Exit\n1.Insert elements into Binary Search Tree\n2.Traversal of Binary Search Tree (with recursion )\n3.Traversal of Binary Search Tree (without recursion )\n4.Search and Delete\n5.Display Binary Search Tree\n";**

**cout<<"Enter Your Choice : "<<" ";**

**cin>>choice;**

**switch(choice)**

**{**

**case 0:**

**break;**

**case 1:**

**insert();**

**break;**

**case 2:**

**traversal\_with\_recursion();**

**break;**

**case 3:**

**traversal\_without\_recursion();**

**break;**

**case 4:**

**search\_delete();**

**break;**

**case 5:**

**root=list;**

**display(root,1);**

**break;**

**default:**

**cout<<"invalid input"<<endl<<endl;**

**}**

**}while(choice!=0);**

**cout<<endl;**

**}**

**void insert()**

**{**

**cout<<endl;**

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

**cin>>value;**

**p=(struct node\*)malloc(sizeof(node));**

**p->data=value;**

**root=list;**

**if(list == NULL)**

**{**

**p->left=NULL;**

**p->right=NULL;**

**list=p;**

**//display();**

**}**

**else**

**{**

**while(1)**

**{**

**if( value<root->data )**

**{**

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

**{**

**root->left=p;**

**p->left=NULL;**

**p->right=NULL;**

**break;**

**}**

**root=root->left;**

**}**

**else**

**{**

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

**{**

**root->right=p;**

**p->left=NULL;**

**p->right=NULL;**

**break;**

**}**

**root=root->right;**

**}**

**}**

**//display();**

**}**

**arr[count]=value;**

**count++;**

**cout<<endl;**

**}**

**void traversal\_with\_recursion()**

**{**

**q=list;**

**cout<<endl;**

**do**

**{**

**cout<<endl<<endl;**

**cout<<"0.Exit\n1.Inorder\n2.Preorder\n3.Postorder\n";**

**cout<<"Enter Your Choice : "<<" ";**

**cin>>ch;**

**switch(ch)**

**{**

**case 0:**

**break;**

**case 1:**

**cout<<"Inorder with recursion : ";**

**inorder\_with\_recursion(q);**

**break;**

**case 2:**

**cout<<"Preorder with recursion : ";**

**preorder\_with\_recursion(q);**

**break;**

**case 3:**

**cout<<"Postorder with recursion : ";**

**postorder\_with\_recursion(q);**

**break;**

**default:**

**cout<<"invalid input"<<endl<<endl;**

**}**

**}while(ch!=0);**

**cout<<endl;**

**}**

**void traversal\_without\_recursion()**

**{**

**cout<<endl;**

**do**

**{**

**cout<<endl<<endl;**

**cout<<"0.Exit\n1.Inorder\n2.Preorder\n3.Postorder\n";**

**cout<<"Enter Your Choice : "<<" ";**

**cin>>ch;**

**switch(ch)**

**{**

**case 0:**

**break;**

**case 1:**

**inorder\_without\_recursion();**

**break;**

**case 2:**

**preorder\_without\_recursion();**

**break;**

**case 3:**

**postorder\_without\_recursion();**

**break;**

**default:**

**cout<<"invalid input"<<endl<<endl;**

**}**

**}while(ch!=0);**

**cout<<endl;**

**}**

**void search\_delete()**

**{**

**q=list;**

**cout<<endl;**

**do**

**{**

**cout<<endl<<endl;**

**cout<<"0.Exit\n1.Delete elements from Binary Search Tree\n2.Search an Element\n";**

**cout<<"Enter Your Choice : "<<" ";**

**cin>>ch;**

**switch(ch)**

**{**

**case 0:**

**break;**

**case 1:**

**cout<<"enter the element : ";**

**cin>>dele;**

**delete\_ele(q);**

**break;**

**case 2:**

**cout<<"enter the element : ";**

**cin>>ele;**

**search\_ele(q);**

**break;**

**default:**

**cout<<"invalid input"<<endl<<endl;**

**}**

**}while(ch!=0);**

**cout<<endl;**

**}**

**void inorder\_with\_recursion(struct node \*root)**

**{**

**if(root != NULL)**

**{**

**inorder\_with\_recursion(root->left);**

**cout<<"\t"<<root->data;**

**inorder\_with\_recursion(root->right);**

**}**

**else**

**return;**

**}**

**void preorder\_with\_recursion(struct node \*root)**

**{**

**if(root != NULL)**

**{**

**cout<<"\t"<<root->data;**

**preorder\_with\_recursion(root->left);**

**preorder\_with\_recursion(root->right);**

**}**

**else**

**return;**

**}**

**void postorder\_with\_recursion(struct node \*root)**

**{**

**if(root != NULL)**

**{**

**postorder\_with\_recursion(root->left);**

**postorder\_with\_recursion(root->right);**

**cout<<"\t"<<root->data;**

**}**

**else**

**return;**

**}**

**void inorder\_without\_recursion()**

**{**

**cout<<endl;**

**stack<node\*> stack;**

**node \*curr = list;**

**cout<<"Inorder without recursion : ";**

**while (!stack.empty() || curr != NULL)**

**{**

**if (curr != NULL)**

**{**

**stack.push(curr);**

**curr = curr->left;**

**}**

**else**

**{**

**curr = stack.top();**

**stack.pop();**

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

**curr = curr->right;**

**}**

**}**

**cout<<endl;**

**}**

**void preorder\_without\_recursion()**

**{**

**cout<<endl;**

**root=list;**

**cout<<"Preorder without recursion : ";**

**if (root == NULL)**

**return;**

**stack<node\*> stack;**

**stack.push(root);**

**while (!stack.empty())**

**{**

**node \*curr = stack.top();**

**stack.pop();**

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

**if (curr->right)**

**stack.push(curr->right);**

**if (curr->left)**

**stack.push(curr->left);**

**}**

**cout<<endl;**

**}**

**void postorder\_without\_recursion()**

**{**

**cout<<endl;**

**root=list;**

**cout<<"Postorder without recursion : ";**

**if (root == NULL)**

**return;**

**stack<node\*> stk;**

**stk.push(root);**

**stack<int> out;**

**while (!stk.empty())**

**{**

**node \*curr = stk.top();**

**stk.pop();**

**out.push(curr->data);**

**if (curr->left)**

**stk.push(curr->left);**

**if (curr->right)**

**stk.push(curr->right);**

**}**

**while (!out.empty())**

**{**

**cout << out.top() << " ";**

**out.pop();**

**}**

**cout<<endl;**

**}**

**void search\_ele(struct node \*root)**

**{**

**if(root != NULL)**

**{**

**if(root->data == ele)**

**{**

**cout<<"Element was Found at "<<count<<"th place";**

**cout<<endl;**

**}**

**else if (root->data > ele)**

**{**

**count++;**

**search\_ele(root->left);**

**}**

**else**

**{**

**count++;**

**search\_ele(root->right);**

**}**

**}**

**else**

**return;**

**}**

**void delete\_ele(struct node \*root)**

**{**

**if(root != NULL)**

**{**

**if(root->data == dele)**

**{**

**if(root->data == list->data)**

**{**

**cout<<"Soory Can't Delete the node element "<<endl;**

**}**

**else if (root->left == NULL && root->right == NULL)**

**{**

**cout<<"Element : "<<root->data<<" Deleted ";**

**temp=root->right;**

**d=list;**

**while( d !=root)**

**{**

**if(root->data < d->data)**

**{**

**n=d;**

**d=d->left;**

**}**

**else**

**{**

**n=d;**

**d=d->right;**

**}**

**}**

**if(root->data < list->data)**

**n->left=temp;**

**else**

**n->right=temp;**

**delete root;**

**cout<<endl;**

**}**

**else if (root->left == NULL && root->right != NULL)**

**{**

**temp=root->right;**

**d=list;**

**while( d !=root)**

**{**

**if(root < d)**

**{**

**n=d;**

**d=d->left;**

**}**

**else**

**{**

**n=d;**

**d=d->right;**

**}**

**}**

**if(root->data < list->data)**

**n->left=temp;**

**else**

**n->right=temp;**

**cout<<"Element : "<<root->data<<" Deleted ";**

**delete root;**

**cout<<endl;**

**}**

**else if (root->left != NULL && root->right == NULL)**

**{**

**temp=root->left;**

**d=list;**

**while( d !=root)**

**{**

**if(root->data < d->data)**

**{**

**n=d;**

**d=d->left;**

**}**

**else**

**{**

**n=d;**

**d=d->right;**

**}**

**}**

**if(root->data < list->data)**

**n->left=temp;**

**else**

**n->right=temp;**

**cout<<"Element : "<<root->data<<" Deleted ";**

**delete root;**

**cout<<endl;**

**}**

**else**

**{**

**p=root->left;**

**q=root->right;**

**d=list;**

**while( d !=root)**

**{**

**if(root->data < d->data)**

**{**

**n=d;**

**d=d->left;**

**}**

**else**

**{**

**n=d;**

**d=d->right;**

**}**

**}**

**n->left=q;**

**if(p->data < q->data)**

**n->left->left=p;**

**else**

**n->left->right=p;**

**cout<<"Element : "<<root->data<<" Deleted ";**

**delete root;**

**cout<<endl;**

**}**

**}**

**else if (root->data > dele)**

**{**

**delete\_ele(root->left);**

**}**

**else**

**{**

**delete\_ele(root->right);**

**}**

**}**

**else**

**return;**

**}**

**void display(node \*ptr, int level)**

**{**

**int i;**

**if (ptr != NULL)**

**{**

**display(ptr->right, level+1);**

**cout<<endl;**

**if (ptr == root)**

**cout<<"Root->: ";**

**else**

**{**

**for (i = 0;i < level;i++)**

**cout<<" ";**

**}**

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

**display(ptr->left, level+1);**

**}**

**cout<<endl<<endl;**

**}**

**};**

**int main()**

**{**

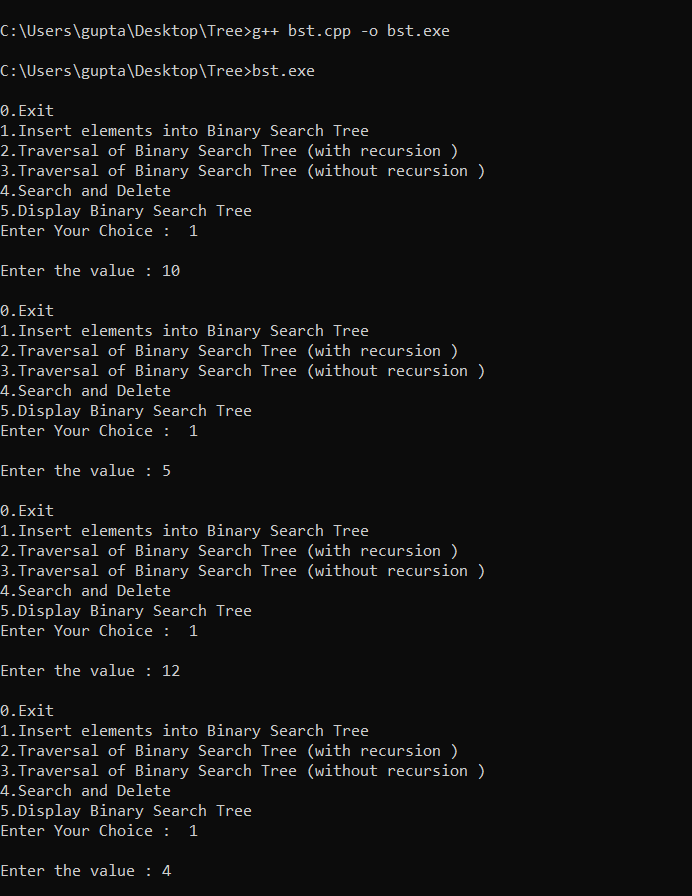
**BinarySearchTree d;**

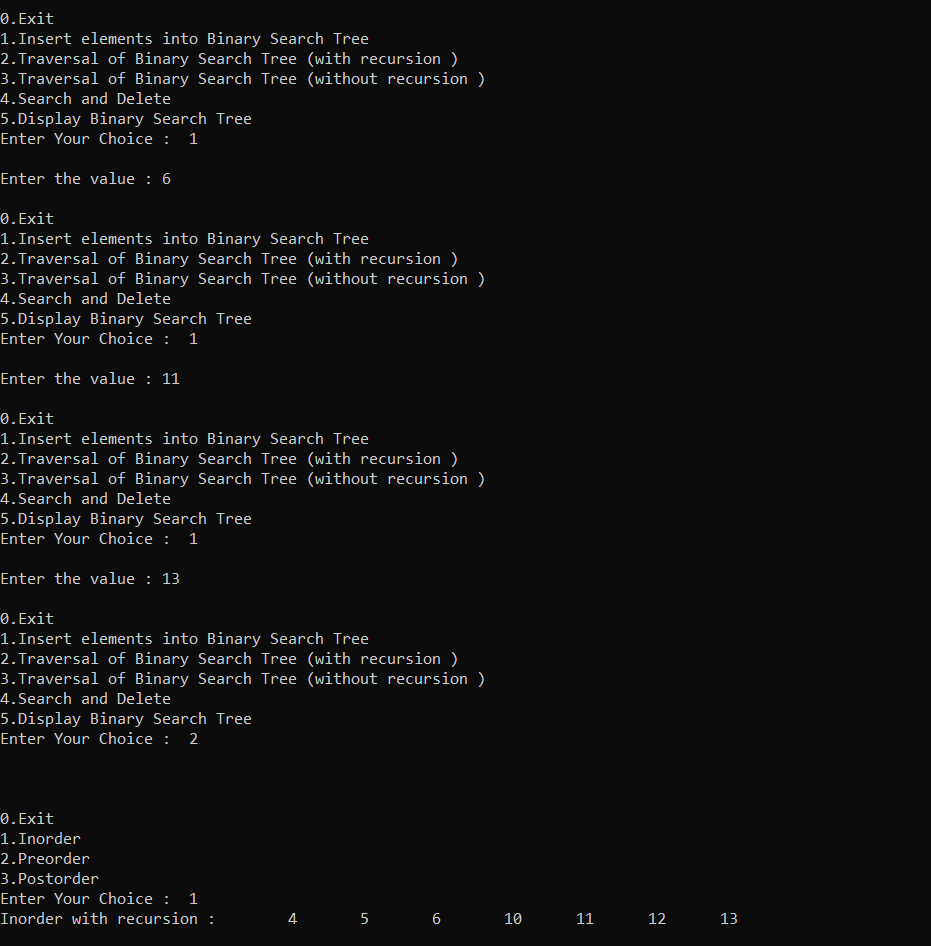
**d.get();**

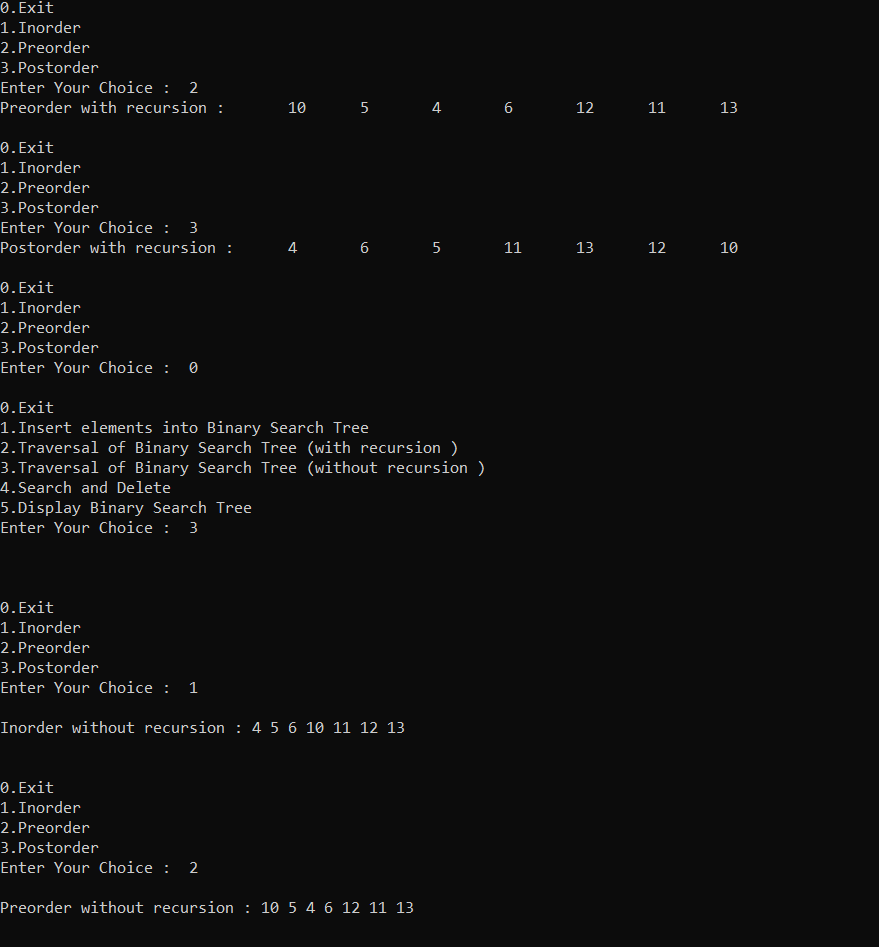
**return 0;**

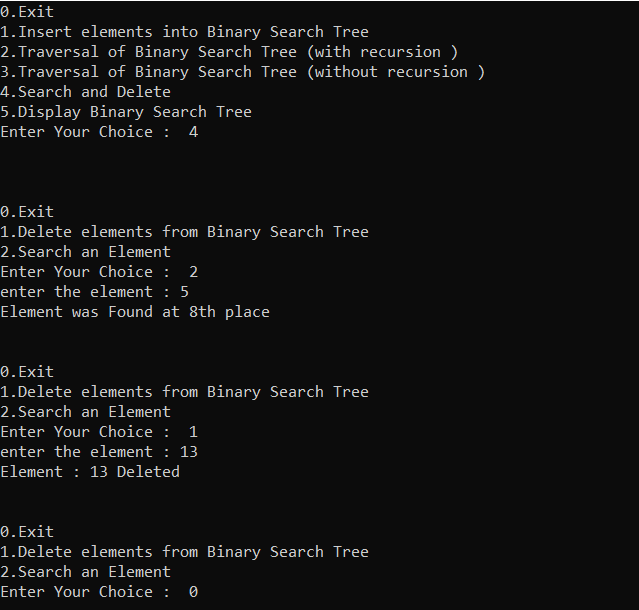
**}**

**Output :**

****

****

****

****

****