#include<iostream>

#include<stdlib.h>

using namespace std;

class BSTnode

{

public:

int data;

BSTnode \*rchild,\*lchild;

};

class tree

{

public:

void create();

void insert(BSTnode \*,BSTnode\*);

void display();

void preorder(BSTnode \*);

void postorder(BSTnode \*);

void inorder(BSTnode \*);

void search();

BSTnode\* searchmin(BSTnode \*);

void searchmax(BSTnode \*);

void searchgiven(BSTnode \*,int,int);

void searchgiven1(BSTnode \*,int,int);

BSTnode \* mirror(BSTnode \*);

BSTnode\* delet(BSTnode \*,int);

BSTnode \*root;

tree()

{

root=NULL;

}

};

/\*Using create function we just create new node of tree\*/

void tree::create()

{

char ch;

BSTnode \*freshnode;

while(1)

{

freshnode=new BSTnode;

freshnode->rchild=NULL;

freshnode->lchild=NULL;

cout<<"\n\n Enter data to create new node\t";

cin>>freshnode->data;

if(root==NULL)

{

root=freshnode;

}

else

insert(root,freshnode);

cout<<"\n You Wnat to add more nodes\t\t";

cin>>ch;

if(ch=='n')

{

break;

}

}

}

/\*using insert function we insert node into tree at given position\*/

void tree::insert(BSTnode \*root,BSTnode \*freshnode)

{

char ch;

if(root->data<freshnode->data)

{

if(root->rchild==NULL)

{

root->rchild=freshnode;

}

else

{

insert(root->rchild,freshnode);

}

}

else

{

if(root->lchild==NULL)

{

root->lchild=freshnode;

}

else

{

insert(root->lchild,freshnode);

}

}

}

void tree::display()

{

int ans;

do

{

cout<<"\n\n\t\tMethods of tree traversal";

cout<<"\n\n\t1.Preorder\n\t2.Postorder\n\t3.Inorder\n\t4.Exit\n";

cout<<"\nEnter your choice\t";

cin>>ans;

switch(ans)

{

case 1:

cout<<"\n\nPreorder travrsal of Tree\n\n";

cout<<"\n-------------------------------------\n";

preorder(root);

cout<<"\n-------------------------------------\n";

break;

case 2:

cout<<"\n\nPostorder travrsal of Tree\n\n";

cout<<"\n-------------------------------------\n";

postorder(root);

cout<<"\n-------------------------------------\n";

break;

case 3:

cout<<"\n\nInorder travrsal of Tree\n\n";

cout<<"\n-------------------------------------\n";

inorder(root);

cout<<"\n-------------------------------------\n";

break;

case 4:

break;

}

} while(ans<4);

}

void tree::preorder(BSTnode \*root1)

{

if(root1!=NULL)

{

cout<<"\t"<<root1->data;

preorder(root1->lchild);

preorder(root1->rchild);

}

}

void tree::postorder(BSTnode \*root1)

{

if(root1!=NULL)

{

postorder(root1->lchild);

postorder(root1->rchild);

cout<<"\t"<<root1->data;

}

}

void tree::inorder(BSTnode \*root1)

{

if(root1!=NULL)

{

inorder(root1->lchild);

cout<<"\t"<<root1->data;

inorder(root1->rchild);

}

}

void tree::search()

{

int ans,ele;

BSTnode \*p;

BSTnode \*m;

do

{

cout<<"\n\n\n\t\tOptions for Search";

cout<<"\n\n\t1.Search Smallest\n\t2.Search Maximum\n\t3.Search the Element\n\t4.Search non Recursion\n\t5.Exit\n";

cout<<"\nEnter your choice\t";

cin>>ans;

switch(ans)

{

case 1:

p=searchmin(root);

break;

case 2:

searchmax(root);

break;

case 3:

cout<<"\nEnter the Element to Search";

cin>>ele;

searchgiven(root,ele,1);

break;

case 4:

cout<<"\nEnter the Element to Search";

cin>>ele;

searchgiven1(root,ele,1);

break;

}

} while(ans<5);

}

BSTnode\* tree::searchmin(BSTnode \*root1)

{

while(root1->lchild!=NULL)

{

root1=root1->lchild;

}

cout<<"\n\n Smallest Element is "<<root1->data;

return(root1);

}

void tree::searchmax(BSTnode \*root1)

{

while(root1->rchild!=NULL)

{

root1=root1->rchild;

}

cout<<"\n\n Maximum Element is "<<root1->data;

}

void tree::searchgiven(BSTnode \*root1,int elm,int count)

{

if(root1==NULL)

cout<<"\nElement is not present in tree\t";

else

{

if(root1->data==elm)

cout<<"\nElement found at level\t"<<count;

else

{

if(root1->data<elm)

{

count++;

searchgiven(root1->rchild,elm,count);

}

if(root1->data>elm)

{

count++;

searchgiven(root1->lchild,elm,count);

}

}

}

}

void tree::searchgiven1(BSTnode \*root1,int key,int count)

{

int flag=0;

while(root1!=NULL)

{

if (key<root1->data)

{

root1=root1->lchild;

//cout<<"\n\t"<<root1->data;

count++;

}

if(key>root1->data)

{

root1=root1->rchild;

//cout<<"\n\t"<<root1->data;

count++;

}

if(root1->data==key)

{

cout<<"Element found at level \t"<<count;

flag=1;

break;

}

}

if(root1==NULL)

cout<<"\nElement is not present in tree\t";

}

BSTnode \* tree ::delet(BSTnode \*root1,int key)

{

BSTnode \*temp;

if(root1==NULL)

{

cout<<"\nElement not found :";

return(root1);

}

if(key< root1->data) // delete in left subtree

{

root1->lchild=delet(root1->lchild,key);

return(root1);

}

if(key > root1->data) // delete in right subtree

{

root1->rchild=delet(root1->rchild,key);

return(root1);

}

if(root1->data==key)

{

// element is found

if(root1->lchild==NULL && root1->rchild==NULL) // a leaf node

{

temp=root1;

root1=NULL;

free(temp);

return(root1);

}

if(root1->lchild==NULL&&root1->rchild!=NULL) //Having Right child

{

temp=root1;

root1=root1->rchild;

free(temp);

return(root1);

}

if(root1->rchild==NULL&&root1->lchild!=NULL) //Having Leftchild

{

temp=root1;

root1=root1->lchild;

free(temp);

return(root1);

}

if(root1->lchild!=NULL && root1->rchild!=NULL) //Both Child

{

int temp1;

// node with two children

temp=searchmin(root1->rchild);

root1->data=temp->data;

root1->rchild=delet(root1->rchild,temp->data);

return(root1);

}

}

}

BSTnode \* tree::mirror(BSTnode\*root1)

{

BSTnode \*temp;

temp=NULL;

if(root1!=NULL)

{

temp=new(BSTnode);

temp->data=root1->data;

temp->lchild=mirror(root1->rchild);

temp->rchild=mirror(root1->lchild);

}

return(temp);

}

int main()

{

tree t;

int ch;

do

{

cout<<"\n\n-----------------TREE------------------------ \n\n";

cout<<"\n\t1.Create\n\t2.Disply\n\t3.Search\n\t4.Insert the Node\n\t5.Delete\n\t6.Mirror\n\t7.Exit\n";

cout<<"\nEnter your choice\t";

cin>>ch;

switch(ch)

{

case 1:

t.create();

break;

case 2:

t.display();

break;

case 3:

t.search();

break;

case 4:

BSTnode \*freshnode;

freshnode=new BSTnode;

freshnode->rchild=NULL;

freshnode->lchild=NULL;

cout<<"\n\n Enter data to create new node\t";

cin>>freshnode->data;

t.insert(t.root,freshnode);

break;

case 5:

int key;

BSTnode \*temp;

cout<<"\nEnter the data to Delete the node";

cin>>key;

temp=t.delet(t.root,key);

cout<<"\n\n\tDelete the node";

break;

case 6:

BSTnode \*mroot;

mroot=t.mirror(t.root);

cout<<"\n\n\t Original tree\n";

t.preorder(t.root);

cout<<"\n\t Mirror Image of tree \n";

t.preorder(mroot);

break;

}

} while(ch<7);

return(0);

}