EXERCISE 1-5

#include<bits/stdc++.h>

using namespace std;

#include <stdlib.h>

#include <conio.h>

struct node{

int data;

struct node \*left;

struct node \*right;

}\*root=NULL,\*parent;

void insert (void)

{

struct node \*NewRoot;

int x,f=0;

cout<<"Enter the value to be insert: ";

cin>>x;

if(root==NULL)

{

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

root->data =x;

root->left =NULL;

root->right =NULL;

}

else

{

NewRoot= root;

while(NewRoot != NULL && f==0)

{

if(NewRoot->data>x)

{

parent = NewRoot;

NewRoot = NewRoot->left;

}

else if(NewRoot->data<x)

{

parent = NewRoot;

NewRoot = NewRoot->right;

}

else{cout<<"\nAlready Inserted";f=1;}

}

if(f!=1)

{

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

NewRoot->data = x;

NewRoot->right = NULL;

NewRoot->left = NULL;

if(parent->data<x)

parent->right = NewRoot;

else

parent->left = NewRoot;

}

}

cout<<"\nInsertion in the tree is Successful!!!\n"<<endl;

cout<<"\n\t\tEnter any key to go to menu"<<endl;

getch();

system("cls");

}

void preorder(struct node \*ptr)

{

if(ptr !=NULL)

{

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

preorder(ptr->left);

preorder(ptr->right);

}

}

void inorder(struct node \*ptr)

{

if(ptr !=NULL)

{

inorder(ptr->left);

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

inorder(ptr->right);

}

}

void postorder(struct node \*ptr)

{

if(ptr !=NULL)

{

postorder(ptr->left);

postorder(ptr->right);

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

}

}

struct node\* findMin(struct node\* root)

{

if(root==NULL)

return NULL;

if(root->left!=NULL)

return findMin(root->left);

else

return root;

}

struct node\* Delete(struct node\* root,int x)

{

if(root==NULL){return root; cout<<"The tree is empty."<<endl;}

else if(x<root->data) root->left=Delete(root->left,x);

else if(x>root->data) root->right=Delete(root->right,x);

else

{

//case 1

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

{

delete root;

root=NULL;

return root;

}

// case 2

else if(root->left==NULL)

{

struct node\* temp= root;

root=root->right;

free (temp);

return root;

}

else if(root->right==NULL)

{

struct node\* temp= root;

root=root->right;

free (temp);

return root;

}

//case 3

else

{

struct node\* temp= findMin(root->right);

root->data=temp->data;

root->right=Delete(root->right,temp->data);

}

}

return root;

}

void Min(struct node \*ptr)

{

if(ptr==NULL)

cout<<"\nThe tree is empty"<<endl;

else if(ptr->left==NULL)

cout<<"\nMinimum: "<<ptr->data<<endl;

else

return Min(ptr->left);

cout<<"\n\t\tEnter any key to go to menu"<<endl;

getch();

system("cls");

}

void Max(struct node \*ptr)

{

if(ptr==NULL)

cout<<"\nThe tree is empty"<<endl;

else if(ptr->right==NULL)

cout<<"\nMaximum: "<<ptr->data<<endl;

else

return Max(ptr->right);

cout<<"\n\t\tEnter any key to go to menu"<<endl;

getch();

system("cls");

}

int main()

{

while(1)

{

int n;

cout<<"## Binary Search Tree ##"<<endl;

cout<<"\_\_\_\_\_\_\_Menu\_\_\_\_\_\_\_"<<endl;

cout<<"\n1. Insert\n2. Preorder\n3. Inorder\n4. Postorder\n5. Delete Node\n6. Find the max number\n7. Find the min number\n8. Exit"<<endl;

cout<<"\nEnter Choice: ";

cin>>n;

system("cls");

switch(n)

{

case 1:

insert();

break;

case 2:

cout<<"Here is the preorder of the tree!!!\n"<<endl;

preorder(root);

cout<<endl;

cout<<"\n\t\tEnter any key to go to menu"<<endl;

getch();

system("cls");

break;

case 3:

cout<<"Here is the inorder of the tree!!!\n"<<endl;

inorder(root);

cout<<endl;

cout<<"\n\t\tEnter any key to go to menu"<<endl;

getch();

system("cls");

break;

case 4:

cout<<"Here is the postorder of the tree!!!\n"<<endl;

postorder(root);

cout<<endl;

cout<<"\n\t\tEnter any key to go to menu"<<endl;

getch();

system("cls");

break;

case 5:

int a;

cout<<"Enter the number you want to delete: ";

cin>>a;

Delete(root,a);

cout<<endl;

cout<<"\nDeletation in the tree is Successful!!!\n"<<endl;

cout<<"\n\t\tEnter any key to go to menu"<<endl;

getch();

system("cls");

break;

case 6:

Max(root);

cout<<endl;

break;

case 7:

Min(root);

cout<<endl;

break;

case 8:

exit(0);

default: cout<<"\nInvalid Option";

}

}

}