// binary\_tree3.cpp : Defines the entry point for the console application.

//

#include"stdafx.h"

#include<stdio.h>

#include<stdlib.h>

structnode

{

intkey;

node\*left,\*right;

};

// C function to search a given key in a given BST

structnode\*search(structnode\*root,intkey)

{

// Base Cases: root is null or key is present at root

if(root==NULL||root->key==key)

returnroot;

// Key is greater than root's key

if(root->key<key)

returnsearch(root->right,key);

// Key is smaller than root's key

returnsearch(root->left,key);

}

// A utility function to create a new BST node

structnode\*newNode(intitem)

{

structnode\*temp=newnode;;

temp->key=item;

temp->left=temp->right=NULL;

returntemp;

}

// A utility function to do inorder traversal of BST

voidinorder(structnode\*root)

{

if(root!=NULL)

{

inorder(root->left);

printf("%d \n",root->key);

inorder(root->right);

}

}

/\* A utility function to insert a new node with given key in BST \*/

structnode\*insert(structnode\*node,intkey)

{

/\* If the tree is empty, return a new node \*/

if(node==NULL)returnnewNode(key);

/\* Otherwise, recur down the tree \*/

if(key<node->key)

node->left=insert(node->left,key);

elseif(key>node->key)

node->right=insert(node->right,key);

/\* return the (unchanged) node pointer \*/

returnnode;

}

/\* Given a non-empty binary search tree, return the node with minimum

key value found in that tree. Note that the entire tree does not

need to be searched. \*/

structnode\*minValueNode(structnode\*node)

{

structnode\*current=node;

/\* loop down to find the leftmost leaf \*/

while(current->left!=NULL)

current=current->left;

returncurrent;

}

/\* Given a binary search tree and a key, this function deletes the key

and returns the new root \*/

structnode\*deleteNode(structnode\*root,intkey)

{

// base case

if(root==NULL)returnroot;

// If the key to be deleted is smaller than the root's key,

// then it lies in left subtree

if(key<root->key)

root->left=deleteNode(root->left,key);

// If the key to be deleted is greater than the root's key,

// then it lies in right subtree

elseif(key>root->key)

root->right=deleteNode(root->right,key);

// if key is same as root's key, then This is the node

// to be deleted

else

{

// node with only one child or no child

if(root->left==NULL)

{

structnode\*temp=root->right;

delete(root);

returntemp;

}

elseif(root->right==NULL)

{

structnode\*temp=root->left;

delete(root);

returntemp;

}

// node with two children: Get the inorder successor (smallest

// in the right subtree)

structnode\*temp=minValueNode(root->right);

// Copy the inorder successor's content to this node

root->key=temp->key;

// Delete the inorder successor

root->right=deleteNode(root->right,temp->key);

}

returnroot;

}

// Driver Program to test above functions

intmain()

{

/\* Let us create following BST

 50

 / \

 30 70

 / \ / \

 20 40 60 80 \*/

structnode\*root=NULL;

root=insert(root,50);

insert(root,30);

insert(root,20);

insert(root,40);

insert(root,70);

insert(root,60);

insert(root,80);

// print inoder traversal of the BST

inorder(root);

printf("Inorder traversal of the given tree \n");

inorder(root);

printf("\nDelete 20\n");

root=deleteNode(root,20);

printf("Inorder traversal of the modified tree \n");

inorder(root);

printf("\nDelete 30\n");

root=deleteNode(root,30);

printf("Inorder traversal of the modified tree \n");

inorder(root);

printf("\nDelete 50\n");

root=deleteNode(root,50);

printf("Inorder traversal of the modified tree \n");

inorder(root);

return0;

}