Q1. Implement insertion, deletion, searching, and traversal in a Binary Search Tree.

https://ideone.com/I6ZCzL

Input file: bit.ly/3uB3oXw

Ans.

/\*

1. Create a binary search tree from a randomly generated set of integers (Insertion)

2. Delete nodes from a binary search tree - a) with 0 child, ii) with 1 child and ii) with 2 children

3. Implement an inorder traversal on the binary search tree in 1. to ensure ascending order sort

4. Search for a key in the binary search tree in 1. You should enumerate the nodes in the search path

\*/

// C program to demonstrate

// delete operation in binary

// search tree

#include <stdio.h>

#include <stdlib.h>

struct node {

    int key;

    struct node \*left, \*right;

};

// A utility function to create a new BST node

struct node\* newNode(int item)

{

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

    temp->key = item;

    temp->left = temp->right = NULL;

    return temp;

}

// A utility function to do inorder traversal of BST

void inorder(struct node\* root)

{

    if(root != NULL){

        inorder(root->left);

        printf("%d ", root->key);

        inorder(root->right);

    }

}

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

struct node\* insert(struct node\* node, int key)

{

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

    struct node\* new\_node = newNode(key);

    if(node == NULL){

        return new\_node;

    }

    struct node\* store = node;

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

    int inserted = 0;

    while(1){

        if(node->key>=new\_node->key){

            if(node->left == NULL){

                node->left = new\_node;

                inserted = 1;

            }

            else{

                node = node->left;

            }

        }

        else{

            if(node->right == NULL){

                node->right = new\_node;

                inserted = 1;

            }

            else{

                node = node->right;

            }

        }

        if(inserted){

            break;

        }

    }

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

    return store;

}

/\* 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. \*/

struct node\* minValueNode(struct node\* node)

{

    //Create a temporary node

    struct node\* temp = node;

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

    while(temp->left!=NULL){

        temp = temp->left;

    }

    // Return the temporary node which is the desired minValue node

    return temp;

}

/\*

Function to search for a given node

\*/

struct node\* search(struct node\* root, int key)

{

    // Display the node if it is not NULL otherwise return the NULL node

    if(root == NULL){

        return NULL;

    }

    printf("\nNode: %d", root->key);

    if(root->key == key){

        return root;

    }

    else if(key>=root->key){

        return search(root->right, key);

    }

    else{

        return search(root->left, key);

    }

    // If the key to be searched is smaller than the root's key, then it lies in left subtree

    // Call the search function on the left child and return it

    // If the key to be searched is greater than the root's key, then it lies in right subtree

    // Call the search function on the right child and return it

    // else if key is same as root's key, then this is the node, return it

}

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

struct node\* deleteNode(struct node\* root, int key)

{

    // if root is NULL then display Not found and return the node

    if(root == NULL){

        printf("\nNot found");

        return root;

    }

    // If the key to be deleted is smaller than the root's key, then it lies in left subtree

    // Call the deleteNode function on the left child of root and assign it to root->left

    if(key<root->key){

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

        return root;

    }

    else if(key>root->key){

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

        return root;

    }

    else{

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

            // Has no child

            free(root);

            return NULL;

        }

        else if(root->left == NULL){

            //Has only right child

            struct node\* temp = root->right;

            root->right = NULL;

            free(root);

            return temp;

        }

        else if(root->right == NULL){

            //Has only left child

            struct node\* temp = root->left;

            root->left = NULL;

            free(root);

            return temp;

        }

        else{

            //Has 2 children

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

            root->key = temp->key;

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

            return root;

        }

    }

    // If the key to be deleted is greater than the root's key, then it lies in right subtree

    // Call the deleteNode function on the right child of root and assign it to root->right

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

    // else {

        // Write code for node with only one child or no child, store the deleted node in a temporary node, free the deleted node and then return the temporary node

        // node with two children:

        // Get the inorder successor (Call minValueNode)

        // (smallest in the right subtree)

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

        // Delete the inorder successor

    // return root

}

// Driver Code

int main()

{

    printf("Creating Binary Search Tree\n");

    /\* Let us create following BST

              50

           /     \

          30      70

         /  \    /  \

       20   40  60   80 \*/

    struct node\* root = NULL;

    root = insert(root, 50);

    root = insert(root, 30);

    root = insert(root, 20);

    root = insert(root, 40);

    root = insert(root, 70);

    root = insert(root, 60);

    root = insert(root, 80);

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

    inorder(root);

    printf("\nSearching for key: 40");

    struct node\* data = search(root, 40);

    if(data->key==40)

        printf("\nFound 40 in Binary Search tree");

    else

        printf("\n40 not found");

    printf("\nSearching for key: 1234");

    data = search(root, 1234);

    if(data==NULL)

        printf("\n1234 not found");

    else

        printf("\nFound 1234 in Binary Search tree");

    printf("\nDeleting 20 (node with 0 child)\n");

    root = deleteNode(root, 20);

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

    inorder(root);

    printf("\nDeleting 30 (node with 1 child)\n");

    root = deleteNode(root, 30);

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

    inorder(root);

    printf("\nDelete 70 (node with 2 children)\n");

    root = deleteNode(root, 70);

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

    inorder(root);

    return 0;

}

Output

