**Collage : Vishwakarma Institute of Technology**

**Course Name : Data Structure in C**

**Name : Vedika Vikas Sontakke**

**Roll no : 37**

**PRN NO 12220206**

Assignment 4 : create BST and perform following operations :

1. INSERT
2. DELETE
3. LevelWise DISPLAY
4. MIRROR image
5. HEIGHT of the tree

Program :

#include<stdio.h>

#include<stdlib.h>

struct node{

    int data;

    struct node \* left;

    struct node \* right;

};

struct node \* root ;

struct node \*queue[1000];

int front = -1 , back = -1;

void insert(int num)

{

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

   newnode->data = num;

   newnode->left = newnode->right = NULL;

   if(root == NULL){

      root = newnode;

      return;

   }

   struct node \* curr = root;

   struct node \* prev = root;

   while(curr != NULL)

   {

      if(num < curr->data)

      {

         prev = curr;

         curr = curr->left;

      }else if(num > curr->data)

      {

         prev = curr;

         curr = curr->right;

      }

   }

   if(num < prev->data) prev->left = newnode;

   else prev->right = newnode;

}

void inorder(struct node \*root)

{

   if(root == NULL) return;

   inorder(root->left);

   printf("%d |",root->data);

   inorder(root->right);

}

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

{

    struct node\* curr = root;

    struct node\* prev = NULL;

    while (curr != NULL && curr->data != key) {

        prev = curr;

        if (key < curr->data) curr = curr->left;

        else curr = curr->right;

    }

    if (curr == NULL) {

       printf("%d key not found ", key);

        return root;

    }

    // Check if the node to be deleted has atmost one child.

    if (curr->left == NULL || curr->right == NULL) {

        struct node\* newCurr;

        if (curr->left == NULL) newCurr = curr->right;

        else newCurr = curr->left;

        if (prev == NULL) return newCurr;

        if (curr == prev->left)  prev->left = newCurr;

        else prev->right = newCurr;

        free(curr);

    }

    // node to be deleted has two children.

    else {

        struct node\* p = NULL;

        struct node\* temp;

        // Compute the inorder successor

        temp = curr->right;

        while (temp->left != NULL) {

            p = temp;

            temp = temp->left;

        }

        if (p != NULL) p->left = temp->right;

        else curr->right = temp->right;

        curr->data = temp->data;

        free(temp);

    }

    return root;

}

void enqueue(struct node\* data)

{

   if(front == -1)

   {

     queue[++back] = data;

     front++;

   }else queue[++back] = data;

}

struct node\* dequeue()

{

    return queue[front++];

}

void displayLevelWise(struct node\* root)

{

   enqueue(root);

   struct node\* tp1 = root;

   while(1)

   {

      if(front > back) break;

      tp1 = dequeue();

      printf("%d | ",tp1->data);

      if(tp1->left != NULL) enqueue(tp1->left);

      if(tp1->right != NULL) enqueue(tp1->right);

   }

   printf("\n");

}

void mirror(struct node\* root)

{

    if (root == NULL)

        return;

    else {

        struct node\* temp;

        /\* do the subtrees \*/

        mirror(root->left);

        mirror(root->right);

        /\* swap the pointers in this node \*/

        temp = root->left;

        root->left = root->right;

        root->right = temp;

    }

}

int tree\_height(struct node\* root) {

    // Get the height of the tree

    if (!root)

        return 0;

    else {

        // Find the height of both subtrees

        // and use the larger one

        int left\_height = tree\_height(root->left);

        int right\_height = tree\_height(root->right);

        if (left\_height >= right\_height) return left\_height + 1;

        else return right\_height + 1;

    }

}

int main()

{

   root = NULL;

   int size;

   printf("enter the number of element you want to insert :\n");

   scanf("%d",&size);

   int element;

   for(int i=0; i<size ; i++)

   {

      printf("enter the %d element : ", i+1);

      scanf("%d", &element);

      insert(element);

   }

   printf("inorder traversal :");

   inorder(root);

   printf("\nenter a elment which you want to insert :");

   scanf("%d" , &element);

   insert(element);

   printf("\ninorder traversal after inseration :");

   inorder(root);

   int del\_element;

   printf("\nenter the node which you want to delete :");

   scanf("%d",&del\_element);

   deleteNode(root , del\_element);

   printf("\ninorder traversal after deletion : ");

   inorder(root);

   printf("\ndisplay level wise : ");

   displayLevelWise(root);

   mirror(root);

   printf("\nInorder traversal of the mirror tree is : ");

   inorder(root);

    int height = tree\_height(root);

    printf("\nHeight of the Binary Tree: %d", height);

}

/\*

/\* Change a tree so that the roles of the  left and

    right pointers are swapped at every node.

 So the tree...

       4

      / \

     2   5

    / \

   1   3

 is changed to...

       4

      / \

     5   2

        / \

       3   1

\*/

Output :

