**// Tree traversal using recursive function (preorder)**

#include <stdio.h>

#include <stdlib.h>

struct node{

int dt;

struct node \*l, \*r;

};

struct node\* newnode(int val)

{

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

nn->dt=val;

nn->l=NULL;

nn->r=NULL;

}

void disp(struct node \*tp)

{

if(tp!=NULL){

printf("%d ", tp->dt);

disp(tp->l);

disp(tp->r);

}}

int main() {

struct node \*root = newnode(1);

root->l=newnode(2);

root->r=newnode(3);

root->l->l=newnode(4);

root->l->r=newnode(5);

disp(root);

return 0;

}

**// Tree traversal using recursive function (Inorder)**

#include <stdio.h>

#include <stdlib.h>

struct node{

int dt;

struct node \*l, \*r;

};

struct node\* newnode(int val)

{

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

nn->dt=val;

nn->l=NULL;

nn->r=NULL;

}

void disp(struct node \*tp)

{

if(tp!=NULL){

disp(tp->l);

printf("%d ", tp->dt);

disp(tp->r);

}}

int main() {

struct node \*root = newnode(1);

root->l=newnode(2);

root->r=newnode(3);

root->l->l=newnode(4);

root->l->r=newnode(5);

disp(root);

return 0;

}

**//Tree traversal using recursive function (Postorder)**

#include <stdio.h>

#include <stdlib.h>

struct node{

int dt;

struct node \*l, \*r;

};

struct node\* newnode(int val)

{

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

nn->dt=val;

nn->l=NULL;

nn->r=NULL;

}

void disp(struct node \*tp)

{

if(tp!=NULL){

disp(tp->l);

disp(tp->r);

printf("%d ", tp->dt);

}}

int main() {

struct node \*root = newnode(1);

root->l=newnode(2);

root->r=newnode(3);

root->l->l=newnode(4);

root->l->r=newnode(5);

disp(root);

return 0;

}

**//Binary Search Tree**

#include <stdio.h>

#include <stdlib.h>

int s;

struct node{

int dt;

struct node \*l, \*r;

}\*root=NULL,\*tp=NULL,\*nn=NULL,\*dn=NULL;

void insert(){

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

printf("Enter val:");

scanf("%d",&nn->dt);

nn->l=NULL;

nn->r=NULL;

if(root==NULL)

root=nn;

else{

tp=root;

while((nn->dt<tp->dt && tp->l!=NULL) || (nn->dt>tp->dt && tp->r!=NULL)){

if(nn->dt<tp->dt && tp->l!=NULL)

tp=tp->l;

if(nn->dt>tp->dt && tp->r!=NULL)

tp=tp->r;

}

if(nn->dt<tp->dt)

tp->l=nn;

else

tp->r=nn;

}}

struct node\* search(struct node \*tp){

printf("Enter element: ");

scanf("%d",&s);

while(tp!=NULL){

if(tp->dt==s){

printf("Element available ");

return tp;

}

else if(s<tp->dt)

tp=tp->l;

else

tp=tp->r;

}

printf("Element not available");

return NULL;

}

void del(){

dn=search(root);

if(dn!=NULL){

if(dn->l!=NULL && dn->r!=NULL){ //Node with 2 children

tp=dn->l;

nn=tp;

while(nn->r!=NULL){ //To find largest value in left sub tree of delete node dn

tp=nn;

nn=tp->r;

}

if(tp!=nn)

tp->r=nn->l;

else

dn->l=nn->l;

dn->dt=nn->dt;

free(nn);

}

else{

if(dn==root) //To delete root node with 0 or 1 child

root = (root->l!=NULL)? root->l: root->r;

else{

tp=root;

while(tp->l!=dn && tp->r!=dn)

tp=(s<tp->dt)? tp->l : tp->r;

if (tp->l==dn)

tp->l=(dn->r!=NULL)? dn->r:dn->l;

else

tp->r=(dn->r!=NULL)? dn->r:dn->l;

}

free(dn);

}}

printf(" To Delete");}

void disp(struct node \*tp){

if(tp!=NULL){

disp(tp->l);

printf("%d ", tp->dt);

disp(tp->r);

}}

int main() {

int ch;

do{

printf("\nEnter 0 to exit, 1 to insert, 2 to disp, 3 to search, 4 to del : ");

scanf("%d",&ch);

switch(ch){

case 0: break;

case 1: insert();

break;

case 2: disp(root);

break;

case 3: search(root);

break;

case 4: del();

break;

}}while(ch!=0);

return 0;

}

**//Binary Search Tree- Display all the elements greater than the deleted element**

#include <stdio.h>

#include <stdlib.h>

int s;

struct node{

int dt;

struct node \*l, \*r;

}\*root=NULL,\*tp=NULL,\*nn=NULL,\*dn=NULL;

void insert(){

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

printf("Enter val:");

scanf("%d",&nn->dt);

nn->l=NULL;

nn->r=NULL;

if(root==NULL)

root=nn;

else{

tp=root;

while((nn->dt<tp->dt && tp->l!=NULL) || (nn->dt>tp->dt && tp->r!=NULL)){

if(nn->dt<tp->dt && tp->l!=NULL)

tp=tp->l;

if(nn->dt>tp->dt && tp->r!=NULL)

tp=tp->r;

}

if(nn->dt<tp->dt)

tp->l=nn;

else

tp->r=nn;

}}

struct node\* search(struct node \*tp){

printf("Enter element: ");

scanf("%d",&s);

while(tp!=NULL){

if(tp->dt==s){

printf("Element available ");

return tp;

}

else if(s<tp->dt)

tp=tp->l;

else

tp=tp->r;

}

printf("Element not available");

return NULL;

}

void del(){

dn=search(root);

if(dn!=NULL){

if(dn->l!=NULL && dn->r!=NULL){

tp=dn->l;

nn=tp;

while(nn->r!=NULL){

tp=nn;

nn=tp->r;

}

if(tp!=nn)

tp->r=nn->l;

else

dn->l=nn->l;

dn->dt=nn->dt;

free(nn);

}

else{

if(dn==root){

if(root->l!=NULL)

root=root->l;

else

root=root->r;

}

else{

tp=root;

while(tp->l!=dn && tp->r!=dn){

if(s<tp->dt)

tp=tp->l;

else

tp=tp->r;

}

if (tp->l==dn){

if(dn->r!=NULL)

tp->l=dn->r;

else

tp->l=dn->l;

}

else{

if(dn->r!=NULL)

tp->r=dn->r;

else

tp->r=dn->l;

}}

free(dn);

dn=NULL;

}}

printf(" To Delete");}

void disp(struct node \*tp){

if(tp!=NULL){

disp(tp->l);

if(tp->dt>s) // To Display all the elements greater than the deleted element

printf("%d ", tp->dt);

disp(tp->r);

}}

int main() {

int ch;

do{

printf("\nEnter 0 to exit, 1 to insert, 2 to disp, 3 to search, 4 to del : ");

scanf("%d",&ch);

switch(ch){

case 0: break;

case 1: insert();

break;

case 2: disp(root);

break;

case 3: search(root);

break;

case 4: del();

break;

}}while(ch!=0);

return 0;

}

**// C program to insert and delete a node in AVL tree**

#include<stdio.h>

#include<stdlib.h>

struct node{

int dt, h;

struct node \*l, \*r;

}\*root=NULL,\*tp=NULL,\*nn=NULL,\*dn=NULL;

int height(struct node \*tp){

if (tp != NULL)

return tp->h;

return 0;

}

int max(int x, int y) {

return (x > y)? x : y;

}

struct node\* newNode(int key){

nn = malloc(sizeof(struct node));

nn->dt = key;

nn->l = NULL;

nn->r = NULL;

nn->h = 1;

return(nn);

}

struct node\* rightRotate(struct node \*tp){

nn = tp->l;

dn = nn->r;

nn->r = tp; // Perform rotation

tp->l = dn;

tp->h = max(height(tp->l), height(tp->r)) + 1;

nn->h = max(height(nn->l), height(nn->r)) + 1;

return nn; // Return new root

}

struct node \*leftRotate(struct node \*tp){

nn = tp->r;

dn = nn->l;

nn->l = tp; // Perform rotation

tp->r = dn;

tp->h = max(height(tp->l), height(tp->r)) + 1; //Update heights

nn->h = max(height(nn->l), height(nn->r)) + 1;

return nn; // Return new root

}

int BalFact(struct node \*tp) {

if (tp != NULL)

return height(tp->l) - height(tp->r);

return 0;

}

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

if (tp != NULL){

if (key < tp->dt)

tp->l = insert(tp->l, key);

if (key > tp->dt)

tp->r = insert(tp->r, key);

tp->h = 1 + max(height(tp->l), height(tp->r)); // Update height of ancestor node

if (BalFact(tp) > 1 && key < tp->l->dt) // Left Left Case

return rightRotate(tp);

if (BalFact(tp) < -1 && key > tp->r->dt) // Right Right Case

return leftRotate(tp);

if (BalFact(tp) > 1 && key > tp->l->dt){ // Left Right Case

tp->l = leftRotate(tp->l);

return rightRotate(tp);

}

if (BalFact(tp) < -1 && key < tp->r->dt){ // Right Left Case

tp->r = rightRotate(tp->r);

return leftRotate(tp);

}

return tp;

}

return(newNode(key));

}

void disp(struct node \*tp){

if(tp != NULL){

disp(tp->l);

printf("%d ", tp->dt);

disp(tp->r);

}}

struct node\* deleteNode(struct node\* dn, int del){

if (dn != NULL){//Similar to BST Delete

if ( del < dn->dt )// If delete element is smaller it lies in left subtree

dn->l = deleteNode(dn->l, del);

else if( del > dn->dt)// If delete element is greater it lies in right subtree

dn->r = deleteNode(dn->r, del);

else{//This is the node to be deleted

if(dn->l!=NULL && dn->r!=NULL){

tp=dn->l;

nn=tp;

while(nn->r!=NULL){

tp=nn;

nn=tp->r;

}

if(tp!=nn)

tp->r=nn->l;

else

dn->l=nn->l;

dn->dt=nn->dt;

free(nn);

}

else{

if(dn==root)

root = (root->l!=NULL)? root->l: root->r;

else{

tp=root;

while(tp->l!=dn && tp->r!=dn)

tp=(del<tp->dt)? tp->l : tp->r;

if (tp->l==dn)

tp->l=(dn->r!=NULL)? dn->r:dn->l;

else

tp->r=(dn->r!=NULL)? dn->r:dn->l;

}

free(dn);

dn=NULL;

}}

if (dn != NULL){

dn->h = 1 + max(height(dn->l), height(dn->r)); //UPDATE HEIGHT OF THE CURRENT NODE

if (BalFact(dn) > 1 && BalFact(dn->l) > 0) // Left Left Case

return rightRotate(dn);

if (BalFact(dn) > 1 && BalFact(dn->l) < 0){ // Left Right Case

dn->l = leftRotate(dn->l);

return rightRotate(dn);

}

if (BalFact(dn) < -1 && BalFact(dn->r) < 0) // Right Right Case

return leftRotate(dn);

if (BalFact(dn) < -1 && BalFact(dn->r) > 0){ // Right Left Case

dn->r = rightRotate(dn->r);

return leftRotate(dn);

}}

return dn;

}}

int main(){

root = insert(root, 1);

root = insert(root, 9);

root = insert(root, 2);

root = insert(root, 8);

root = insert(root, 3);

root = insert(root, 11);

root = insert(root, 91);

root = insert(root, 22);

root = insert(root, 84);

root = insert(root, 43);

root = insert(root, 76);

root = insert(root, 34);

root = insert(root, 16);

root = insert(root, 52);

root = insert(root, 7);

root = insert(root, 4);

root = insert(root, 6);

root = insert(root, 5);

root = deleteNode(root, 8);

root = deleteNode(root, 7);

root = deleteNode(root, 9);

disp(root);

printf("\nThe height of tree is %d",root->h);

return 0; }