#include<stdio.h>

#include<stdlib.h>

struct avltree

{

struct avltree \*left;

int data;

struct avltree \*right;

int height;

};

typedef struct avltree avltree;

int HeightAVL(avltree \*root)

{

int leftHeight,rightHeight;

if (root==NULL)

return -1;

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

return 0;

leftHeight = HeightAVL(root->left);

rightHeight= HeightAVL(root->right);

if (leftHeight>rightHeight)

return (leftHeight+1);

return (rightHeight+1);

}

void PreOrderAVL(avltree\* root)

{

if (root!=NULL)

{

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

PreOrderAVL(root->left);

PreOrderAVL(root->right);

}

}

void PostOrderAVL(avltree\* root)

{

if (root!=NULL)

{

PostOrderAVL(root->left);

PostOrderAVL(root->right);

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

}

}

void InOrderAVL(avltree\* root)

{

if (root!=NULL)

{

InOrderAVL(root->left);

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

InOrderAVL(root->right);

}

}

void Nodes\_level(avltree\* root,int level)

{

if(root == NULL)

return;

if(level == 0)

printf("%d\t",root->data);

else

{

Nodes\_level(root->left,level-1);

Nodes\_level(root->right,level-1);

}

}

void LevelOrderAVL(avltree \*root)

{

int level=0,h;

h = HeightAVL(root);

while (level<=h)

{

Nodes\_level(root,level);

level++;

}

}

avltree\* SmallestNodeAVL(avltree\* root)

{

avltree\* current = root;

while (current->left != NULL)

current = current->left;

return current;

}

int max(int a,int b)

{

if(a>b)

return(a);

return(b);

}

avltree \*leftrotate(avltree \*t)

{

avltree \*x,\*y;

x = t->right;

y = x->left;

x->left = t;

t->right = y;

t->height = max(HeightAVL(t->left),HeightAVL(t->right))+1;

x->height = max(HeightAVL(x->left),HeightAVL(x->right))+1;

return(x);

}

avltree \*rightrotate(avltree \*t)

{

avltree \*x,\*y;

x = t->left;

y = x->right;

x->right = t;

t->left = y;

t->height = max(HeightAVL(t->left),HeightAVL(t->right))+1;

x->height = max(HeightAVL(x->left),HeightAVL(x->right))+1;

return(x);

}

int get\_balance(avltree \*t)

{

if(t==NULL)

return(-1);

else

return(HeightAVL(t->left)-HeightAVL(t->right));

}

avltree\* AVLInsert(avltree \*t,int val)

{

int bf = 0;

if (t==NULL)

{

t = (avltree\*)malloc(sizeof(avltree));

t->data = val;

t->left = NULL;

t->right = NULL;

t->height = 0;

}

else

{

if(val<t->data)

t->left = AVLInsert(t->left,val);

else

t->right = AVLInsert(t->right,val);

}

t->height = max(HeightAVL(t->left),HeightAVL(t->right))+1;

bf = get\_balance(t);

if(bf > 1 && val < t->left->data)

return(rightrotate(t));

else if(bf > 1 && val > t->left->data)

{

t->left = leftrotate(t->left);

return(rightrotate(t));

}

else if (bf < -1 && val > t->right->data)

return(leftrotate(t));

else if (bf < -1 && val < t->right->data)

{

t->right = rightrotate(t->right);

return(leftrotate(t));

}

return(t);

}

avltree \*AVLDelete(avltree \*t,int val)

{

avltree \*tmp;

int bf;

if (t==NULL)

return t;

if(val<t->data)

t->left = AVLDelete(t->left,val);

else if(val>t->data)

t->right = AVLDelete(t->right,val);

else

{

if(t->left && t->right)

{

tmp = SmallestNodeAVL(t->right);

t->data = tmp->data;

t->right = AVLDelete(t->right,tmp->data);

}

else

{

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

{

tmp = t;

t = NULL;

}

else

{

tmp = t->left ? t->left:t->right;

\*t = \*tmp;

}

free(tmp);

}

}

if (t == NULL)

return t;

t->height = max(HeightAVL(t->left),HeightAVL(t->right))+1;

bf = get\_balance(t);

if(get\_balance(t)>1 && get\_balance(t->left)>=0)

rightrotate(t);

else if(get\_balance(t)>1 && get\_balance(t->left)<0)

{

t->left = leftrotate(t->left);

return(rightrotate(t));

}

else if(get\_balance(t)<-1 && get\_balance(t->right)<=0)

return(leftrotate(t));

else if(get\_balance(t)<-1 && get\_balance(t->right)>0)

{

t->right = rightrotate(t->right);

return(leftrotate(t));

}

return(t);

}

void main()

{

int choice, c2=-999, h=0, ele;

avltree \*avl=NULL;

while (1)

{

printf("\nEnter choice: 1.Insertion. 2.Deletion. 3.Traverse. 4.Height. 5.Exit.");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

printf("\nEnter the element to be inserted: ");

scanf("%d",&ele);

avl = AVLInsert(avl,ele);

break;

}

case 2:

{

printf("\nEnter the element to be deleted: ");

scanf("%d",&ele);

avl = AVLDelete(avl,ele);

if (avl == NULL)

printf("\nThe data is not found. ");

break;

}

case 3:

{

c2 = -9;

while (c2 != 5)

{

printf("\nEnter choice: 1.Pre Order. 2.Post Order. 3.In Order. 4.Level Order. 5.Exit. ");

scanf("%d",&c2);

printf("\nData is :");

switch(c2)

{

case 1:

PreOrderAVL(avl);

break;

case 2:

PostOrderAVL(avl);

break;

case 3:

InOrderAVL(avl);

break;

case 4:

LevelOrderAVL(avl);

break;

}

}

break;

}

case 4:

{

h = HeightAVL(avl);

printf("\nThe Height of the tree is: %d.\n",h);

break;

}

case 5:

{

exit(0);

}

}

}

}