avltree.c

```
#include <stdio.h>
#include <stdlib.h>
#define max(a,b) ((a)>(b)?(a):(b))
struct Node {
    int key, height;
    struct Node *left, *right;
};
int height(struct Node *n){return n?n->height:0;}
struct Node* newNode(int key){
    struct Node* n=malloc(sizeof(struct Node));
    n->key=key; n->height=1; n->left=n->right=NULL;
    return n;
}
struct Node* rightRotate(struct Node *y){
    struct Node *x=y->left,*T2=x->right;
    x->right=y; y->left=T2;
    y->height=max(height(y->left),height(y->right))+1;
    x->height=max(height(x->left),height(x->right))+1;
    return x;
}
struct Node* leftRotate(struct Node *x){
    struct Node *y=x->right,*T2=y->left;
    y->left=x; x->right=T2;
    x->height=max(height(x->left),height(x->right))+1;
    y->height=max(height(y->left),height(y->right))+1;
    return y;
}
int getBalance(struct Node* n){return n?height(n->left)-heigl
struct Node* insert(struct Node* node,int key){
    if(!node) return newNode(key);
    if(key<node->key) node->left=insert(node->left,key);
    else if(key>node->key) node->right=insert(node->right,key)
    else return node;
    node->height=1+max(height(node->left),height(node->righ
    int balance=getBalance(node);
    if(balance>1 && key<node->left->key) return rightRotate(n
    if(balance<-1 && key>node->right->key) return leftRotate(r
    if(balance>1 && key>node->left->key){ node->left=leftRotat
    if(balance<-1 && key<node->right->key){ node->right=righ
    return node;
}
struct Node* minValueNode(struct Node* node){
    while(node->left) node=node->left;
    return node;
}
struct Node* delete(struct Node* root,int key){
    if(!root) return root;
```

avltree.c

```
    return node;
}
struct Node* delete(struct Node* root,int key){
    if(!root) return root;
    if(key<root->key) root->left=delete(root->left,key);
    else if(key>root->key) root->right=delete(root->right,key);
    else{
        if(!root->left||!root->right){
            struct Node* tmp=root->left?root->left:root->right;
            if(!tmp){tmp=root; root=NULL;}
            else *root=*tmp;
            free(tmp);
        } else{
            struct Node* tmp=minValueNode(root->right);
            root->key=tmp->key;
            root->right=delete(root->right,tmp->key);
        }
    }
    if(!root) return root;
    root->height=1+max(height(root->left),height(root->right))
    int balance=getBalance(root);
    if(balance>1 && getBalance(root->left)>=0) return rightRot
    if(balance>1 && getBalance(root->left)<0){ root->left=leftR
    if(balance<-1 && getBalance(root->right)<=0) return leftRo
    if(balance<-1 && getBalance(root->right)>0){ root->right=r
    return root;
}
int search(struct Node* root,int key){
    if(!root) return 0;
    if(key==root->key) return 1;
    if(key<root->key) return search(root->left,key);
    return search(root->right,key);
}
void preorder(struct Node* root){
    if(root){printf("%d ",root->key);preorder(root->left);preord
}
int main(){
    struct Node* root=NULL; int ch,v;
    do{
        printf("1.Insert 2.Delete 3.Search 4.Display 5.Exit\n"); sc
        if(ch==1){scanf("%d",&v);root=insert(root,v);}
        else if(ch==2){scanf("%d",&v);root=delete(root,v);}
        else if(ch==3){scanf("%d",&v);printf(search(root,v)?"Fou
        else if(ch==4){preorder(root);printf("\n");}
    }while(ch!=5);
    return 0;
}
```