AVL Delete

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

struct Node
{
    struct Node *lchild;
    int data;
    int height;
    struct Node *rchild;
}*root=NULL;

int Height(struct Node *p,struct Node *q)
{
    int x,y;
    x=p?p->height:-1;
    y=q?q->height:-1;
    return x>y?x+1:y+1;
}
int BF(struct Node *p)
{
    int hl,hr;
    if(!p)return -1;
    hl=p->lchild?p->lchild->height:-1;
    hr=p->rchild?p->rchild->height:-1;
    return hl-hr;
}

void Insert(int key)
{
    struct Node *t=root;
    struct Node *r=NULL,*p;

    if(root==NULL)
    {
        p=(struct Node *)malloc(sizeof(struct Node));
        p->data=key;
```

```
        p->height=0;
        p->lchild=p->rchild=NULL;
        root=p;
        return;
    }
    while(t!=NULL)
    {
        r=t;
        if(key<t->data)
            t=t->lchild;
        else if(key>t->data)
            t=t->rchild;
        else
            return;
    }
    p=(struct Node *)malloc(sizeof(struct Node));
    p->data=key;
    p->lchild=p->rchild=NULL;

    if(key<r->data) r->lchild=p;
    else r->rchild=p;
}

struct Node * LLRotation(struct Node *p)
{
    struct Node *pl=p->lchild;
    pl->height=0;
    p->lchild=pl->rchild;
    pl->rchild=p;
    p->height=Height(p->lchild,p->rchild);
    if(p==root)root=pl;
    return pl;
}
struct Node *LRRotation(struct Node *p)
{
    struct Node *pl=p->lchild;
    struct Node *plr=pl->rchild;
    plr->height=0;

    p->lchild=plr->rchild;
```

```
    pl->rchild=plr->lchild;
    plr->lchild=pl;
    plr->rchild=p;
    p->height=Height(p->lchild,p->rchild);

    pl->height=Height(pl->lchild,pl->rchild);
    if(p==root)root=plr;
    return plr;
}
struct Node *RRRotation(struct Node *p)
{

    struct Node *pr=p->rchild;
    pr->height=0;
    p->rchild=pr->lchild;
    pr->lchild=p;
    p->height=Height(p->lchild,p->rchild);
    if(p==root)root=pr;
    return pr;
}
struct Node *RLRotation(struct Node *p)
{
    struct Node *pr=p->rchild;
    struct Node *prl=pr->lchild;
    prl->height=0;

    p->rchild=prl->lchild;
    pr->lchild=prl->rchild;
    prl->rchild=pr;
    prl->lchild=p;
    p->height=Height(p->lchild,p->rchild);

    pr->height=Height(pr->lchild,pr->rchild);
    if(p==root)root=prl;
    return prl;
}


struct Node* RInsert(struct Node *p,int key)
{
```

```
    struct Node *t;

    if(p==NULL)
    {
        t=(struct Node *)malloc(sizeof(struct Node));
        t->data=key;
        t->height=0;
        t->lchild=t->rchild=NULL;
        return t;
    }
    if(key<p->data)
        p->lchild=RInsert(p->lchild,key);
    else if(key>p->data)
        p->rchild=RInsert(p->rchild,key);

    p->height=Height(p->lchild,p->rchild);

    if(BF(p)==2 && BF(p->lchild)==1)
        return LLRotation(p);
    if(BF(p)==2 && BF(p->lchild)==-1)
        return LRRotation(p);
    if(BF(p)==-2 && BF(p->rchild)==-1)
        return RRRotation(p);
    if(BF(p)==-2 && BF(p->rchild)==1)
        return RLRotation(p);
    return p;
}

void Inorder(struct Node *p)
{
    if(p)
    {
        Inorder(p->lchild);
        printf("%d ",p->data);
        Inorder(p->rchild);
    }
}

struct Node * Search(int key)
{
```

```
    struct Node *t=root;

    while(t!=NULL)
    {
        if(key==t->data)
            return t;
        else if(key<t->data)
            t=t->lchild;
        else
            t=t->rchild;
    }
    return NULL;
}

struct Node * InPre(struct Node *p)
{
    while(p && p->rchild!=NULL)
        p=p->rchild;
    return p;
}
struct Node * InSucc(struct Node *p)
{
    while(p && p->lchild!=NULL)
        p=p->lchild;
    return p;
}
int HT(struct Node *p)
{
    int x,y;
    if(!p)return 0;
    x=HT(p->lchild);
    y=HT(p->rchild);
    return x>y?x+1:y+1;
}
struct Node* Delete(struct Node *p,int key)
{
    struct Node *q;

    if(p==NULL)
        return NULL;
```

```
    if(p->lchild==NULL && p->rchild==NULL)
    {
        if(p==root)
            root=NULL;
        free(p);
        return NULL;
    }
    if(key<p->data)
        p->lchild=Delete(p->lchild,key);
    else if(key>p->data)
        p->rchild=Delete(p->rchild,key);
    else
    {
        if((p->lchild && p->lchild->height)> (p-
>rchild && p->rchild->height))
        {
            q=InPre(p->lchild);
            p->data=q->data;
            p->lchild=Delete(p->lchild,p->data);
        }
        else
        {
            q=InSucc(p->rchild);
            p->data=q->data;
            p->rchild=Delete(p->rchild,p->data);
        }
    }

    p->height=Height(p->lchild,p->rchild);

    if(BF(p)==2)
    {
        if(BF(p->lchild)==1)
            return LLRotation(p);
        else
            return LRRotation(p);
    }
    if(BF(p)==-2)
    {
       if(BF(p->lchild)==-1)
```

```
            return RRRotation(p);
        else
            return RLRotation(p);
    }
    return p;

}


int main()
{
    struct Node *temp;

    Insert(10);
    RInsert(root,20);
    RInsert(root,50);
    RInsert(root,15);
    RInsert(root,12);
   // RInsert(root,5);
   // RInsert(root,4);
    Delete(root,20);
    Inorder(root);
    printf("\n");

    temp=Search(2);
    if(temp!=NULL)
        printf("element %d is found\n",temp->data);
    else
        printf("element is not found\n");

    return 0;
}
```