BST Insert and Delete

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

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

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->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;
}

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 *RInsert(struct Node *p,int key)
{
    struct Node *t=NULL;

    if(p==NULL)
    {
        t=(struct Node *)malloc(sizeof(struct Node));
        t->data=key;
        t->lchild=t->rchild=NULL;
        return t;
```

```c
    }
    if(key < p->data)
        p->lchild=RInsert(p->lchild,key);
    else if(key > p->data)
        p->rchild=RInsert(p->rchild,key);

    return p;


}

int Height(struct Node *p)
{
    int x,y;
    if(p==NULL)return 0;
    x=Height(p->lchild);
    y=Height(p->rchild);
    return x>y?x+1:y+1;
}

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;
}
```

```
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(Height(p->lchild)>Height(p->rchild))
        {
            q=InPre(p->lchild);
            p->data=q->data;
            p->lchild=Delete(p->lchild,q->data);
        }
        else
        {
            q=InSucc(p->rchild);
            p->data=q->data;
            p->rchild=Delete(p->rchild,q->data);
        }


    }
    return p;
```

```c
}


int main()
{
    struct Node *temp;

    root=RInsert(root,50);
    RInsert(root,10);
    RInsert(root,40);
    RInsert(root,20);
    RInsert(root,30);

    Delete(root,30);

    Inorder(root);
    printf("\n");

    temp=Search(20);
    if(temp!=NULL)
        printf("element %d is found\n",temp->data);
    else
        printf("element is not found\n");

    return 0;
}
```