```cpp
//日期：2018/ 时间：
/*
二叉查找树：
查找：        向两边查找，知道找到相等的元素
插入：        在查找失败的点处，新建节点，并插入原来的树
创建二叉查找树：不断插入元素
删除：        叶节点，直接删除；  左子树不为空，则寻找左子树的大节点，删除之；右子树不为空，则寻找右子树的最小节点，删除之。
            找不到，则返回

*/
#include <stdio.h>
#include <stdlib.h>
#include <queue>
using namespace std;

struct node{
    int data;
    node* lchild;
    node* rchild;
    int layer;      //层次遍历时用到
};

node* newNode(int x){
    node* Node = new node;
    Node->data = x;
    Node->lchild = Node->rchild = NULL;
    return Node;
}

//查找
void search(node* root,int x){
    if(root == NULL){
        printf("search failed\n");
        return;
    }
    if(x == root->data){
        printf("%d\n",root->data);
    } else if(x < root->data){
        search(root->lchild,x);
    } else if(x > root->data){
        search(root->rchild,x);
    }
}

//查找失败的点就是需要插入数据的地方，root==NULL时需要新建插入的节点
void insert(node* &root,int x){     //注意：当return发生时，return到最好，root还是起初的那个root。所以根节点并没有改变
    if(root == NULL){
        root = newNode(x);
        return;
    }
    if(x == root->data){        //查找成功说明节点已经存在
        return;
    } else if(x < root->data){
        insert(root->lchild,x);
```

```
55      } else if(x > root->data){
56          insert(root->rchild,x);
57      }
58  }
59
60  //二叉树的建立
61  node* Create(int data[],int n){
62      node* root = NULL;
63      for(int i=0;i<n;i++)
64          insert(root,data[i]);
65      return root;
66  }
67
68  node* findMax(node* root){              //找出root为根节点的树中的最大权值节点
69      while(root->rchild!=NULL){
70          root=root->rchild;              //不断往右，直到没有右孩子
71      }
72      return root;
73  }
74
75  node* findMin(node* root){              //找出root为根节点的树中最小权值节点
76      while(root->lchild!=NULL){
77          root=root->lchild;
78      }
79      return root;
80  }
81
82  //删除以root为根节点的树中权值为x的节点
83  void deleteNode(node* &root,int x){
84      if(root == NULL) return;    //不存在权值为x的节点
85      if(root->data == x){        //找到欲删除节点
86          if(root->lchild == NULL && root->rchild==NULL){     //叶子节点直接删除
87              root = NULL;        //把root的地址设为NULL,父节点就引用不到他了
88          }else if(root->lchild != NULL){         //左子树不为空
89              node* pre = findMax(root->lchild);  //找root前驱
90              root->data = pre->data;     //用前驱覆盖root
91              deleteNode(root->lchild,pre->data); //在左子树中删除节点pre
92          }else{                                  //右子树不为空
93              node* next = findMin(root->rchild);
94              root->data = next->data;
95              deleteNode(root->rchild,next->data);
96          }
97      }else if(root->data > x){   //欲删除的节点在左子树中
98          deleteNode(root->lchild,x);
99      } else{
100         deleteNode(root->rchild,x);
101     }
102 }
103
104 int main(){
105     int num[10];
106     for(int i=0;i<10;i++)
107         num[i]=i+1;
108     node* root = Create(num,10);
109     //deleteNode(root,5);
110     search(root,5);
```

```
111
112     printf("%d",root->data);
113
114     return 0;
115 }
116
117
```