**实 验 九：二叉排序树的构造、删除与查找（2学时）**

**[问题描述]**

**按照输入的数列顺序进行插入构造二叉排序树，并完成查找。**

**[实验目的]**

1. 掌握二叉排序树的构造过程。
2. 掌握二叉树的查找过程。
3. 掌握二叉树排序树删除结点的过程。

**[实验内容及要求]**

（1）构造具有n结点的二叉排序树，并完成对某个输入数据的查找。

（2）删除给定结点后，使得删除后仍是二叉排序树。

**[测试数据]**

**#include** <stdio.h>

**typedef** ***int*** *KeyType*;

**typedef** ***int*** *InfoType*;

**typedef** ***struct***

{

*KeyType* key;

*InfoType* otherinfo;

} *ElemType*;

**typedef** ***struct*** *BSTNode*

{

*ElemType* data;

***struct*** *BSTNode* \*lchild, **\***rchild;

} *BSTNode*, \**BSTree*;

*BSTree* SearchBST(*BSTree* *T*, *KeyType* *kval*) //查找

{

**if** (**!***T*)

**return** *T*;

**if** (*kval* **==** *T*->data.key) //查找成功

  {

    printf("查找成功！\n");

**return** *T*;

  }

**else** **if** (*kval* **<** *T*->data.key)

**return** SearchBST(*T*->lchild, *kval*); //在左子树中继续查找

**else**

**return** SearchBST(*T*->rchild, *kval*); //在右子树中继续查找

}

***void*** Insert\_BST(*BSTree* &*T*, *ElemType* *e*) //插入

{                                      //当二叉排序树中不存在关键字等于e.key的数据元素时，插入元素值为e的结点；否则，不进行插入

*BSTNode* **\***s;

  s **=** **new** *BSTNode*; //为新结点分配空间

  s->data.key **=** *e*.key;

  s->lchild **=** NULL;

  s->rchild **=** NULL;

**if** (**!***T*)

*T* **=** s;

**if** (*T*->data.key **==** *e*.key)

**return**;

**else** **if** (*e*.key **<** *T*->data.key)

    Insert\_BST(*T*->lchild, *e*); //插到左子树

**else** **if** (*e*.key **>** *T*->data.key)

    Insert\_BST(*T*->rchild, *e*); //插到右子树

}

***void*** DeleteBST(*BSTree* &*T*, *KeyType* *kval*) //删除

{

*BSTree* f **=** NULL, p, q, s;

  p **=** *T*;

**while** (p **!=** NULL **&&** p->data.key **!=** *kval*)

  {

    f **=** p;

**if** (*kval* **<** p->data.key)

      p **=** p->lchild;

**else**

      p **=** p->rchild;

  }

**if** (p **==** NULL)

**return**;

  s **=** p;

**if** (p->lchild **&&** p->rchild)

  {

    f **=** p;

    s **=** p->lchild;

**while** (s->rchild)

    {

      f **=** s;

      s **=** s->rchild;

    }

    p->data.key **=** s->data.key;

    p->data.otherinfo **=** s->data.otherinfo;

  }

**if** (s->lchild)

    q **=** s->lchild;

**else**

    q **=** s->rchild;

**if** (f **==** NULL)

*T* **=** q;

**else** **if** (f->lchild **==** s)

    f->lchild **=** q;

**else**

    f->rchild **=** q;

**delete** s;

}

***void*** InOrderTraverse(*BSTree* *T*) // 递归中序遍历

{

**if** (*T*)

  {

    InOrderTraverse(*T*->lchild);

    printf("%d", *T*->data.key);

    InOrderTraverse(*T*->rchild);

  }

}

***int*** main()

{

*BSTree* T **=** NULL;

*BSTNode* **\***p;

*ElemType* key;

***int*** m, kval;

  printf("请输入组成二叉排序树的元素个数：");

  scanf("%d", **&**m);

  printf("请输入组成二叉排序树的元素：");

**for** (***int*** i **=** 0; i **<** m; i**++**)

  {

    scanf("%d", **&**key.key);

    Insert\_BST(T, key);

  }

  // 输出

  printf("二叉排序树为：\n");

  InOrderTraverse(T);

  printf("\n");

  // 查找

  printf("请输入要查找的数：");

  scanf("%d", **&**kval);

  p **=** SearchBST(T, kval);

  // 删除

  printf("请输入要删除的数：");

  scanf("%d", **&**kval);

  DeleteBST(T, kval);

  // 再输出

  printf("删除后的二叉排序树为：\n");

  InOrderTraverse(T);

**return** 0;

}

****