```
BiTree SearchBST(BiTree T,KeyType key){
    //如果递归过程中 T 为空，则查找结果，返回NULL；或者查找成功，返回指向该关键字的指针
    if (!T || key==T->data) {
        return T;
    }else if(key<T->data){
        //递归遍历其左孩子
        return SearchBST(T->lchild, key);
    }else{
        //递归遍历其右孩子
        return SearchBST(T->rchild, key);
    }
}
```

```
BOOL SearchBST(BiTree T,KeyType key,BiTree f,BiTree *p){
    //如果 T 指针为空，说明查找失败，令 p 指针指向查找过程中最后一个叶子结点，并返回查找失败的信息
    if (!T){
        *p=f;
        return false;
    }
    //如果相等，令 p 指针指向该关键字，并返回查找成功信息
    else if(key==T->data){
        *p=T;
        return true;
    }
    //如果 key 值比 T 根结点的值小，则查找其左子树；反之，查找其右子树
    else if(key<T->data){
        return SearchBST(T->lchild,key,T,p);
    }else{
        return SearchBST(T->rchild,key,T,p);
    }
}
//插入函数
BOOL InsertBST(BiTree T,ElemType e){
    BiTree p=NULL;
    //如果查找不成功，需做插入操作
    if (!SearchBST(T, e,NULL,&p)) {
        //初始化插入结点
        BiTree s=(BiTree)malloc(sizeof(BiTree));
        s->data=e;
        s->lchild=s->rchild=NULL;
```

```
        //如果 p 为NULL，说明该二叉排序树为空树，此时插入的结点为整棵树的根结点
        if (!p) {
            T=s;
        }
        //如果 p 不为 NULL，则 p 指向的为查找失败的最后一个叶子结点，只需要通过比较
p 和 e 的值确定 s 到底是 p 的左孩子还是右孩子
        else if(e<p->data){
            p->lchild=s;
        }else{
            p->rchild=s;
        }
        return true;
    }
    //如果查找成功，不需要做插入操作，插入失败
    return false;
}
```