```
BinTree Insert( BinTree BST, ElementType X )
{
    if( !BST ){ /* 若原树为空，生成并返回一个结点的二叉搜索树 */
        BST = (BinTree)malloc(sizeof(struct TNode));
        BST->Data = X;
        BST->Left = BST->Right = NULL;
    }
    else { /* 开始找要插入元素的位置 */
        if( X < BST->Data )
            BST->Left = Insert( BST->Left, X );   /*递归插入左子树*/
        else  if( X > BST->Data )
            BST->Right = Insert( BST->Right, X ); /*递归插入右子树*/
        /* else X已经存在，什么都不做 */
    }
    return BST;
}

BinTree Delete( BinTree BST, ElementType X )
{
    Position Tmp;

    if( !BST )
        printf("要删除的元素未找到");
    else {
        if( X < BST->Data )
            BST->Left = Delete( BST->Left, X );   /* 从左子树递归删除 */
        else if( X > BST->Data )
            BST->Right = Delete( BST->Right, X ); /* 从右子树递归删除 */
        else { /* BST就是要删除的结点 */
            /* 如果被删除结点有左右两个子结点 */
            if( BST->Left && BST->Right ) {
                /* 从右子树中找最小的元素填充删除结点 */
                Tmp = FindMin( BST->Right );
                BST->Data = Tmp->Data;
                /* 从右子树中删除最小元素 */
                BST->Right = Delete( BST->Right, BST->Data );
            }
            else { /* 被删除结点有一个或无子结点 */
                Tmp = BST;
                if( !BST->Left )       /* 只有右孩子或无子结点 */
                    BST = BST->Right;
                else                   /* 只有左孩子 */
                    BST = BST->Left;
                free( Tmp );
            }
        }
    }
    return BST;
}
```