数据结构 hw7

刘良宇 PB20000180

6.38

利用标志栈 S_flag。 0 表示在访问左子树，1 表示在访问右子树。

```
Status PostOrderTraverse(BiTree T, Status (*Visit)(TElemType e)) {
    InitStack(S_tree);
    BiTree p = T;
    InitStack(S_flag);
    int flag = 0;
    while (p || !StackEmpty(S_tree)) {
        if (p) {
            Push(S_tree, p);
            Push(S_flag, flag);
            p = flag == 0 ? p->lchild : p->rchild;
            flag = 0;
        } else {
            Pop(S_tree, p);
            Pop(S_flag, flag);
            // 出栈可能是因为这个结点的左子树访问完了（接下来访问右子树）
            // 也可能是这个结点的右子树访问完了（接下来访问它自身）
            // 用 flag 的值判断
            if (flag == 0) {
                Push(S_tree, p);
                Push(S_flag, 1);  // 接下来要访问的是这个结点的右子树
                p = p->rchild;
                flag = 0;
                continue;
            }
            // 否则 flag 为 1，访问自身，维持退栈即可
            if (!Visit(p->data)) {
                return ERROR;
            }
            p = nullptr;
        }
    }
    return OK;
}
```

6.47

采用 BFS 搜索即可。

```
Status Traverse(Bitree T, Status (*Visit)(TElemType e)) {
    InitQueue(Q);
    if (!T)
        return;
    EnQueue(T);
    while (!QueueEmpty(Q)) {
        DeQueue(Q, temp);
```

```
        if (!Visit(temp)) {
            return ERROR;
        }
        if (temp->lchild) {
            EnQueue(Q, temp->lchild);
        }
        if (temp->rchild) {
            EnQueue(Q, temp->rchild);
        }
    }
    return OK;
}
```

6.58

当 p 有左子树时，这里的理解是：先把 p 的左子树移动为 x 的右子树，再把 x 整体作为 p 的左子树

```
Status ThreadInsert(ThrBiTree p, ThrBiTree x) {
    if (p->ltag == 1) {  // 无左子树
        // x 的后继修改为 p
        x->rchild = p;
        // x 的第一个结点的前驱修改为 p 的前驱
        x_first = x;
        while (x_first->ltag == 0 && x_first->lchild) {
            x_first = x_first->lchild;
        }
        x_first->lchild = p->lchild;
        // 修改树结构
        p->lchild = x; p->ltag = 0;
    } else {
        // 此时 p 有左子树
        // 相当于在原来 p 的第一个结点的前驱和原来 p 的第一个结点之间插入了 x
        // 首先找到 p 的第一个结点和 x 的第一个结点
        p_first = p;
        while (p_first->ltag == 0 && p_first->lchild) {
            p_first = p_first->lchild;
        }
        x_first = x;
        while (x_first->ltag == 0 && x_first->lchild) {
            x_first = x_first->lchild;
        }
        // 然后插入
        x_first->lchild = p_first->lchild;
        p_first->lchild = x;
        // 最后修改树结构
        p->lchild = x;
        x->rchild = p; x->rtag = 0;
    }
}
```