**二叉树的非递归遍历（考察能力）**

**一、**

**//先序**

**//借助栈结构 栈是先进后出 所以要先遍历左树就要先把右树入栈，左树才能先出栈**

**//出栈一个数据，结点类型的指针t指向出栈的数据，将所指向结点的右左两侧数据入栈。循环上述操作。**

void PreOrder\_1(BinTreeNode\*& t) {

if (t != nullptr) {

SeqStack st;

InitStack(st);

BinTreeNode\* p;

Push(st, t);

while (!IsEmpty(st)) {

GetTop(st, p);

Pop(st);

cout << p->data;

if (p->rightChild != nullptr) {

Push(st, p->rightChild);

}

if (p->leftChild != nullptr) {

Push(st, p->leftChild);

}

}

}

cout << endl;

}

void PreOrder\_1(BinTree& bt) {

PreOrder\_1(bt.root);

}

**二、**

**//中序**

**//画图慢慢分析就能解决**

void InOrder\_1(BinTreeNode\*& t) {

if (t != nullptr) {

SeqStack st;

InitStack(st);

BinTreeNode\* p;

Push(st, t);

while (!IsEmpty(st)) {

while (t != nullptr && t->leftChild != nullptr) {

Push(st, t->leftChild);

t = t->leftChild;

}

GetTop(st, p);

Pop(st);

cout << p->data;

//上方的代码执行完了左树，说明找到最左边的结点

//如果右树为空则不执行以下代码，（如果不为空，则说明上面找到的左结点是另外一个小子树的根结点，已经打印过了，下面轮到小子树的右树了）则从GetTop(st, p)继续执行，说明下面该打印的是子树的根结点

if (p->rightChild != nullptr) {

t = p->rightChild; //这一段检验右树是关键

Push(st, t);

}

}

}

cout << endl;

}

void InOrder\_1(BinTree & bt) {

InOrder\_1(bt.root);

}

**三、**

**//后序(最复杂的)**

**//如果一个结点左子树为空，则还不能直接打印，因为其可能是根结点（存在右子树），因此要在检验完右子树也为空的情况下才能打印。**

**//因此要对根的访问加上左标记和右标记（看是由左标记退回还是右标记退回）**

typedef enum{L,R}Tag;

#define ElemType StkNode //后序访问入栈的不是地址而是新定义的一个结构体（结点地址+标记符）

typedef struct StkNode {

BinTreeNode\* ptr;

Tag tag;

}StkNode;

void PostOrder\_1(BinTreeNode\*& t) {

BinTreeNode\* p;

if (t != nullptr) { //根结点不空（二叉树存在）才有意义

SeqStack st;

InitStack(st);

StkNode sn;

do { //借助do—while循环 至少执行一次 保证根结点能入栈

while(t!=nullptr){

sn.ptr = t; //每次要压栈前就初始化结点

sn.tag = L; //给的初始标记为左，访问左树

Push(st, sn);

t = t->leftChild; //循环条件相当于左树不空继续入栈

}

bool flag = true; //是否继续出栈的标记

while (flag && !IsEmpty(st)) {

GetTop(st, sn);

Pop(st); //先拿出栈顶元素考察

p = sn.ptr;

switch (sn.tag) {

case L:

sn.tag = R; //要观察右标记

Push(st, sn); //重新入栈

flag = false;

t = p->rightChild; //这里如果右边为空 则不执行上面的while循环 说明没有左树 即不再访问

break;

case R:

cout << p->data;

break;

}

}

} while (!IsEmpty(st));

}

}

void PostOrder\_1(BinTree& bt) {

PostOrder\_1(bt.root);

}