```
/*-----------------------------------------+
 |  leetcode 144 pre_order                 |
 |  leetcode 145 post_order                |
 |  leetcode 94  in_order                  |
 +-----------------------------------------*/

<PART 1.>
对于二叉树的遍历，递归式遍历很好写出
/*前序：根->左->右*/
void _pre_recur(TreeNode *root, vector<int>&ans)
{
        if (root){
                ans.push_back(root->val);
                _pre_recur(root->left, ans);
                _pre_recur(root->right, ans);
            }
}
/*后序：左->右->根*/
void _post_recur(TreeNode *root,vector<int>&ans)
{
        if (root){
                _post(root->left,ans);
                ans.push_back(root->val);
                _post(root->right,ans);
        }
}
/*中序：左->根->右*/
void _in_recur(TreeNode *root,vector<int>&ans)
{
        if(root){
                _in_recur(root->left,ans);
                ans.push_back(root->val);
                _in_recur(root->right,ans);
        }
}

<PART 2.>
对于迭代式的遍历，则需要借助栈这一结构。首先对于前序遍历而言，每次处理的都是
根节点，然后对其左右子树操作，因此，由于栈的结构原因，入栈的时候先入栈右子树
因此就可以得到前序序列。
void _pre_it(TreeNode *root, vector<int>&ans)
{
        stack<TreeNode*> s;
        s.push(root);
        while (!s.empty()){
                TreeNode *cur=s.top();
                s.pop();
                if (cur!= nullptr){
                        ans.push_back(cur->val);
                        s.push(cur->right);
                        s.push(cur->left);
                }
        }
}
对于后序遍历，只需入栈的顺序改一下，先入栈左节点，从而ans中便是根->右->左
再将ans反转便是左->右->根
void _post_it(TreeNode *root,vector<int>&ans)
{
        stack<TreeNode*>s;
        s.push(root);
        while (!s.empty()){
                TreeNode *cur=s.top();
                s.pop();
                if (cur!= nullptr){
                        ans.push_back(cur->val);
                        s.push(cur->left);
                        s.push(cur->right);
                }
        }
        reverse(ans.begin(),ans.end());
}
但是对于中序遍历，这个过程有些许的变化，不仅需要栈，还需要指针的遍历。中序
遍历首先需要找到最左边的节点。之后取出这个节点，往右边走。
```

```cpp
void _in_it(TreeNode *root,vector<int>&ans)
{
        stack<TreeNode*>s;
        TreeNode *cur=root;
        while (cur!= nullptr||!s.empty()){
                if(cur!=nullptr){
                        s.push(cur);
                        cur=cur->left;
                }else{
                        /*最左边的节点*/
                        TreeNode *leftest=s.top();
                        ans.push_back(leftest->val);
                        s.pop();
                        cur=leftest->right;
                }
        }
}

<PART 3.>
然而，如果没有统一的套路，迭代式的写法就有些许麻烦，可以使用标记法，将迭代
纳入统一的框架。也就是在需要处理的节点后面放入一个空指针进行标记，在碰到空
节点时就统一处理该节点。
void _pre_uni(TreeNode *root,vector<int>&ans)
{
        stack<TreeNode *> s;
        if (root) s.push(root);
        while (!s.empty()) {
                TreeNode *cur = s.top();
                if (cur) {
                        s.pop();
                        /*右*/
                        if (cur->right) s.push(cur->right);
                        /*左*/
                        if (cur->left) s.push(cur->left);
                        /*根*/
                        s.push(cur);
                        s.push(nullptr);
                } else {
                        /*统一处理*/
                        s.pop();
                        cur = s.top();
                        s.pop();
                        ans.push_back(cur->val);
                }
        }
} « end _pre_uni »
void _post_uni(TreeNode *root,vector<int>&ans)
{
        stack<TreeNode*>s;
        if (root) s.push(root);
        while(!s.empty()){
                TreeNode *cur=s.top();
                if (cur){
                        s.pop();
                        s.push(cur);
                        s.push(nullptr);
                        if (cur->right) s.push(cur->right);
                        if (cur->left) s.push(cur->left);
                }else{
                        s.pop();
                        cur=s.top();
                        s.pop();
                        ans.push_back(cur->val);
                }
        }
} « end _post_uni »
void _in_uni(TreeNode *root,vector<int>&ans)
{
        stack<TreeNode*>s;
        if (root) s.push(root);
        while (!s.empty()){
                TreeNode *cur=s.top();
                if (cur){
```

```
147:                         s.pop();
148:                         if (cur->right) s.push(cur->right);
149:                         s.push(cur);
150:                         s.push(nullptr);
151:                         if (cur->left) s.push(cur->left);
152:                 }else{
153:                         s.pop();
154:                         cur=s.top();
155:                         s.pop();
156:                         ans.push_back(cur->val);
157:                 }
158:         }
159: } « end _in_uni »
160: 可以看到，只有入栈地方的次序不一样，别的地方都是一样的框架。
161:
```