

非递归框架

```

#include <stack>
#include <vector>

#include "treenode.h"

class Traver {
public:
    std::vector<int> traverse(TreeNode *root) {
        pushLeftBranch(root);
        TreeNode *visited = new TreeNode(); // 上一次遍历完成的根节点
        while (sk.size()) {
            TreeNode *t = sk.top();

            // 左子树遍历完成, 且右子树没有遍历
            if ((nullptr == t->left || visited == t->left) && t->right !=
visited) {
                // 中序遍历位置
                // 遍历右子树
                pushLeftBranch(t->right);
            }

            // 右子树遍历完成
            if (nullptr == t->right || visited == t->right) {
                // 后序遍历位置
                // 以p为根的子树遍历完成, 出栈
                // visited 指向当前栈顶
                visited = sk.top();
                sk.pop();
            }
        }

        return res;
    }

private:
    void pushLeftBranch(TreeNode *p) {
        while (p != nullptr) {
            // 前序代码遍历位置
            // res.push_back(p->val);
            sk.push(p);
            p = p->left;
        }
    }

    std::vector<int> res;

    std::stack<TreeNode *> sk; // 用于模拟系统递归栈
};

```

先序遍历改非递归

```

#include <stack>
#include <vector>

#include "treenode.h"

class Traver {
public:
    std::vector<int> traverse(TreeNode *root) {
        pushLeftBranch(root);
        TreeNode *visited = new TreeNode(); // 上一次遍历完成的根节点
        while (sk.size()) {
            TreeNode *t = sk.top();

            // 左子树遍历完成, 且右子树没有遍历
            if ((nullptr == t->left || visited == t->left) && t->right !=
visited) {
                // 中序遍历位置
                // 遍历右子树
                pushLeftBranch(root->right);
            }

            // 右子树遍历完成
            if (nullptr == t->right || visited == t->right) {
                // 后序遍历位置
                // 以p为根的子树遍历完成, 出栈
                // visited 指向当前栈顶
                visited = sk.top();
                sk.pop();
            }
        }

        return res;
    }

private:
    void pushLeftBranch(TreeNode *p) {
        while (p != nullptr) {
            // 前序代码遍历位置
            res.push_back(p->val);
            sk.push(p);
            p = p->left;
        }
    }

    std::vector<int> res;

    std::stack<TreeNode *> sk; // 用于模拟系统递归栈
};

```

后序遍历改非递归

```

#include <stack>
#include <vector>

#include "treenode.h"

class Traver {
public:
    std::vector<int> traverse(TreeNode *root) {
        pushLeftBranch(root);
        TreeNode *visited = new TreeNode(); // 上一次遍历完成的根节点
        while (sk.size()) {
            TreeNode *t = sk.top();

            // 左子树遍历完成, 且右子树没有遍历
            if ((nullptr == t->left || visited == t->left) && t->right !=
visited) {
                // 中序遍历位置
                // 遍历右子树
                pushLeftBranch(t->right);
            }

            // 右子树遍历完成
            if (nullptr == t->right || visited == t->right) {
                // 后序遍历位置
                // 以p为根的子树遍历完成, 出栈
                // visited 指向当前栈顶
                res.push_back(t->val);

                visited = sk.top();
                sk.pop();
            }
        }

        return res;
    }

private:
    void pushLeftBranch(TreeNode *p) {
        while (p != nullptr) {
            // 前序代码遍历位置
            sk.push(p);
            p = p->left;
        }
    }

    std::vector<int> res;

    std::stack<TreeNode *> sk; // 用于模拟系统递归栈
};

```

中序遍历改非递归

```

#include <stack>
#include <vector>

#include "treenode.h"

class Traver {
public:
    std::vector<int> traverse(TreeNode *root) {
        pushLeftBranch(root);
        TreeNode *visited = new TreeNode(); // 上一次遍历完成的根节点
        while (sk.size()) {
            TreeNode *t = sk.top();

            // 左子树遍历完成，且右子树没有遍历
            if ((nullptr == t->left || visited == t->left) && t->right !=
visited) {
                // 中序遍历位置
                // 遍历右子树
                res.push_back(t->val);

                pushLeftBranch(root->right);
            }

            // 右子树遍历完成
            if (nullptr == t->right || visited == t->right) {
                // 后序遍历位置
                // 以p为根的子树遍历完成，出栈
                // visited 指向当前栈顶

                visited = sk.top();
                sk.pop();
            }
        }

        return res;
    }

private:
    void pushLeftBranch(TreeNode *p) {
        while (p != nullptr) {
            // 前序代码遍历位置
            sk.push(p);
            p = p->left;
        }
    }

    std::vector<int> res;

    std::stack<TreeNode *> sk; // 用于模拟系统递归栈

```

```
};
```