```cpp
//BinaryTree
#include<iostream>
#include<stack>
#include<queue>
using namespace std;

class btNode {
    public:
        double value;
        btNode* left;
        btNode* right;
    public:
        btNode() {
            this->value=0.0;
            this->left=NULL;
            this->right=NULL;
        }
        btNode(double m_value=0.0) {
            this->value=m_value;
            this->left=NULL;
            this->right=NULL;
        }
        ~btNode() {
        }
        void setLeft(double m_value) {
            this->left=new btNode(m_value);
//          this->left->value=m_value;
        }
        void setRight(double m_value) {
            this->right=new btNode(m_value);
//          this->right->value=m_value;
        }
//      int printSelf();
        //三种非递归遍历（栈）
        int preGo1();
        int miGo1();
        int postGo1();
        //层序遍历 （队列）
        int levelGo();
        //三种递归遍历
        int preGo2();
        int miGo2();
        int postGo2();
};
```

```cpp

int btNode::preGo1() {
    stack<pair<btNode*,bool>> s;
    btNode *p;
    bool visited;
    //false 表示此点是第一次进栈
    s.push(make_pair(this,false));
    while(!s.empty()) {
        p=s.top().first;
        visited=s.top().second;
        s.pop();
        if(p==NULL){
            continue;
        }
        if(visited){
            cout<<"v: "<<p->value<<endl;
        }else{
            s.push(make_pair(p->right,false));
            s.push(make_pair(p->left,false));
            s.push(make_pair(p,true));
        }
    }
    return 0;
}

int btNode::miGo1() {
    stack<pair<btNode*,bool>> s;
    btNode *p;
    bool visited;
    //false 表示此点是第一次进栈
    s.push(make_pair(this,false));
    while(!s.empty()) {
        p=s.top().first;
        visited=s.top().second;
        s.pop();
        if(p==NULL){
            continue;
        }
        if(visited){
            cout<<"v: "<<p->value<<endl;
        }else{
            s.push(make_pair(p->right,false));
            s.push(make_pair(p,true));
            s.push(make_pair(p->left,false));
        }
    }
    return 0;
}

int btNode::postGo1() {
    stack<pair<btNode*,bool>> s;
    btNode *p;
    bool visited;
    //false 表示此点是第一次进栈
    s.push(make_pair(this,false));
    while(!s.empty()) {
        p=s.top().first;
        visited=s.top().second;
        s.pop();
        if(p==NULL){
            continue;
        }
        if(visited){
            cout<<"v: "<<p->value<<endl;
        }else{
            s.push(make_pair(p,true));
            s.push(make_pair(p->right,false));
            s.push(make_pair(p->left,false));
        }
    }
    return 0;
}

int btNode::levelGo(){
    queue<btNode*> q;
    q.push(this);
    btNode* p;
    while(!q.empty()){
        p=q.front();
        q.pop();
        cout<<"v: "<<p->value<<endl;
        if(p->left!=NULL){
            q.push(p->left);
        }
        if(p->right!=NULL){
            q.push(p->right);
        }
    }
```

```cpp
}

int btNode::preGo2() {
    cout<<"v: "<<this->value<<endl;
    if(this->left!=NULL) {
        this->left->preGo2();
    }
    if(this->right!=NULL) {
        this->right->preGo2();
    }
    return 0;
}

int btNode::miGo2() {
    if(this->left!=NULL) {
        this->left->miGo2();
    }
    cout<<"v: "<<this->value<<endl;
    if(this->right!=NULL) {
        this->right->miGo2();
    }
    return 0;
}

int btNode::postGo2() {
    if(this->left!=NULL) {
        this->left->postGo2();
    }
    if(this->right!=NULL) {
        this->right->postGo2();
    }
    cout<<"v: "<<this->value<<endl;
    return 0;
}

int main() {
    double a=0,b=1,c=2,d=3,e=4;
    //    0
    //  1   2
    // 3 4
    btNode *root=new btNode(a);
    root->setLeft(b);
    root->setRight(c);
    root->left->setLeft(d);
    root->left->setRight(e);
    cout<<"迭代式前序遍历："<<endl;
    root->preGo1();
    cout<<"迭代式中序遍历："<<endl;
    root->miGo1();
    cout<<"迭代式后序遍历："<<endl;
    root->postGo1();
    cout<<"---------------------"<<endl;
    cout<<"层序遍历："<<endl;
    root->levelGo();
    cout<<"---------------------"<<endl;
    cout<<"递归式前序遍历："<<endl;
    root->preGo2();
    cout<<"递归式中序遍历："<<endl;
    root->miGo2();
    cout<<"递归式后序遍历："<<endl;
    root->postGo2();
}
```