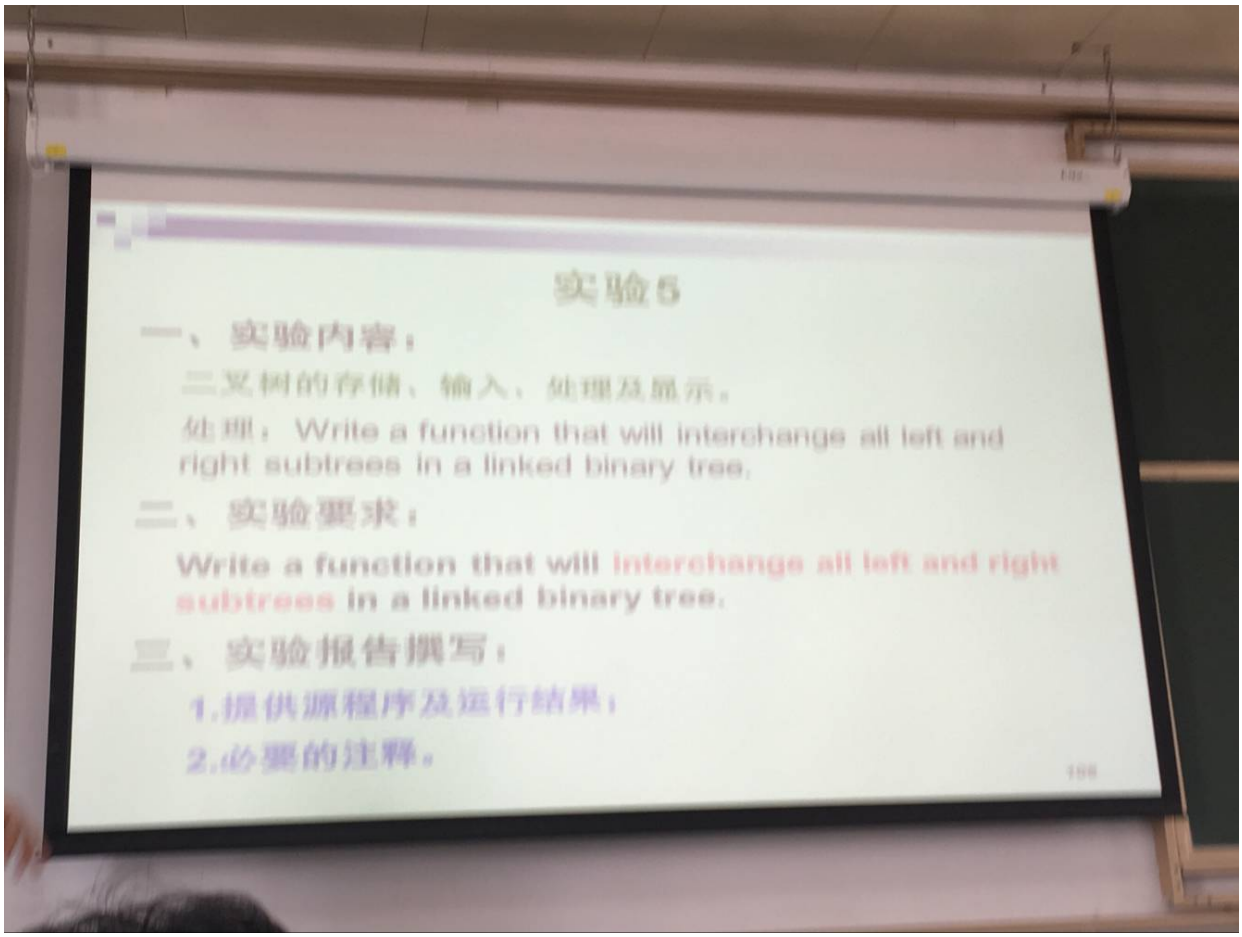
实验5 二叉查找树的左右子树交换



实验操作：

```
Input the choice you want to do

i           insert a node in linked binary tree
d           delete a node in linked binary tree
c           interchange all left and right subtrees
p           !!!output the binary tree to the file:binaryTree.dot
q           quit
```

基于是搜索树，插入删除操作均采用递归方式：

```
void insert_node(Node* &p,int e){
        if(p==nullptr){
            p=new Node(e);
            count++;
        }
        else if(p->entry<e){
            insert_node(p->right,e);
        }
        else
            insert_node(p->left,e);
        return;
    }
```

```
void remove_node(Node* p,int e){
        if(e<p->entry&&p->left){
            if(e==p->left->entry)
                remove_root(p->left);
            else
                remove_node(p->left,e);
        }
        else if(e>p->entry&&p->right){
            if(e==p->right->entry)
                remove_root(p->right);
            else
                remove_node(p->right,e);
        }
    }
```

核心算法:

```
void remove_root(Node* &r){
        Node* to_delete=r;
        if(!r->right)
            r=r->left;
        else if(!r->left)
            r=r->right;
        else{
            to_delete=r->left;
            Node* parent=r;
            while(to_delete->right){
                parent=to_delete;
                to_delete=to_delete->right;
            }
            r->entry=to_delete->entry;
            if(parent==r)
                r->left=to_delete->left;
            else
                parent->right=to_delete->left;
        }
        delete to_delete;
        count--;
        return;
    }
```

交换方法：

```
//递归
void interchange(Node* &p){
        // Node* pre=p;
        if(p){
            Node* tem=p->left;
            p->left=p->right;
            p->right=tem;
            interchange(p->left);
            interchange(p->right);
        }
    }


//非递归

void interchange(Node* &p){
//      stack<Node*> sta;
//      sta.push(p);
//      while(!sta.empty()){
//          p=sta.top();
//          sta.pop();
//          Node* tem=p->left;
//          p->left=p->right;
//          p->right=tem;
//          if(p->left)
//              sta.push(p->left);
//          if(p->right)
//              sta.push(p->right);
//          cout<<sta.size()<<endl;
//      }
//  }
```

附加功能：

利用c++文件操作将变换前和变换后的二叉树按dot语言格式写入before.dot和after.dot两个文件

```
digraph BST {
    node [fontname="Arial"];
    4 -> 3;
    3 -> 1;
    null0 [shape=point];
    1 -> null0;
    1 -> 2;
    null1 [shape=point];
    2 -> null1;
    null2 [shape=point];
    2 -> null2;
    null3 [shape=point];
    3 -> null3;
    4 -> 5;
    null4 [shape=point];
    5 -> null4;
    null5 [shape=point];
    5 -> null5;
}
//after.dot
```

利用shell script写生成png的命令到文件tree.sh

```
#!/bin/bash
dot ./code/before.dot -Tpng -o ./graph/before.png
dot ./code/after.dot -Tpng -o ./graph/after.png
open ./graph/before.png
open ./graph/after.png

//tree.sh
```

重要提示：

现在命令行编译然后运行程序

要将生成好的二叉树写入文件需**在字符选单中选择'p'**

然后**选择'c'将交换二叉树的左右子树**

最后要选择**'p'将交换好的子树保存到dot文件中**

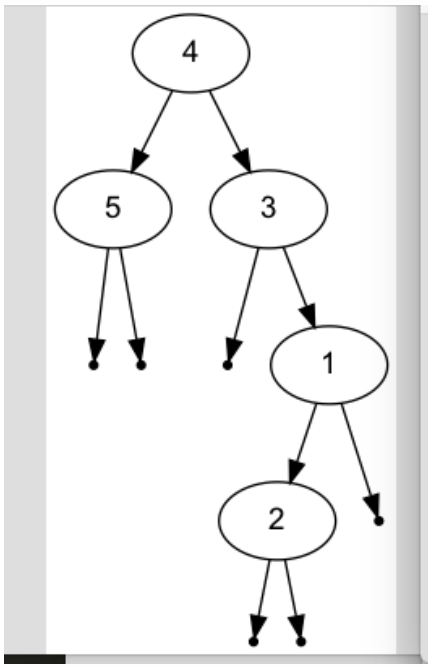
退出程序

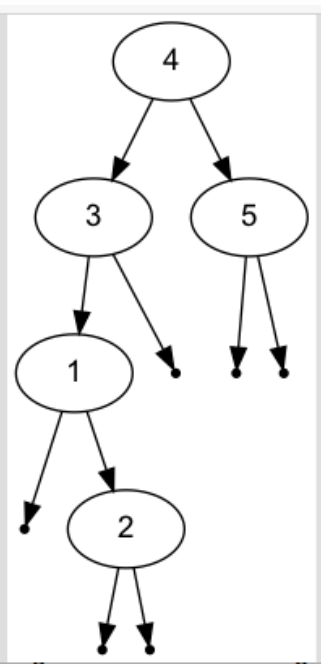
进入根目录执行./tree.sh

如有必要可以先加chmod +x tree.sh再执行脚本

即可打开两张png，显示出二叉树的树状图

测试样例：


4
5
3
1
2


4
3
5
1
2