```
/*
* 医院设施管理系统
* 对医院的各种设施的关系和数量进行统计和显示
* 程序有插入新的设施，统计设施数量，和显示设施等功能
*/
#include<iostream>
#include<queue>
#include<string>
#include<stack>
using namespace std;

//医院设施类
class HosPartNode{
private:
        int num;
        string name;
public:
        HosPartNode(string n="null", int number=1){
                name=n;
                num=number;
        }
        int getNum(){
                return num;
        }
        void setNum(int number){
                num=number;
        }
        string getName(){
                return name;
        }
        void setName(string n){
```

```
            name=n;
        }
        friend ostream &operator<< (ostream& out,HosPartNode &node){
            out << node.getName()<<":"<<node.getNum();
            return out;
        }
};

//孩子-兄弟表示法，二叉树节点类
class TreeNode{
private:
        HosPartNode m_value;
        TreeNode *FirstLeftChild;
        TreeNode *RightBrother;
public:
        TreeNode(HosPartNode value,TreeNode *FLChild=NULL,TreeNode *RBrother=NULL){
            m_value.setName(value.getName());
            m_value.setNum(value.getNum());
            FirstLeftChild=FLChild;
            RightBrother=RBrother;
        }
        HosPartNode getValue(){
            return m_value;
        }
        TreeNode *getChild(){
            return FirstLeftChild;
        }
        TreeNode *getBrother(){
            return RightBrother;
        }
        void setValue(HosPartNode value){
```

```
        m_value.setName(value.getName());
        m_value.setNum(value.getNum());
}
void setChild(TreeNode *pointer){
        FirstLeftChild=pointer;
}
void setBrother(TreeNode *pointer){
        RightBrother=pointer;
}
void InsertFirst(TreeNode *node){
        FirstLeftChild=node;
}
void InsertBrother(TreeNode *node){
        RightBrother=node;
}
void printNode(){
        stack<TreeNode*>s;
        TreeNode *pointer=FirstLeftChild;
        cout<<m_value<<endl;
        while(!s.empty() || pointer)
        {
                if(pointer)
                {
                        cout<<(pointer->getValue().getName())<<endl;
                        //pointer->getValue();
                        if(pointer->getBrother()!=NULL)
                                s.push(pointer->getBrother());
                        pointer=pointer->getChild();
                }
                else
                {
```

```
                        pointer=s.top();
                        s.pop();
                    }
                }
        }
};

// 设施树类
class Tree{
private:
        TreeNode *root;
public:
        Tree(){
                root=NULL;
        }
        Tree(TreeNode *node){
                root=node;
        }
        TreeNode *getRoot(){
                return root;
        }
        //获取当前节点的父节点
        TreeNode *Parent(TreeNode *current){
                queue<TreeNode *> aQueue;
                TreeNode *pointer=root,*parent=NULL;
                if(!current)                            //目标节点为空
                        return NULL;
                else if(pointer==current)               //目标节点是根节点
                        return NULL;
                else                                    //目标节点存在并且不是根节点
                {
```

```
                aQueue.push(pointer);
                while(!aQueue.empty())
                {
                        parent=aQueue.front();
                        pointer=parent->getChild();
                        while(pointer)
                        {
                                if(pointer==current)
                                {
                                        return parent;
                                }
                                aQueue.push(pointer);
                                pointer=pointer->getBrother();
                        }
                        aQueue.pop();
                }
        }
}
//查找值为current的节点
TreeNode *Find(string current){
        queue<TreeNode *> aQueue;
        TreeNode *pointer=root;
        if(current=="\0")
                return NULL;
        else{
                while(pointer){
                        if(pointer->getValue().getName()==current)
                                return pointer;
                        else{
                                aQueue.push(pointer);
                                pointer=pointer->getBrother();
```

```
                }
            }
            while(!aQueue.empty()){
                pointer=aQueue.front()->getChild();
                aQueue.pop();
                while(pointer){
                    if(pointer->getValue().getName()==current)
                        return pointer;
                    else{
                        aQueue.push(pointer);
                        pointer=pointer->getBrother();
                    }
                }
            }
            return NULL;
        }
}
//插入节点
void Insert(string parent,HosPartNode rootVlaue){
        HosPartNode h(parent);
        if(!root){
            root=new TreeNode(h);
            root->setChild(new TreeNode(rootVlaue));
        }
        else{
            TreeNode *temp=Find(parent);
            if(temp->getChild()){
                temp=temp->getChild();
                while(temp->getBrother())
                {
                    temp=temp->getBrother();
```

```
                        }
                        temp->setBrother(new TreeNode(rootVlaue));
                }
                else
                        temp->setChild(new TreeNode(rootVlaue));
        }
}
//先根遍历
void RootFirstTraverse(TreeNode *root){
        if(root!=NULL)
        {
                cout<<root->getValue().getName()<<endl;
                RootFirstTraverse(root->getChild());
                RootFirstTraverse(root->getBrother());
        }
}
//后根遍历
void RootLastTraverse(TreeNode *root){
        if(root!=NULL)
        {
                RootLastTraverse(root->getChild());
                cout<<root->getValue().getName()<<endl;
                RootLastTraverse(root->getBrother());
        }
}
//广度优先遍历
void WithTraverse(TreeNode *root){
        queue<TreeNode *>q;
        TreeNode *pointer=root;
        if(pointer)
                q.push(pointer);
```

```
        while(!q.empty())
        {
                pointer=q.front();
                cout<<pointer->getValue().getName()<<endl;
                q.pop();
                pointer=pointer->getChild();
                while(pointer)
                {
                        q.push(pointer);
                        pointer=pointer->getBrother();
                }
        }
}
//统计parent包含child的数量
int Count(string parent,string child){
        TreeNode *p=Find(parent),*c=Find(child);
        if(!p||!c)
                return 0;
        else{
                int Num=1;
                TreeNode *direParent=Parent(c);
                if(!direParent)
                        return 0;
                else{
                        Num*=c->getValue().getNum();
                        while(parent!=direParent->getValue().getName()){
                                Num*=direParent->getValue().getNum();
                                direParent=Parent(direParent);
                        }
                        return Num;
                }
```

```
                }
        }
};

int main(){
        Tree Hospital;
        // 建立医院节点，并添加到医院树上
        HosPartNode h0("医院");
        HosPartNode h1("楼层",10);
        HosPartNode h2("中央大厅",1);
        HosPartNode h3("配楼",4);
        HosPartNode h4("电视",1);
        HosPartNode h5("沙发",2);
        HosPartNode h6("长走廊",2);
        HosPartNode h7("走廊连接",1);
        HosPartNode h8("病房",21);
        HosPartNode h9("库房",5);
        HosPartNode h10("卫生间",1);
        HosPartNode h11("插座",4);
        HosPartNode h12("病床",2);
        HosPartNode h13("洗面盆",1);
        HosPartNode h14("坐便器",1);
        HosPartNode h15("插口",2);
        HosPartNode h16("面板",1);
        Hospital.Insert(h0.getName(),h1);
        Hospital.Insert(h1.getName(),h2);
        Hospital.Insert(h1.getName(),h3);
        Hospital.Insert(h2.getName(),h4);
        Hospital.Insert(h2.getName(),h5);
        Hospital.Insert(h3.getName(),h6);
        Hospital.Insert(h3.getName(),h7);
```

```
Hospital.Insert(h6.getName(),h8);
Hospital.Insert(h7.getName(),h9);
Hospital.Insert(h8.getName(),h10);
Hospital.Insert(h8.getName(),h11);
Hospital.Insert(h8.getName(),h12);
Hospital.Insert(h10.getName(),h13);
Hospital.Insert(h10.getName(),h14);
Hospital.Insert(h11.getName(),h15);
Hospital.Insert(h11.getName(),h16);
TreeNode *result=NULL;
int select;
string s1,s2;
while(1)
{
        cout<<"----------------------医院设施管理系统----------------------"<<endl;
        cout<<"1.插入新设施  2.统计数量  3.查找设施  4.浏览全部  0.退出系统"<<endl;
        cin>>select;
        switch(select)
        {
        case 0: return 0;
        case 1:{
                int num;
                cout<<"输入父单位名称"<<endl;
                cin>>s1;
                cout<<"输入设施名称"<<endl;
                cin>>s2;
                cout<<"输入设施数量"<<endl;
                cin>>num;
                HosPartNode newNode(s2,num);
                Hospital.Insert(s1,newNode);
                break;
```

```
                }
            case 2:{
                cout<<"输入父单位名称"<<endl;
                cin>>s1;
                cout<<"输入设施名称"<<endl;
                cin>>s2;
                cout<<s1<<"中包含"<<s2<<"的数量为"<<Hospital.Count(s1,s2)<<endl;
                break;
                }
            case 3:{
                cout<<"输入查找设施名称: "<<endl;
                cin>>s1;
                result=Hospital.Find(s1);
                cout<<"查找结果为: "<<endl;
                if(!result)
                        cout<<"未找到! "<<endl;
                else
                        result->printNode();
                break;
                }
            case 4:{
                cout<<"广度优先遍历: "<<endl;
                Hospital.WithTraverse(Hospital.getRoot());
                break;
                }
            }
        }

}
```