```
#include<iostream>
using namespace std;
class binarytreenode//二叉树结点
{
        int data;
public:
        binarytreenode * leftchild;
        binarytreenode * rightchild;
        binarytreenode * next;
        binarytreenode(){};
        binarytreenode(int & d)
        {
                data=d;
                leftchild=NULL;
                rightchild=NULL;
        }
        int& get_data(){        return data;    }
        void change_data(int& n){       data=n; }
        ~binarytreenode(){};
};
//队列
class queue
{
        int size;
        binarytreenode* front;
        binarytreenode* rear;
public:
        queue()
        {
                size=0;
                front=rear=NULL;
```

```
}
void push(binarytreenode* temp)//队尾插入
{
	if(front==NULL)
		front=rear=temp;
	else
	{
		rear->next=temp;
		rear=rear->next;
	}
	size++;
}
bool pop()//队首删除
{
	if(front==NULL)
	{
		cout<<"队列为空!Failed!"<<endl;
		return false;
	}
	front=front->next;
	size--;
	return true;
}
binarytreenode* get()//访问队首元素，但不删除
{
	if(front==NULL)
	{
		cout<<"队列为空，无队首元素!"<<endl;
		return NULL;
	}
	return front;
	}
	bool empty()//判断是否为空
	{
		if(size==0)
			return true;
		return false;
	}
	void clear()
	{
		binarytreenode * temp;
		while(front)
		{
			temp=front;
			front=front->next;
			delete temp;
		}
		rear=NULL;
		size=0;
	}
	~queue(){};
};
//栈
class stack
{
private:
	int size;
	int top;
	binarytreenode** ar;
public:
	stack(int size)
	{
        this->size=size;
        top=-1;
        ar=new binarytreenode*[size];
    }
    bool push(binarytreenode* item)//入栈
    {
        if(top==size-1)
        {
            cout<<"栈已满!"<<endl;
            return false;
        }
        else
        {
            ar[++top]=item;
            return true;
        }
    }
    bool pop()//出栈
    {
        if(top==-1)
        {
            cout<<"栈为空!"<<endl;
            return false;
        }
        else
        {
            top--;
            return true;
        }
    }
    binarytreenode* read()//读取栈顶元素
    {
        if(top==-1)
        {
            cout<<"栈为空!"<<endl;
            return NULL;
        }
        return ar[top];
    }
    bool empty()
    {
        if(top==-1)
            return true;
        return false;
    }
    ~stack(){};
};
//二叉树
class binarytree
{
    binarytreenode * root;
    static int times;
    int size;
public:
    binarytree()
    {   size=0;
        root=NULL;  }
    int get_size(){ return size;    }
    binarytreenode* get_root(){     return root;    }
    void creat()//创建二叉树
    {
        binarytreenode * prev;
        cout<<"输入数据:(以0结束)";
        int temp;
        cin>>temp;
        while(temp!=0)
        {
            if(root==NULL)
            {
                root=new binarytreenode(temp);
                size++;
                prev=root;
            }
            else
            {
                prev=root;
                size++;
                for(;;)
                {
                    if(temp < prev->get_data())
                    {
                        if(prev->leftchild==NULL)
                        {
                            prev->leftchild=new binarytreenode(temp);
                            break;
                        }
                        prev=prev->leftchild;
                    }
                    else
                    {
                        if(prev->rightchild==NULL)
                        {
                            prev->rightchild=new
binarytreenode(temp);
                            break;
                        }
                        prev=prev->rightchild;
                    }
                }
            }
            cin>>temp;
        }
    }
    void preorder()//前序遍历
    {
        binarytreenode * p = get_root();
        stack st(get_size());
        cout<<"前序遍历结果: ";
        if(p==NULL)
            cout<<"二叉树为空!";
        while(!st.empty()||p!=NULL)
            if(p!=NULL)
            {
                cout<<p->get_data()<<" ";
                if(p->rightchild!=NULL)
                    st.push(p->rightchild);
                p=p->leftchild;
            }
            else
            {
                p=st.read();
                st.pop();
            }
        cout<<endl;
}
void inorder()//中序遍历
{
        binarytreenode * p = get_root();
        stack st(get_size());
        cout<<"中序遍历结果: ";
        if(p==NULL)
                cout<<"二叉树为空!";
        while(!st.empty()||p!=NULL)
                if(p!=NULL)
                {
                        st.push(p);
                        p=p->leftchild;
                }
                else
                {
                        p=st.read();
                        cout<<p->get_data()<<" ";
                        p=p->rightchild;
                        st.pop();
                }
        cout<<endl;
}
void postorder()//后序遍历
{
        binarytreenode * p = get_root(),*prev=NULL;
        stack st(get_size());
        cout<<"后序遍历结果: ";
        if(p==NULL)
                cout<<"二叉树为空!";
        while(p!=NULL)
        {
            for(;p->leftchild!=NULL;p=p->leftchild)
                st.push(p);
            while(p!=NULL && (p->rightchild==NULL||p->rightchild==prev))//右子树
不存在或已经访问过,访问该结点
            {
                cout<<p->get_data()<<" ";
                prev=p;
                if(st.empty())
                    goto last;
                p=st.read();//读取栈顶元素
                st.pop();
            }
            st.push(p);
            p=p->rightchild;//访问右子树
        }
        last:
            cout<<endl;
    }
    void levelorder()//广度优先遍历
    {
        binarytreenode * p=get_root();
        queue que;
        cout<<"广度遍历结果: ";
        if(p!=NULL)
            que.push(p);
        else
            cout<<"二叉树为空!";
        while(!que.empty())
        {
```

```
		cout<<que.get()->get_data()<<" ";
		if(p->leftchild!=NULL)
			que.push(p->leftchild);
		if(p->rightchild!=NULL)
			que.push(p->rightchild);
		que.pop();
		p=que.get();
	}
	cout<<endl;
}
int degree1(binarytreenode*p)//统计度为1的结点
{
	if(p->leftchild!=NULL&&p->rightchild==NULL)
		return 1+degree1(p->leftchild);
	else if( p->leftchild==NULL&&p->rightchild!=NULL)
		return 1+degree1(p->rightchild);
	else if(p->leftchild==NULL&&p->rightchild==NULL)
		return 0;
	else if(p->leftchild!=NULL&&p->rightchild!=NULL)
		return 0+degree1(p->leftchild)+degree1(p->rightchild);
}
int degree2(binarytreenode*p)//统计度为2的结点
{
	if(p->leftchild!=NULL&&p->rightchild==NULL)
		return degree2(p->leftchild);
	else if( p->leftchild==NULL&&p->rightchild!=NULL)
		return degree2(p->rightchild);
	else if(p->leftchild==NULL&&p->rightchild==NULL)
		return 0;
	else if(p->leftchild!=NULL&&p->rightchild!=NULL)
		return 1+degree2(p->leftchild)+degree2(p->rightchild);
```

```
}
int degree0(binarytreenode*p)//统计度为0的结点
{
    if(p->leftchild!=NULL&&p->rightchild==NULL)
        return degree0(p->leftchild);
    else if( p->leftchild==NULL&&p->rightchild!=NULL)
        return degree0(p->rightchild);
    else if(p->leftchild==NULL&&p->rightchild==NULL)
        return 1;
    else if(p->leftchild!=NULL&&p->rightchild!=NULL)
        return degree0(p->leftchild)+degree0(p->rightchild);
}
int get_height(binarytreenode*p)//统计高度
{
    if(p->leftchild==NULL && p->rightchild==NULL)//叶子
        return 1;
    else if(p->leftchild!=NULL && p->rightchild==NULL)
        return 1+get_height(p->leftchild);
    else if(p->rightchild!=NULL && p->leftchild==NULL)
        return 1+get_height(p->rightchild);
    else if(p->leftchild!=NULL && p->rightchild!=NULL)
    {
        int i1=1+get_height(p->leftchild);
        int i2=1+get_height(p->rightchild);
        return (i1>i2)?i1:i2;
    }
}
void get_width(binarytreenode*p,int i,int wide[])//统计各层结点数
{
    wide[i++]++;
    if(p->leftchild!=NULL)
```

```
                get_width(p->leftchild,i,wide);
            if(p->rightchild!=NULL)
                get_width(p->rightchild,i,wide);
        }
        int get_max_width()//统计宽度
        {
            int *wide;
            int i=get_height(root);
            wide=new int[i];
            for(int j=0;j<i;j++)
                wide[j]=0;
            get_width(root,0,wide);
            int max=wide[0];
            for(int j=1;j<4;j++)
                if(wide[j] > max)
                    max=wide[j];
            return max;
        }
        int get_max(binarytreenode*p)//计算最大值
        {
            if(p->leftchild==NULL && p->rightchild==NULL)//叶子
                return p->get_data();
            else if(p->leftchild!=NULL && p->rightchild==NULL)
                return (p->get_data() > get_max(p->leftchild))?p->get_data():get_max(p->leftchild);
            else if(p->rightchild!=NULL && p->leftchild==NULL)
                return (p->get_data() > get_max(p->rightchild))?p->get_data():get_max(p->rightchild);
            else if(p->leftchild!=NULL && p->rightchild!=NULL)
            {
                int i1=(p->get_data() > get_max(p->leftchild))? p->get_data():
```

```
get_max(p->leftchild);
			int i2=(p->get_data() > get_max(p->rightchild))?p->get_data()
:get_max(p->rightchild);
			return (i1>i2)?i1:i2;
		}
	}
	void change_children(binarytreenode*p)//交换左右孩子
	{
		binarytreenode*temp;
		temp=p->leftchild;
		p->leftchild=p->rightchild;
		p->rightchild=temp;
		if(p->rightchild!=NULL)
			change_children(p->rightchild);
		if(p->leftchild!=NULL)
			change_children(p->leftchild);
	}
	int find_father(binarytreenode*num,binarytreenode*&fa)//寻找父节点
	{
		binarytreenode*p=root;
		int flag=0;
		while(p!=NULL)
		{
			if(p->get_data()>num->get_data())
			{
				fa=p;
				flag=1;
				p=p->leftchild;
			}
			else if(p->get_data()<num->get_data())
			{
```

```
                    fa=p;
                    flag=2;
                    p=p->rightchild;
                }
                else
                    break;
        }
        return flag;
}
void del_leaf(binarytreenode*p)//删除叶节点
{
        if(p->leftchild==NULL && p->rightchild==NULL)//叶子
        {
                binarytreenode*father=NULL;
                if(find_father(p,father)==1)
                        father->leftchild=NULL;
                else
                        father->rightchild=NULL;
                cout<<father->get_data()<<endl;
                cout<<p->get_data()<<"删除成功!"<<endl;
                delete p;
        }
        else if(p->rightchild!=NULL && p->leftchild==NULL)//右孩子
                del_leaf(p->rightchild);
        else if(p->rightchild==NULL && p->leftchild!=NULL)//左孩子
                del_leaf(p->leftchild);
        else if(p->rightchild!=NULL && p->leftchild!=NULL)//2孩子
        {
                        del_leaf(p->rightchild);
                        del_leaf(p->leftchild);
        }
```

```
		}
};
int binarytree::times=0;
int main()
{
	binarytree tree;
	tree.creat();
	tree.levelorder();
	cout<<"度为1的结点个数："<<tree.degree1(tree.get_root())<<endl;
	cout<<"度为2的结点个数："<<tree.degree2(tree.get_root())<<endl;
	cout<<"度为0的结点个数："<<tree.degree0(tree.get_root())<<endl;
	cout<<"二叉树高度："<<tree.get_height(tree.get_root())<<endl;
	cout<<"二叉树宽度："<<tree.get_max_width()<<endl;
	cout<<"最大值:"<<tree.get_max(tree.get_root())<<endl;
	cout<<"删除叶节点:"<<endl;
	tree.del_leaf(tree.get_root());
	tree.levelorder();
	cout<<"交换左右孩子．．．"<<endl;
	tree.change_children(tree.get_root());
	tree.levelorder();
	return 0;
}
```