**T16**

**T18**

**T20**

void visit(pair<bool,T> \*a,int pos)

{

if(a[pos].first==true)

cout<<a[pos].second<<" ";

}

void preOrder(pair<bool,T> \*a,int pos=1)

{

if(pos>=last) return;

visit(a,pos);

preOrder(a,pos\*2);

preOrder(a,pos\*2+1);

}

**T27**

int height(binaryTreeNode<T> \*t)

{//返回根为\*t的树的高度

if(t==NULL)

return 0;

else

{

int hl=height(t->left);

int hr=height(t->right);

if(hl>hr)

return ++hl;

else

return ++hr;

}

//复杂度O(n)，每一个元素都比较了一遍

}

**T28**

int size(binaryTreeNode<T> \*t)

{//返回根为\*t的树的节点数

if(t==NULL) return 0;

else

{

int ls=size(t->left); //左子树元素数

int rs=size(t->right); //右子树元素数

return ls+rs+1; //加上根节点

}

//复杂度O(n)

}

**T29**

int max\_width(binaryTreeNode<T> \*t)

{//求二叉树的每层最大节点数

arrayQueue< binaryTreeNode<T>\* > q;

binaryTreeNode<T> \*last=root; //last指向每层最右端的元素

int level=1; //层数

int width[20]; //保存每层节点数

while(t!=NULL)

{

if(t->left!=NULL)

q.push(t->left.element);

if(t->right!=NULL)

q.push(t->right.element);

//下一个t

if(q.empty()==true) break; //队列为空，结束循环

t=q.front();

q.pop();

if(t==last)

{//已经扫完当前层,此时队列里全是下一层的节点

width[++level]=q.size();

last=q.back(); //更新last

}

}

//从层数组中找出最大节点数

int max=0;

for(int i=1;i<level;i++)

{

if(width[i]>max)

max=width[i];

}

return max;

}

**T33**

void makeTree(char \*pre,char \*in,int n)

{//由前序和中序序列构造一颗二叉树，n是二叉树的节点总数

if(n<0) return;

char root=pre[0];

binaryTreeNode<T> \*b=new binaryTreeNode<T>(root);

char \*p; int mid=0; //根节点在in中的索引

for(p=in;p<in+n;p++)

{//在中序序列中找到根节点

if( (\*p)==root )

break;

mid++;

}

makeTree(pre+1,in,mid); //对左子树递归

makeTree(pre+1+mid,in+mid+1,n-mid-1); //对右子树递归

}

**T45**

bool compare(linkedBinaryTree<T> x)

{

if( x.preOrder()==\*this.preOrder() && x.inOrder()==\*this.inOrder() )

return true;

else return false;

}

**附：**

void linkedBinaryTree<T>:: visit(BinaryTreeNode<T>\*t,string str)

{

int size=str.size();

str.insert(size,t->element);

}

string linkedBinaryTree<T>:: preOrder()

{

static string str; //静态变量只创建一次

t=root;

if(t!=NULL)

{

visit(t,str);

preOrder(t->left);

preOrder(t->right);

}

if(\*this.treeSize==str.size())

{//当读完this树的所有元素时

string temp\_str=str;

str=""; //清空

return temp\_str;

}

else return str;

}

**T56**