#include<iostream>  
 #include<cstring>  
 #include<malloc.h> /\* malloc()等 \*/  
 #include<limits.h> /\* INT\_MAX等 \*/  
 #include<stdio.h> /\* EOF(=^Z或F6),NULL \*/  
 #include<stdlib.h> /\* atoi() \*/  
 #include<io.h> /\* eof() \*/  
 #include<math.h> /\* floor(),ceil(),abs() \*/  
 #include<process.h> /\* exit() \*/  
  
  
using namespace std;  
  
struct xtf\_data  //先定义好一个数据的结构  
{  
char data;  
xtf\_data \*l;  
xtf\_data \*r;  
};  
  
class xtf  
{  
private:  
int n;                   
int n1;                 
xtf\_data \*temp[1000];  
public:  
xtf\_data \*root;  
xtf()  //初始化  
{  
xtf\_data \*p;  
char t[1000];  
int i;  
int front=0,rear=1;           
cin.getline(t,1000);          
int n1=strlen(t);           
n=0;  
for(i=0;i<n1;i++)  
{  
if(t[i]!='#')  
{  
p=NULL;  
if(t[i]!=',')        
{  
n++;  
p=new xtf\_data;  
p->data=t[i];  
p->l=NULL;  
p->r=NULL;  
  
  
}  
front++;  
temp[front]=p;  
if(1 == front){root=p;}  
else  
{  
if((p!=NULL)&&(0==front%2))  
{  
temp[rear]->l=p;  
}  
if((p!=NULL)&&(1==front%2))  
{  
temp[rear]->r=p;  
}  
if(1==front%2)rear++;        
}  
}  
}  
}  
~xtf()                           
{  
int i;  
for(i=1;i<=n;i++)  
if(temp[i]!=NULL)  
delete temp[i];  
}  
void JS()                          
{  
int s;  
s=n;  
cout<<"(许腾飞NB）节点数为:"<<s<<endl;  
}  
  
void BL1(xtf\_data \*t)//先序遍历  
{  
if(NULL!=t)  
{  
cout<<t->data<<",";  
BL1(t->l);  
BL1(t->r);  
}  
}  
void BL2(xtf\_data \*t)//中序遍历  
{  
if(NULL!=t)  
{  
BL2(t->l);  
cout<<t->data<<",";  
BL2(t->r);  
}  
}  
void BL3(xtf\_data \*t)//后续遍历  
{  
if(NULL!=t)  
{  
BL3(t->l);  
BL3(t->r);  
cout<<t->data<<",";  
}  
}  
};  
  
int main()  
{  
xtf a;  
a.JS();  
a.BL1(a.root);  
cout<<endl;  
a.BL2(a.root);  
cout<<endl;  
a.BL3(a.root);  
cout<<endl;  
}

在这里，所有的二叉树都以数组的形式储存。在搜索路线中，若访问结点均是第一次经过结点时进行的，则是前序遍历;若访问结点均是在第二次(或第三次)经过结点时进行的，则是中序遍历(或后序遍历)。只要将搜索路线上所有在第一次、第二次和第三次经过的结点分别列表，即可分别得到该二叉树的前序序列、中序序列和后序序列。