天津大学 2013 年 901 试卷

一实做题（50 分）

1（10 分）请给出中缀表达式转换成后缀表达式的过程中栈的变化过程。（用一个栈来模拟表达式的转换过程）中缀表达式为：E=（（100-4）/3+3*（36-7））*2。

2（10 分）高度为 h 的满 K 叉树有如下特征：从 h 层上的节点度为 0，其余各层上的节点的度为 K。如果按从上到下，从左子树到右子树的次序对树中节点从 1 开始编号，则：

1） 各层的节点数是多少？

2） 编号为 i 的双亲节点（若存在）的编号是多少？

3） 编号为 i 的节点的第 m 个孩子节点（若存在）的编号是多少？

3（10 分）从空数开始，使用关键字：a，g，f，b，k，d，h，m，j，e，c，i，r，x 建立四阶 B-树。

4（10 分）设某项工程由下图所示的工序组成。若各工序以流水方式进行（即串进行）。其中：图中的紧前工序是指，没有工序 A 和 B，工序 B 必须在工序 A 完成之后才能开始。则工序 A 称为工序 B 的紧前工序。请完成题目：

工序	紧前工序
A	B,C
B	D
C	-----
D	-----
E	A,C,D

1） 画出流工程的 AOV 网络

2） 给出该工程的全部合理的工作流程

5（10 分）有一组关键{14，15，30，28，5，10}，给出构造出事小顶堆的过程图示，再根据初始小顶堆给出排序过程的图示。

二 算法设计题（25 分）

1（10 分）一个用邻接矩阵存储的有向图，请用栈来实现该图的深度优先搜索算法。

2（15 分）一个人从某年某月某日开始，三天打渔，两天晒网。写一个程序，计算他在以后的某年某月某日，是打渔，还是晒网。起始和终止日期从键盘输入。（假设计算从 2000 年 1 月开始到 2012 年 11 月 18 日结束）

三 程序填空（共 20 分，每空 2 分）

1 下面程序使用递归实现汉诺塔游戏

```
#include <iostream>
Using namespace std;
Void moveDisks(int n,char fromTower,char toTower,char auxTower)
{
    If(n==1)
        Cout<<"move disk"<<n<<"from"<<(  1  )<<"to "<<(  2  )<<endl;
    Else
    {
        moveDisks(  (3)  );
        Cout<<"move disk"<<n<<"from"<<(  4  )<<"to "<<(  5  )<<endl;
        moveDisks(  (6)  );
    }
}
```

```
Int main()
{
    Cout<<"Enter number of disks";
    Int n;
    Cin>>n;
    Cout<<"Enter number of disks"<<endl;
    moveDisks(n,'A','B','C');
    Return 0;
}
```

2 下面的程序通过继承关系实现对姓名的控制。类 class1 实现对名字访问的接口，class2 实现对名字的设置和输出。程序输出为：

```
Class2Name
Mike
```

程序中 定义的类并不完整，按要求完成下列操作，将类的定义补充完整。

（1） 类 Class1 中的定义接口函数 GetName（），为纯虚函数。请在空（7）填写适当语句。

（2） 函数 GetName2（）实现获得名字，但仅获得只读操作，请在空（8）填写适当语句。

（3） 实现 Class2（）的构造函数，请在空（9）填写适当语句。

（4） 完成构造函数，实现对名字的处理，请在空（10）填写适当语句。

```
#include <iostream>
Using namespace std;
Class Class1
{
Public :
（7）;
};
Class Class2:public Class1
{
Public :
    Void GetName()
    {
        Cout<<"Class2Name"<<endl;
    }
    （8）
    {
    Return   m_str;
    }
    （9）
    {
        Int I;
        For(int i=0;str!='\0';i++)
        {
            m_str[i]=str[i];
            （9）;
        }
```

```
    }
Private :
    Char m_str[32];
}
Int main()
{
    Class1    *p;
    Class2    obj(“Mike”);
    P=&obj;
    p->GetName();
    cout<<obj.GetName2()<<endl;
    return 0;
}
```

四 读下面程序，给出输出结果（共 30 分，每题 6 分）

1

```
#include <iostream>
Using namespace std;
Int main()
{
    Int x =18;
    Do
    {
        Switch(x%2)
        {
        Case 1:x- -;break;
        Case 0:x=x/2;break;
        }
        x- -;
        cout<<x<<endl;
    }While(x>0);
    Return 0;
}
```

2

```
#include <iostream>
Using namespace std;
Void    f( int j);
Iint main()
{
    For(int i=1;i<=4;i++)
        F( I );
    Return 0;
}
Void    f( int j )
```

```
{
    Static int a=2;
    Int b=1;
    b++;
    cout<<a<<”+”<<b<<”+”<<j<<”=”<<a+b+j<<endl;
    a+=10;
}
3
#include    <iostream>
Using namespace std;
Template    < typename    T >    class pushOnFull
{
    T    _value;
Public:
    pushOnFull( T    i )
    {
        _value = i;
    }
    T    value()
    {
        Return _value;
    }
    Void print()
    {
        Cout<<”Stack is fall, ”<<_value<<”is not pushed”<<endl;
    }
};
Template < typename    T>    class    popOnEmpty
{
Public:
    Void print()
    {
        Cout<<”stack    is    empty    ,con’t    pop”<<endl;
    }
};
Template    < typename    T    > class    Stack
{
    Int top;
    T    *elements;
    Int    maxSize;
Public:
    Stack( int =20);
    ~Stack()
    {
```

```
            Delete[]    elements;
        }
        Void Push(const T &    data);
        T    Pop();

        T    GetEle(int i)
        {
            Return element[i];
        }
        Void makeEmpty()
        {
            Top=-1;
        }
        Bool IsEmpty()const
        {
            Return top==-1;
        }
        Bool IsFull()const
        {
            Return top==maxSize-1;
        }
        Void printStack();

};
Template <typename    T>Stack<T>::Stack(int maxs)
{
        maxSize=maxs;
        top=-1;
        elements= new T[maxSize];
}
Template <typename    T> void Stack(T)::printStack()
{
        For(int i=0; i<top ; i++)
            Cout<<elements[i]<<"    ";
        Cout<<endl;
}
Template    <typename    T>    void    Stack<T>::Push( const T & data)
{
        If(IsFull())
            Throw pushOnFull<T>(data);
        Elements[++top]=data;
}
Template <typename    T> T Stack<T>::print()
{
```

```
        If(IsEmpty())
            Throw popOnEmpty<T>();
        Return elements[top--];
}
Int main()
{
        Int a[9]={1,8,7,6,5,4,3,2,1} , b[9]={0},i ;
        Stack <int> istack(8)'
        Try
        {
            For(int i=0; i<9 ; i++)
                Istack.Push(a[i]);
            Istack.printStack();
        }
        Catsh(pushOnFull<int> & eobj)
        {
            Eobj.print();
        }
        Try
        {
            For(int i=0; i<9 ; i++)
                b[i]=Istack.Pop(a[i]);
            Istack.printStack();
        }
        Catsh(PopOnEmptyl<int> & eobj)
        {
            Eobj.print();
        }
        Cout<<" Pop order is:";
        For(ini =0;i<9;i++)
            Cout<<b[i]<<" ";
        Cout<<endl;
        Return 0;
}
```

4 无处可查

5

```
#include    <iostream>
Using namespace std;

Class    AA
{
Public:
        AA()
        {
```

```
        Cout<<" Constructor   of   AA"<<endl;
    };
    Virtual   void   funs()
    {
        Count<<"AA::funs()   called"<<endl;
    }
};
Class   BB:public AA
{
Public :
    BB()
    {
        Count<<"construct   of   BB"<<endl;
    }
    Void   fun()
    {
        Count<<"BB::fun()   called"<<endl;
    }
}

Void Cal2(BB   a)
{
    a.fun();
}
Void Cal2(AA   a)
{
    a.fun();
}
Int   main()
{
    BB b;;
    Cal1(b);
    Cal2(b);
    Return 0;
}
```

五 按照题目要求，采用 C++语言编写程序（共 25 分）

1 交叉奇偶校验（本题 10 分）

1	0	1	0
0	0	0	0
1	1	1	1
0	1	0	1

在检验中有一种检验方法是交叉奇偶检验，检验规则是：行和列的 1 的个数为偶数时，表示正确。下面举例说明:所有行中 1 的个数为 2，0，4，2；所有列中 1 的个数为 2，2，2，2。任务是写一个程序，给定规模的矩阵（n*n,n<1000）进行交叉校验。若校验正确，则输出“OK”，若校验不正确且只有一位错误，则输出“change bif（2，3）”，（2，3）表示哪一行哪一列出错，若校验不正确且多个错误，则输出“error”。（个别单词文档编辑人臆测）。

2 二叉树遍历（15 分）

给定一个二叉树的先序和中序遍历结果，求出其后序遍历结果。下面举例说明：

根据中的二叉树，给定的先序为 DBACEGF，中序为 ABCDEFG，求出其后序遍历结果 ACBFGED。

请编写程序读入先序和后序遍历结果，求出后序遍历结果，并输出字母序列。（数据规模不定，类型为字符型）。

以下面的例题为例进行讲解：

已知一棵二叉树的先序遍历序列和中序遍历序列分别是 ABDCEF、BDAECF，求二叉树及后序遍历序列。

分析：先序遍历序列的第一个字符为根结点。对于中序遍历，根结点在中序遍历序列的中间，左边部分是根结点的左子树的中序遍历序列，右边部分是根结点的右子树的中序遍历序列。

先序：ABDCEF --> A BD CEF

中序：BDAECF --> BD A ECF

得出结论：A 是树根，A 有左子树和右子树，左子树有 BD 结点，右子树有 CEF 结点。

先序：BD --> B D

中序：BD --> B D

得出结论：B 是左子树的根结点，B 无左子树，有右子树(只有 D 结点)。

先序：CEF --> C E F

中序：ECF --> E C F

得出结论：C 是右子树的根结点，C 有左子树(只有 E 结点)，有右子树(只有 F 结点)。

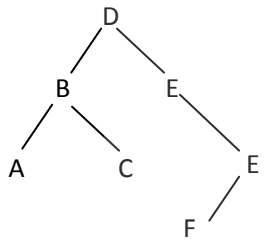
还原二叉树为：

A

B C

D E F

后序遍历序列：DBEFCA



这种题一般有二种形式，共同点是都已知中序序列。如果没有中序序列，是无法唯一确定一棵树的。

<1>已知二叉树的前序序列和中序序列，求解树。

1、确定树的根节点。树根是当前树中所有元素在前序遍历中最先出现的元素。

2、求解树的子树。找出根节点在中序遍历中的位置，根左边的所有元素就是左子树，根右边的所有元素就是右子树。若根节点左边或右边为空，则该方向子树为空；若根节点边和右边都为空，则根节点已经为叶子节点。

3、递归求解树。将左子树和右子树分别看成一棵二叉树，重复 1、2、3 步，直到所有的节点完成定位。

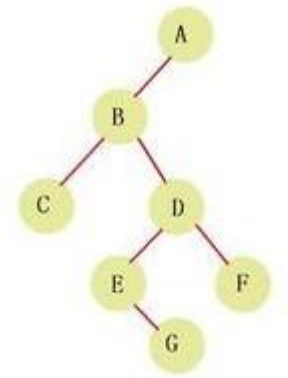
<2>、已知二叉树的后序序列和中序序列，求解树。

1、确定树的根。树根是当前树中所有元素在后序遍历中最后出现的元素。

2、求解树的子树。找出根节点在中序遍历中的位置，根左边的所有元素就是左子树，根右边的所有元素就是右子树。若根节点左边或右边为空，则该方向子树为空；若根节点边和右边都为空，则根节点已经为叶子节点。

3、递归求解树。将左子树和右子树分别看成一棵二叉树，重复 1、2、3 步，直到所有的节点完成定位。

测试用例：



<1>先序 中序 求 后序

输入：

先序序列：ABCDEGF

中序序列：CBEGDFA

输出后序：CGEFDBA

代码：

[cpp] view plaincopy

```
1.  /*
2.      PreIndex: 前序序列字符串中子树的第一个节点在 PreArray[]中的下标
3.      InIndex:  中序序列字符串中子树的第一个节点在 InArray[]中的下标
4.      subTreeLen: 子树的字符串序列的长度
5.      PreArray:  先序序列数组
```

```
    InArray: 中序序列数组
*/
void PreInCreateTree(BiTree &T,int PreIndex,int InIndex,int subTreeLen){
    //subTreeLen < 0 子树为空
    if(subTreeLen <= 0){
        T = NULL;
        return;
    }
    else{
        T = (BiTree)malloc(sizeof(BiTNode));
        //创建根节点
        T->data = PreArray[PreIndex];
        //找到该节点在中序序列中的位置
        int index = strchr(InArray,PreArray[PreIndex]) - InArray;
        //左子树结点个数
        int LenF = index - InIndex;
        //创建左子树
        PreInCreateTree(T->lchild,PreIndex + 1,InIndex,LenF);
        //右子树结点个数(总结点 - 根节点 - 左子树结点)
        int LenR = subTreeLen - 1 - LenF;
        //创建右子树
        PreInCreateTree(T->rchild,PreIndex + LenF + 1,index + 1,LenR);
    }
}
```

主函数调用：

[cpp] view plaincopy

```
BiTree T;
    PreInCreateTree(T,0,0,strlen(InArray));
    PostOrder(T);
```

另一种算法：

[cpp] view plaincopy

```
/*
    PreS        先序序列的第一个元素下标
    PreE        先序序列的最后一个元素下标
    InS         中序序列的第一个元素下标
    InE         先序序列的最后一个元素下标
    PreArray    先序序列数组
    InArray     中序序列数组
```

```
*/
void PreInCreateTree(BiTree &T,int PreS ,int PreE ,int InS ,int InE){
    int RootIndex;
    //先序第一个字符
    T = (BiTree)malloc(sizeof(BiTNode));
    T->data = PreArray[PreS];
    //寻找该结点在中序序列中的位置
    for(int i = InS;i <= InE;i++){
        if(T->data == InArray[i]){
            RootIndex = i;
            break;
        }
    }
    //根结点的左子树不为空
    if(RootIndex != InS){
        //以根节点的左结点为根建树
        PreInCreateTree(T->lchild,PreS+1,(RootIndex-InS)+PreS,InS,RootIndex-1);
    }
    else{
        T->lchild = NULL;
    }
    //根结点的右子树不为空
    if(RootIndex != InE){
        //以根节点的右结点为根建树
        PreInCreateTree(T->rchild,PreS+1+(RootIndex-InS),PreE,RootIndex+1,InE);
    }
    else{
        T->rchild = NULL;
    }
}
```

主函数调用：

[cpp] view plaincopy

```
PreInCreateTree(T,0,strlen(PreArray)-1,0,strlen(InArray)-1);
```

具体讲解请看：点击打开链接

<2>中序 后序 求先序

输入：

中序序列：CBEGDFA

后序序列：CGEFDBA

输出先序：ABCDEGF

代码：

[cpp] view plaincopy

```cpp
/*
    PostIndex: 后序序列字符串中子树的最后一个节点在 PreArray[]中的下标
    InIndex:  中序序列字符串中子树的第一个节点在 InArray[]中的下标
    subTreeLen: 子树的字符串序列的长度
    PostArray:  后序序列数组
    InArray：中序序列数组
*/
void PostInCreateTree(BiTree &T,int PostIndex,int InIndex,int subTreeLen){
    //subTreeLen < 0 子树为空
    if(subTreeLen <= 0){
        T = NULL;
        return;
    }
    else{
        T = (BiTree)malloc(sizeof(BiTNode));
        //创建根节点
        T->data = PostArray[PostIndex];
        //找到该节点在中序序列中的位置
        int index = strchr(InArray,PostArray[PostIndex]) - InArray;
        //左子树结点个数
        int LenF = index - InIndex;
        //创建左子树
        PostInCreateTree(T->lchild,PostIndex - (subTreeLen - 1 - LenF) - 1,InIndex,LenF);
        //右子树结点个数(总结点 - 根节点 - 左子树结点)
        int LenR = subTreeLen - 1 - LenF;
        //创建右子树
        PostInCreateTree(T->rchild,PostIndex-1,index + 1,LenR);
    }
}
```

主函数调用：

[cpp] view plaincopy

```
BiTree T2;
    PostInCreateTree(T2,strlen(PostArray) - 1,0,strlen(InArray));
    PreOrder(T2);
```

完整代码：

[cpp] view plaincopy

```cpp
#include<iostream>
#include<string>
using namespace std;

//二叉树结点
typedef struct BiTNode{
    //数据
    char data;
    //左右孩子指针
    struct BiTNode *lchild,*rchild;
}BiTNode,*BiTree;

//先序序列
char PreArray[101] = "ABCDEGF";
//中序序列
char InArray[101] = "CBEGDFA";
//后序序列
char PostArray[101] = "CGEFDBA";
/*
    PreIndex: 前序序列字符串中子树的第一个节点在 PreArray[]中的下标
    InIndex:  中序序列字符串中子树的第一个节点在 InArray[]中的下标
    subTreeLen: 子树的字符串序列的长度
    PreArray:  先序序列数组
    InArray: 中序序列数组
*/
void PreInCreateTree(BiTree &T,int PreIndex,int InIndex,int subTreeLen){
    //subTreeLen < 0 子树为空
    if(subTreeLen <= 0){
        T = NULL;
        return;
    }
    else{
```

```
33.         T = (BiTree)malloc(sizeof(BiTNode));
34.         //创建根节点
35.         T->data = PreArray[PreIndex];
36.         //找到该节点在中序序列中的位置
37.         int index = strchr(InArray,PreArray[PreIndex]) - InArray;
38.         //左子树结点个数
39.         int LenF = index - InIndex;
40.         //创建左子树
41.         PreInCreateTree(T->lchild,PreIndex + 1,InIndex,LenF);
42.         //右子树结点个数(总结点 - 根节点 - 左子树结点)
43.         int LenR = subTreeLen - 1 - LenF;
44.         //创建右子树
45.         PreInCreateTree(T->rchild,PreIndex + LenF + 1,index + 1,LenR);
46.     }
47. }
48. /*
49.     PostIndex: 后序序列字符串中子树的最后一个节点在 PreArray[]中的下标
50.     InIndex:  中序序列字符串中子树的第一个节点在 InArray[]中的下标
51.     subTreeLen: 子树的字符串序列的长度
52.     PostArray:  后序序列数组
53.     InArray: 中序序列数组
54. */
55. void PostInCreateTree(BiTree &T,int PostIndex,int InIndex,int subTreeLen){
56.     //subTreeLen < 0 子树为空
57.     if(subTreeLen <= 0){
58.         T = NULL;
59.         return;
60.     }
61.     else{
62.         T = (BiTree)malloc(sizeof(BiTNode));
63.         //创建根节点
64.         T->data = PostArray[PostIndex];
65.         //找到该节点在中序序列中的位置
66.         int index = strchr(InArray,PostArray[PostIndex]) - InArray;
67.         //左子树结点个数
68.         int LenF = index - InIndex;
69.         //创建左子树
70.         PostInCreateTree(T->lchild,PostIndex - (subTreeLen - 1 - LenF) - 1,I
    nIndex,LenF);
71.         //右子树结点个数(总结点 - 根节点 - 左子树结点)
72.         int LenR = subTreeLen - 1 - LenF;
73.         //创建右子树
74.         PostInCreateTree(T->rchild,PostIndex-1,index + 1,LenR);
75.     }
```

```
}
//先序遍历
void PreOrder(BiTree T){
    if(T != NULL){
        //访问根节点
        printf("%c ",T->data);
        //访问左子结点
        PreOrder(T->lchild);
        //访问右子结点
        PreOrder(T->rchild);
    }
}
//后序遍历
void PostOrder(BiTree T){
    if(T != NULL){
        //访问左子结点
        PostOrder(T->lchild);
        //访问右子结点
        PostOrder(T->rchild);
        //访问根节点
        printf("%c ",T->data);
    }
}
int main()
{
    BiTree T;
    PreInCreateTree(T,0,0,strlen(InArray));
    PostOrder(T);
    printf("\n");
    BiTree T2;
    PostInCreateTree(T2,strlen(PostArray) - 1,0,strlen(InArray));
    PreOrder(T2);
    return 0;
}
```