目录

[实例1.1 最大子列和问题（20 分） 6](#_Toc508544885)

[输入格式: 6](#_Toc508544886)

[输出格式: 6](#_Toc508544887)

[输入样例: 6](#_Toc508544888)

[输出样例: 6](#_Toc508544889)

[习题1.8 二分查找（20 分） 6](#_Toc508544890)

[函数接口定义： 6](#_Toc508544891)

[裁判测试程序样例： 7](#_Toc508544892)

[输入样例1： 8](#_Toc508544893)

[输出样例1： 8](#_Toc508544894)

[输入样例2： 8](#_Toc508544895)

[输出样例2： 8](#_Toc508544896)

[习题1.9 有序数组的插入（20 分） 8](#_Toc508544897)

[函数接口定义： 8](#_Toc508544898)

[裁判测试程序样例： 9](#_Toc508544899)

[输入样例1： 9](#_Toc508544900)

[输出样例1： 10](#_Toc508544901)

[输入样例2： 10](#_Toc508544902)

[输出样例2： 10](#_Toc508544903)

[习题2.1 简单计算器（20 分） 10](#_Toc508544904)

[输入格式: 10](#_Toc508544905)

[输出格式: 10](#_Toc508544906)

[输入样例: 10](#_Toc508544907)

[输出样例: 11](#_Toc508544908)

[习题2.2 数组循环左移（20 分） 11](#_Toc508544909)

[输入格式: 11](#_Toc508544910)

[输出格式: 11](#_Toc508544911)

[输入样例： 11](#_Toc508544912)

[输出样例： 11](#_Toc508544913)

[习题2.3 数列求和-加强版（20 分） 11](#_Toc508544914)

[输入格式： 11](#_Toc508544915)

[输出格式： 12](#_Toc508544916)

[输入样例： 12](#_Toc508544917)

[输出样例： 12](#_Toc508544918)

[习题2.4 递增的整数序列链表的插入（15 分） 12](#_Toc508544919)

[函数接口定义： 12](#_Toc508544920)

[裁判测试程序样例： 12](#_Toc508544921)

[输入样例： 13](#_Toc508544922)

[输出样例： 13](#_Toc508544923)

[习题2.5 两个有序链表序列的合并（15 分） 13](#_Toc508544924)

[函数接口定义： 13](#_Toc508544925)

[裁判测试程序样例： 14](#_Toc508544926)

[输入样例： 15](#_Toc508544927)

[输出样例： 15](#_Toc508544928)

[习题2.6 递归求简单交错幂级数的部分和（15 分） 15](#_Toc508544929)

[函数接口定义： 15](#_Toc508544930)

[裁判测试程序样例： 15](#_Toc508544931)

[输入样例： 16](#_Toc508544932)

[输出样例： 16](#_Toc508544933)

[习题2.7 弹球距离（15 分） 16](#_Toc508544934)

[函数接口定义： 16](#_Toc508544935)

[裁判测试程序样例： 16](#_Toc508544936)

[输入样例： 17](#_Toc508544937)

[输出样例： 17](#_Toc508544938)

[习题2.8 输出全排列（20 分） 17](#_Toc508544939)

[输入格式: 17](#_Toc508544940)

[输出格式: 17](#_Toc508544941)

[输入样例： 17](#_Toc508544942)

[输出样例： 18](#_Toc508544943)

[习题3.4 最长连续递增子序列（20 分） 18](#_Toc508544944)

[输入格式: 18](#_Toc508544945)

[输出格式: 18](#_Toc508544946)

[输入样例： 18](#_Toc508544947)

[输出样例： 18](#_Toc508544948)

[习题3.5 求链表的倒数第m个元素（20 分） 19](#_Toc508544949)

[函数接口定义： 19](#_Toc508544950)

[裁判测试程序样例： 19](#_Toc508544951)

[输入样例： 20](#_Toc508544952)

[输出样例： 20](#_Toc508544953)

[习题3.6 一元多项式的乘法与加法运算（20 分） 20](#_Toc508544954)

[输入格式: 20](#_Toc508544955)

[输出格式: 20](#_Toc508544956)

[输入样例: 20](#_Toc508544957)

[输出样例: 21](#_Toc508544958)

[习题3.8 符号配对（20 分） 21](#_Toc508544959)

[输入格式: 21](#_Toc508544960)

[输出格式: 21](#_Toc508544961)

[输入样例1： 21](#_Toc508544962)

[输出样例1： 21](#_Toc508544963)

[输入样例2： 22](#_Toc508544964)

[输出样例2： 22](#_Toc508544965)

[输入样例3： 22](#_Toc508544966)

[输出样例3： 22](#_Toc508544967)

[习题3.9 堆栈操作合法性（20 分） 22](#_Toc508544968)

[输入格式: 22](#_Toc508544969)

[输出格式: 23](#_Toc508544970)

[输入样例： 23](#_Toc508544971)

[输出样例： 23](#_Toc508544972)

[习题3.10 汉诺塔的非递归实现（25 分） 23](#_Toc508544973)

[输入格式: 23](#_Toc508544974)

[输出格式: 23](#_Toc508544975)

[输入样例: 23](#_Toc508544976)

[输出样例: 24](#_Toc508544977)

[习题3.11 表达式转换（25 分） 24](#_Toc508544978)

[输入格式: 24](#_Toc508544979)

[输出格式: 24](#_Toc508544980)

[输入样例: 24](#_Toc508544981)

[输出样例: 24](#_Toc508544982)

[习题3.12 另类循环队列（20 分） 25](#_Toc508544983)

[函数接口定义： 25](#_Toc508544984)

[裁判测试程序样例： 25](#_Toc508544985)

[输入样例： 27](#_Toc508544986)

[输出样例： 27](#_Toc508544987)

[习题3.13 双端队列（25 分） 27](#_Toc508544988)

[函数接口定义： 27](#_Toc508544989)

[裁判测试程序样例： 28](#_Toc508544990)

[输入样例： 30](#_Toc508544991)

[输出样例： 30](#_Toc508544992)

[习题3.14 另类堆栈（15 分） 30](#_Toc508544993)

[函数接口定义： 30](#_Toc508544994)

[裁判测试程序样例： 31](#_Toc508544995)

[输入样例： 32](#_Toc508544996)

[输出样例： 33](#_Toc508544997)

[练习4.1 根据后序和中序遍历输出先序遍历（25 分） 33](#_Toc508544998)

[输入格式: 33](#_Toc508544999)

[输出格式: 33](#_Toc508545000)

[输入样例: 33](#_Toc508545001)

[输出样例: 33](#_Toc508545002)

[练习4.2 平衡二叉树的根（25 分） 34](#_Toc508545003)

[输入格式: 34](#_Toc508545004)

[输出格式: 34](#_Toc508545005)

[输入样例1: 34](#_Toc508545006)

[输出样例1: 34](#_Toc508545007)

[输入样例2: 34](#_Toc508545008)

[输出样例2: 34](#_Toc508545009)

[习题4.3 是否二叉搜索树（25 分） 34](#_Toc508545010)

[函数接口定义： 34](#_Toc508545011)

[裁判测试程序样例： 35](#_Toc508545012)

[输入样例1：如下图 36](#_Toc508545013)

[输出样例1： 36](#_Toc508545014)

[输入样例2：如下图 36](#_Toc508545015)

[输出样例2： 36](#_Toc508545016)

[练习4.3 堆中的路径（25 分） 37](#_Toc508545017)

[输入格式: 37](#_Toc508545018)

[输出格式: 37](#_Toc508545019)

[输入样例: 37](#_Toc508545020)

[输出样例: 37](#_Toc508545021)

[习题4.5 顺序存储的二叉树的最近的公共祖先问题（25 分） 37](#_Toc508545022)

[输入格式: 37](#_Toc508545023)

[输出格式: 38](#_Toc508545024)

[输入样例1： 38](#_Toc508545025)

[输出样例1： 38](#_Toc508545026)

[输入样例2： 38](#_Toc508545027)

[输出样例2： 38](#_Toc508545028)

[习题5.10 线性探测法的查找函数（20 分） 38](#_Toc508545029)

[函数接口定义： 38](#_Toc508545030)

[裁判测试程序样例： 39](#_Toc508545031)

[输入样例1：（注：-1表示该位置为空。下同。） 40](#_Toc508545032)

[输出样例1： 40](#_Toc508545033)

[输入样例2： 41](#_Toc508545034)

[输出样例2： 41](#_Toc508545035)

[输入样例3： 41](#_Toc508545036)

[输出样例3： 41](#_Toc508545037)

[习题5.11 分离链接法的删除操作函数（20 分） 41](#_Toc508545038)

[函数接口定义： 41](#_Toc508545039)

[裁判测试程序样例： 42](#_Toc508545040)

[输入样例1：散列表如下图 43](#_Toc508545041)

[输出样例1： 43](#_Toc508545042)

[输入样例2：散列表如样例1图 43](#_Toc508545043)

[输出样例2： 43](#_Toc508545044)

[习题5.13 词频统计（30 分） 44](#_Toc508545045)

[输入格式: 44](#_Toc508545046)

[输出格式: 44](#_Toc508545047)

[输入样例： 44](#_Toc508545048)

[输出样例：（注意：虽然单词the也出现了4次，但因为我们只要输出前10%（即23个单词中的前2个）单词，而按照字母序，the排第3位，所以不输出。） 44](#_Toc508545049)

[感谢武汉理工大学的郭小兵老师修正测试数据！ 45](#_Toc508545050)

[练习6.1 邻接矩阵存储图的深度优先遍历（20 分） 45](#_Toc508545051)

[函数接口定义： 45](#_Toc508545052)

[裁判测试程序样例： 45](#_Toc508545053)

[输入样例：给定图如下 46](#_Toc508545054)

[输出样例： 46](#_Toc508545055)

[练习6.2 邻接表存储图的广度优先遍历（20 分） 47](#_Toc508545056)

[函数接口定义： 47](#_Toc508545057)

[裁判测试程序样例： 47](#_Toc508545058)

[输入样例：给定图如下 49](#_Toc508545059)

[输出样例： 49](#_Toc508545060)

[实例6.1 六度空间（30 分） 49](#_Toc508545061)

[输入格式: 50](#_Toc508545062)

[输出格式: 50](#_Toc508545063)

[输入样例: 50](#_Toc508545064)

[输出样例: 51](#_Toc508545065)

[练习7.1 排序（25 分） 51](#_Toc508545066)

[输入格式: 51](#_Toc508545067)

[输出格式: 51](#_Toc508545068)

[输入样例: 51](#_Toc508545069)

[输出样例: 52](#_Toc508545070)

[习题8.1 银行排队问题之单队列多窗口服务（25 分） 52](#_Toc508545071)

[输入格式: 52](#_Toc508545072)

[输出格式: 52](#_Toc508545073)

[输入样例： 52](#_Toc508545074)

[输出样例： 53](#_Toc508545075)

[感谢浙江财经大学王瑞洲、周甄陶同学修正测试数据！ 53](#_Toc508545076)

[习题8.2 银行排队问题之单队列多窗口加VIP服务（30 分） 53](#_Toc508545077)

[输入格式: 53](#_Toc508545078)

[输出格式: 53](#_Toc508545079)

[输入样例： 54](#_Toc508545080)

[输出样例： 54](#_Toc508545081)

[习题8.3 银行排队问题之单窗口“夹塞”版（30 分） 54](#_Toc508545082)

[输入格式: 54](#_Toc508545083)

[输出格式: 55](#_Toc508545084)

[输入样例: 55](#_Toc508545085)

[输出样例: 55](#_Toc508545086)

[习题8.4 畅通工程之最低成本建设问题（30 分） 56](#_Toc508545087)

[输入格式: 56](#_Toc508545088)

[输出格式: 56](#_Toc508545089)

[输入样例1: 56](#_Toc508545090)

[输出样例1: 56](#_Toc508545091)

[输入样例2: 57](#_Toc508545092)

[输出样例2: 57](#_Toc508545093)

[习题8.5 畅通工程之局部最小花费问题（35 分） 57](#_Toc508545094)

[输入格式: 57](#_Toc508545095)

[输出格式: 57](#_Toc508545096)

[输入样例: 57](#_Toc508545097)

[输出样例: 58](#_Toc508545098)

# 实例1.1 最大子列和问题（20 分）

给定*K*个整数组成的序列{ *N*​1​​, *N*​2​​, ..., *N*​*K*​​ }，“连续子列”被定义为{ *N*​*i*​​, *N*​*i*+1​​, ..., *N*​*j*​​ }，其中 1≤*i*≤*j*≤*K*。“最大子列和”则被定义为所有连续子列元素的和中最大者。例如给定序列{ -2, 11, -4, 13, -5, -2 }，其连续子列{ 11, -4, 13 }有最大的和20。现要求你编写程序，计算给定整数序列的最大子列和。

本题旨在测试各种不同的算法在各种数据情况下的表现。各组测试数据特点如下：

* 数据1：与样例等价，测试基本正确性；
* 数据2：102个随机整数；
* 数据3：103个随机整数；
* 数据4：104个随机整数；
* 数据5：105个随机整数；

输入格式:

输入第1行给出正整数*K* (≤100000)；第2行给出*K*个整数，其间以空格分隔。

输出格式:

在一行中输出最大子列和。如果序列中所有整数皆为负数，则输出0。

输入样例:

6

-2 11 -4 13 -5 -2

输出样例:

20

# 习题1.8 二分查找（20 分）

本题要求实现二分查找算法。

### 函数接口定义：

Position BinarySearch( List L, ElementType X );

其中List结构定义如下：

typedef int Position;

typedef struct LNode \*List;

struct LNode {

ElementType Data[MAXSIZE];

Position Last; /\* 保存线性表中最后一个元素的位置 \*/

};

L是用户传入的一个线性表，其中ElementType元素可以通过>、==、<进行比较，并且题目保证传入的数据是递增有序的。函数BinarySearch要查找X在Data中的位置，即数组下标（注意：元素从下标1开始存储）。找到则返回下标，否则返回一个特殊的失败标记NotFound。

### 裁判测试程序样例：

#include <stdio.h>

#include <stdlib.h>

#define MAXSIZE 10

#define NotFound 0

typedef int ElementType;

typedef int Position;

typedef struct LNode \*List;

struct LNode {

ElementType Data[MAXSIZE];

Position Last; /\* 保存线性表中最后一个元素的位置 \*/

};

List ReadInput(); /\* 裁判实现，细节不表。元素从下标1开始存储 \*/

Position BinarySearch( List L, ElementType X );

int main()

{

List L;

ElementType X;

Position P;

L = ReadInput();

scanf("%d", &X);

P = BinarySearch( L, X );

printf("%d\n", P);

return 0;

}

/\* 你的代码将被嵌在这里 \*/

### 输入样例1：

5

12 31 55 89 101

31

### 输出样例1：

2

### 输入样例2：

3

26 78 233

31

### 输出样例2：

0

# 习题1.9 有序数组的插入（20 分）

本题要求将任一给定元素插入从大到小排好序的数组中合适的位置，以保持结果依然有序。

### 函数接口定义：

bool Insert( List L, ElementType X );

其中List结构定义如下：

typedef int Position;

typedef struct LNode \*List;

struct LNode {

ElementType Data[MAXSIZE];

Position Last; /\* 保存线性表中最后一个元素的位置 \*/

};

L是用户传入的一个线性表，其中ElementType元素可以通过>、==、<进行比较，并且题目保证传入的数据是递减有序的。函数Insert要将X插入Data[]中合适的位置，以保持结果依然有序（注意：元素从下标0开始存储）。但如果X已经在Data[]中了，就不要插入，返回失败的标记false；如果插入成功，则返回true。另外，因为Data[]中最多只能存MAXSIZE个元素，所以如果插入新元素之前已经满了，也不要插入，而是返回失败的标记false。

### 裁判测试程序样例：

#include <stdio.h>

#include <stdlib.h>

#define MAXSIZE 10

typedef enum {false, true} bool;

typedef int ElementType;

typedef int Position;

typedef struct LNode \*List;

struct LNode {

ElementType Data[MAXSIZE];

Position Last; /\* 保存线性表中最后一个元素的位置 \*/

};

List ReadInput(); /\* 裁判实现，细节不表。元素从下标0开始存储 \*/

void PrintList( List L ); /\* 裁判实现，细节不表 \*/

bool Insert( List L, ElementType X );

int main()

{

List L;

ElementType X;

L = ReadInput();

scanf("%d", &X);

if ( Insert( L, X ) == false )

printf("Insertion failed.\n");

PrintList( L );

return 0;

}

/\* 你的代码将被嵌在这里 \*/

### 输入样例1：

5

35 12 8 7 3

10

### 输出样例1：

35 12 10 8 7 3

Last = 5

### 输入样例2：

6

35 12 10 8 7 3

8

### 输出样例2：

Insertion failed.

35 12 10 8 7 3

Last = 5

# 习题2.1 简单计算器（20 分）

模拟简单运算器的工作。假设计算器只能进行加减乘除运算，运算数和结果都是整数，四种运算符的优先级相同，按从左到右的顺序计算。

### 输入格式:

输入在一行中给出一个四则运算算式，没有空格，且至少有一个操作数。遇等号”=”说明输入结束。

### 输出格式:

在一行中输出算式的运算结果，或者如果除法分母为0或有非法运算符，则输出错误信息“ERROR”。

### 输入样例:

1+2\*10-10/2=

### 输出样例:

10

# 习题2.2 数组循环左移（20 分）

本题要求实现一个对数组进行循环左移的简单函数：一个数组*a*中存有*n*（>0）个整数，在不允许使用另外数组的前提下，将每个整数循环向左移*m*（≥0）个位置，即将*a*中的数据由（*a*​0​​*a*​1​​⋯*a*​*n*−1​​）变换为（*a*​*m*​​⋯*a*​*n*−1​​*a*​0​​*a*​1​​⋯*a*​*m*−1​​）（最前面的*m*个数循环移至最后面的*m*个位置）。如果还需要考虑程序移动数据的次数尽量少，要如何设计移动的方法？

### 输入格式:

输入第1行给出正整数*n*（≤100）和整数*m*（≥0）；第2行给出*n*个整数，其间以空格分隔。

### 输出格式:

在一行中输出循环左移*m*位以后的整数序列，之间用空格分隔，序列结尾不能有多余空格。

### 输入样例：

8 3

1 2 3 4 5 6 7 8

### 输出样例：

4 5 6 7 8 1 2 3

# 习题2.3 数列求和-加强版（20 分）

给定某数字*A*（1≤*A*≤9）以及非负整数*N*（0≤*N*≤100000），求数列之和*S*=*A*+*AA*+*AAA*+⋯+*AA*⋯*A*（*N*个*A*）。例如*A*=1, *N*=3时，*S*=1+11+111=123。

### 输入格式：

输入数字*A*与非负整数*N*。

### 输出格式：

输出其*N*项数列之和*S*的值。

### 输入样例：

1 3

### 输出样例：

123

# 习题2.4 递增的整数序列链表的插入（15 分）

本题要求实现一个函数，在递增的整数序列链表（带头结点）中插入一个新整数，并保持该序列的有序性。

### 函数接口定义：

List Insert( List L, ElementType X );

其中List结构定义如下：

typedef struct Node \*PtrToNode;

struct Node {

ElementType Data; /\* 存储结点数据 \*/

PtrToNode Next; /\* 指向下一个结点的指针 \*/

};

typedef PtrToNode List; /\* 定义单链表类型 \*/

L是给定的带头结点的单链表，其结点存储的数据是递增有序的；函数Insert要将X插入L，并保持该序列的有序性，返回插入后的链表头指针。

### 裁判测试程序样例：

#include <stdio.h>

#include <stdlib.h>

typedef int ElementType;

typedef struct Node \*PtrToNode;

struct Node {

ElementType Data;

PtrToNode Next;

};

typedef PtrToNode List;

List Read(); /\* 细节在此不表 \*/

void Print( List L ); /\* 细节在此不表 \*/

List Insert( List L, ElementType X );

int main()

{

List L;

ElementType X;

L = Read();

scanf("%d", &X);

L = Insert(L, X);

Print(L);

return 0;

}

/\* 你的代码将被嵌在这里 \*/

### 输入样例：

5

1 2 4 5 6

3

### 输出样例：

1 2 3 4 5 6

# 习题2.5 两个有序链表序列的合并（15 分）

本题要求实现一个函数，将两个链表表示的递增整数序列合并为一个非递减的整数序列。

### 函数接口定义：

List Merge( List L1, List L2 );

其中List结构定义如下：

typedef struct Node \*PtrToNode;

struct Node {

ElementType Data; /\* 存储结点数据 \*/

PtrToNode Next; /\* 指向下一个结点的指针 \*/

};

typedef PtrToNode List; /\* 定义单链表类型 \*/

L1和L2是给定的带头结点的单链表，其结点存储的数据是递增有序的；函数Merge要将L1和L2合并为一个非递减的整数序列。应直接使用原序列中的结点，返回归并后的链表头指针。

### 裁判测试程序样例：

#include <stdio.h>

#include <stdlib.h>

typedef int ElementType;

typedef struct Node \*PtrToNode;

struct Node {

ElementType Data;

PtrToNode Next;

};

typedef PtrToNode List;

List Read(); /\* 细节在此不表 \*/

void Print( List L ); /\* 细节在此不表；空链表将输出NULL \*/

List Merge( List L1, List L2 );

int main()

{

List L1, L2, L;

L1 = Read();

L2 = Read();

L = Merge(L1, L2);

Print(L);

Print(L1);

Print(L2);

return 0;

}

/\* 你的代码将被嵌在这里 \*/

### 输入样例：

3

1 3 5

5

2 4 6 8 10

### 输出样例：

1 2 3 4 5 6 8 10

NULL

NULL

作者: DS课程组

单位: 浙江大学

时间限制: 400ms

内存限制: 64MB

代码长度限制: 16KB

# 习题2.6 递归求简单交错幂级数的部分和（15 分）

本题要求实现一个函数，计算下列简单交错幂级数的部分和：

*f*(*x*,*n*)=*x*−*x*​2​​+*x*​3​​−*x*​4​​+⋯+(−1)​*n*−1​​*x*​*n*​​

### 函数接口定义：

double fn( double x, int n );

其中题目保证传入的n是正整数，并且输入输出都在双精度范围内。函数fn应返回上述级数的部分和。建议尝试用递归实现。

### 裁判测试程序样例：

#include <stdio.h>

double fn( double x, int n );

int main()

{

double x;

int n;

scanf("%lf %d", &x, &n);

printf("%.2f\n", fn(x,n));

return 0;

}

/\* 你的代码将被嵌在这里 \*/

### 输入样例：

0.5 12

### 输出样例：

0.33

# 习题2.7 弹球距离（15 分）

设有一个球从高度为*h*米的地方落下，碰到地面后又弹到高度为原来*p*倍的位置，然后又落下，再弹起，再落下…。请编写函数求初始高度为*h*的球下落后到基本停下来（高度小于给定阈值TOL）时在空中所经过的路程总和。

### 函数接口定义：

double dist( double h, double p );

其中h是球的初始高度，p是球弹起高度与弹起前落下高度的比值；函数dist要返回球下落后到基本停下来时在空中所经过的路程总和。注意：当弹起的高度小于裁判程序定义的常数TOL时，弹起的距离不计算在内。

### 裁判测试程序样例：

#include <stdio.h>

#define TOL 1E-2

double dist( double h, double p );

int main()

{

double h, p, d;

scanf("%lf %lf", &h, &p);

d = dist(h, p);

printf("%.6f\n", d);

return 0;

}

/\* 你的代码将被嵌在这里 \*/

### 输入样例：

1.0 0.4

### 输出样例：

2.319680

# 习题2.8 输出全排列（20 分）

请编写程序输出前*n*个正整数的全排列（*n*<10），并通过9个测试用例（即*n*从1到9）观察*n*逐步增大时程序的运行时间。

### 输入格式:

输入给出正整数*n*（<10）。

### 输出格式:

输出1到*n*的全排列。每种排列占一行，数字间无空格。排列的输出顺序为字典序，即序列*a*​1​​,*a*​2​​,⋯,*a*​*n*​​排在序列*b*​1​​,*b*​2​​,⋯,*b*​*n*​​之前，如果存在*k*使得*a*​1​​=*b*​1​​,⋯,*a*​*k*​​=*b*​*k*​​ 并且 *a*​*k*+1​​<*b*​*k*+1​​。

### 输入样例：

3

### 输出样例：

123

132

213

231

312

321

# 习题3.3 线性表元素的区间删除（20 分）

给定一个顺序存储的线性表，请设计一个函数删除所有值大于min而且小于max的元素。删除后表中剩余元素保持顺序存储，并且相对位置不能改变。

### 函数接口定义：

List Delete( List L, ElementType minD, ElementType maxD );

其中List结构定义如下：

typedef int Position;

typedef struct LNode \*List;

struct LNode {

ElementType Data[MAXSIZE];

Position Last; /\* 保存线性表中最后一个元素的位置 \*/

};

L是用户传入的一个线性表，其中ElementType元素可以通过>、==、<进行比较；minD和maxD分别为待删除元素的值域的下、上界。函数Delete应将Data[]中所有值大于minD而且小于maxD的元素删除，同时保证表中剩余元素保持顺序存储，并且相对位置不变，最后返回删除后的表。

### 裁判测试程序样例：

#include <stdio.h>

#define MAXSIZE 20

typedef int ElementType;

typedef int Position;

typedef struct LNode \*List;

struct LNode {

ElementType Data[MAXSIZE];

Position Last; /\* 保存线性表中最后一个元素的位置 \*/

};

List ReadInput(); /\* 裁判实现，细节不表。元素从下标0开始存储 \*/

void PrintList( List L ); /\* 裁判实现，细节不表 \*/

List Delete( List L, ElementType minD, ElementType maxD );

int main()

{

List L;

ElementType minD, maxD;

int i;

L = ReadInput();

scanf("%d %d", &minD, &maxD);

L = Delete( L, minD, maxD );

PrintList( L );

return 0;

}

/\* 你的代码将被嵌在这里 \*/

### 输入样例：

10

4 -8 2 12 1 5 9 3 3 10

0 4

### 输出样例：

4 -8 12 5 9 10

# 习题3.4 最长连续递增子序列（20 分）

给定一个顺序存储的线性表，请设计一个算法查找该线性表中最长的连续递增子序列。例如，(1,9,2,5,7,3,4,6,8,0)中最长的递增子序列为(3,4,6,8)。

### 输入格式:

输入第1行给出正整数*n*（≤10​5​​）；第2行给出*n*个整数，其间以空格分隔。

### 输出格式:

在一行中输出第一次出现的最长连续递增子序列，数字之间用空格分隔，序列结尾不能有多余空格。

### 输入样例：

15

1 9 2 5 7 3 4 6 8 0 11 15 17 17 10

### 输出样例：

3 4 6 8

# 习题3.5 求链表的倒数第m个元素（20 分）

请设计时间和空间上都尽可能高效的算法，在不改变链表的前提下，求链式存储的线性表的倒数第m（>0）个元素。

### 函数接口定义：

ElementType Find( List L, int m );

其中List结构定义如下：

typedef struct Node \*PtrToNode;

struct Node {

ElementType Data; /\* 存储结点数据 \*/

PtrToNode Next; /\* 指向下一个结点的指针 \*/

};

typedef PtrToNode List; /\* 定义单链表类型 \*/

L是给定的带头结点的单链表；函数Find要将L的倒数第m个元素返回，并不改变原链表。如果这样的元素不存在，则返回一个错误标志ERROR。

### 裁判测试程序样例：

#include <stdio.h>

#include <stdlib.h>

#define ERROR -1

typedef int ElementType;

typedef struct Node \*PtrToNode;

struct Node {

ElementType Data;

PtrToNode Next;

};

typedef PtrToNode List;

List Read(); /\* 细节在此不表 \*/

void Print( List L ); /\* 细节在此不表 \*/

ElementType Find( List L, int m );

int main()

{

List L;

int m;

L = Read();

scanf("%d", &m);

printf("%d\n", Find(L,m));

Print(L);

return 0;

}

/\* 你的代码将被嵌在这里 \*/

### 输入样例：

5

1 2 4 5 6

3

### 输出样例：

4

1 2 4 5 6

# 习题3.6 一元多项式的乘法与加法运算（20 分）

设计函数分别求两个一元多项式的乘积与和。

### 输入格式:

输入分2行，每行分别先给出多项式非零项的个数，再以指数递降方式输入一个多项式非零项系数和指数（绝对值均为不超过1000的整数）。数字间以空格分隔。

### 输出格式:

输出分2行，分别以指数递降方式输出乘积多项式以及和多项式非零项的系数和指数。数字间以空格分隔，但结尾不能有多余空格。零多项式应输出0 0。

### 输入样例:

4 3 4 -5 2 6 1 -2 0

3 5 20 -7 4 3 1

### 输出样例:

15 24 -25 22 30 21 -10 20 -21 8 35 6 -33 5 14 4 -15 3 18 2 -6 1

5 20 -4 4 -5 2 9 1 -2 0

# 习题3.8 符号配对（20 分）

请编写程序检查C语言源程序中下列符号是否配对：/\*与\*/、(与)、[与]、{与}。

### 输入格式:

输入为一个C语言源程序。当读到某一行中只有一个句点.和一个回车的时候，标志着输入结束。程序中需要检查配对的符号不超过100个。

### 输出格式:

首先，如果所有符号配对正确，则在第一行中输出YES，否则输出NO。然后在第二行中指出第一个不配对的符号：如果缺少左符号，则输出?-右符号；如果缺少右符号，则输出左符号-?。

### 输入样例1：

void test()

{

int i, A[10];

for (i=0; i<10; i++) /\*/

A[i] = i;

}

.

### 输出样例1：

NO

/\*-?

### 输入样例2：

void test()

{

int i, A[10];

for (i=0; i<10; i++) /\*\*/

A[i] = i;

}]

.

### 输出样例2：

NO

?-]

### 输入样例3：

void test()

{

int i

double A[10];

for (i=0; i<10; i++) /\*\*/

A[i] = 0.1\*i;

}

.

### 输出样例3：

YES

# 习题3.9 堆栈操作合法性（20 分）

假设以S和X分别表示入栈和出栈操作。如果根据一个仅由S和X构成的序列，对一个空堆栈进行操作，相应操作均可行（如没有出现删除时栈空）且最后状态也是栈空，则称该序列是合法的堆栈操作序列。请编写程序，输入S和X序列，判断该序列是否合法。

### 输入格式:

输入第一行给出两个正整数N和M，其中N是待测序列的个数，M（≤50）是堆栈的最大容量。随后N行，每行中给出一个仅由S和X构成的序列。序列保证不为空，且长度不超过100。

### 输出格式:

对每个序列，在一行中输出YES如果该序列是合法的堆栈操作序列，或NO如果不是。

### 输入样例：

4 10

SSSXXSXXSX

SSSXXSXXS

SSSSSSSSSSXSSXXXXXXXXXXX

SSSXXSXXX

### 输出样例：

YES

NO

NO

NO

# 习题3.10 汉诺塔的非递归实现（25 分）

借助堆栈以非递归（循环）方式求解汉诺塔的问题（n, a, b, c），即将N个盘子从起始柱（标记为“a”）通过借助柱（标记为“b”）移动到目标柱（标记为“c”），并保证每个移动符合汉诺塔问题的要求。

### 输入格式:

输入为一个正整数N，即起始柱上的盘数。

### 输出格式:

每个操作（移动）占一行，按柱1 -> 柱2的格式输出。

### 输入样例:

3

### 输出样例:

a -> c

a -> b

c -> b

a -> c

b -> a

b -> c

a -> c

作者: DS课程组

单位: 浙江大学

时间限制: 400ms

内存限制: 64MB

代码长度限制: 16KB

# 习题3.11 表达式转换（25 分）

算术表达式有前缀表示法、中缀表示法和后缀表示法等形式。日常使用的算术表达式是采用中缀表示法，即二元运算符位于两个运算数中间。请设计程序将中缀表达式转换为后缀表达式。

输入格式:

输入在一行中给出不含空格的中缀表达式，可包含+、-、\*、\以及左右括号()，表达式不超过20个字符。

输出格式:

在一行中输出转换后的后缀表达式，要求不同对象（运算数、运算符号）之间以空格分隔，但结尾不得有多余空格。

输入样例:

2+3\*(7-4)+8/4

输出样例:

2 3 7 4 - \* + 8 4 / +

# 习题3.12 另类循环队列（20 分）

如果用一个循环数组表示队列，并且只设队列头指针Front，不设尾指针Rear，而是另设Count记录队列中元素个数。请编写算法实现队列的入队和出队操作。

### 函数接口定义：

bool AddQ( Queue Q, ElementType X );

ElementType DeleteQ( Queue Q );

其中Queue结构定义如下：

typedef int Position;

typedef struct QNode \*PtrToQNode;

struct QNode {

ElementType \*Data; /\* 存储元素的数组 \*/

Position Front; /\* 队列的头指针 \*/

int Count; /\* 队列中元素个数 \*/

int MaxSize; /\* 队列最大容量 \*/

};

typedef PtrToQNode Queue;

注意：如果队列已满，AddQ函数必须输出“Queue Full”并且返回false；如果队列是空的，则DeleteQ函数必须输出“Queue Empty”，并且返回ERROR。

### 裁判测试程序样例：

#include <stdio.h>

#include <stdlib.h>

#define ERROR -1

typedef int ElementType;

typedef enum { addq, delq, end } Operation;

typedef enum { false, true } bool;

typedef int Position;

typedef struct QNode \*PtrToQNode;

struct QNode {

ElementType \*Data; /\* 存储元素的数组 \*/

Position Front; /\* 队列的头、尾指针 \*/

int Count; /\* 队列中元素个数 \*/

int MaxSize; /\* 队列最大容量 \*/

};

typedef PtrToQNode Queue;

Queue CreateQueue( int MaxSize )

{

Queue Q = (Queue)malloc(sizeof(struct QNode));

Q->Data = (ElementType \*)malloc(MaxSize \* sizeof(ElementType));

Q->Front = 0;

Q->Count = 0;

Q->MaxSize = MaxSize;

return Q;

}

bool AddQ( Queue Q, ElementType X );

ElementType DeleteQ( Queue Q );

Operation GetOp(); /\* 裁判实现，细节不表 \*/

int main()

{

ElementType X;

Queue Q;

int N, done = 0;

scanf("%d", &N);

Q = CreateQueue(N);

while ( !done ) {

switch( GetOp() ) {

case addq:

scanf("%d", &X);

AddQ(Q, X);

break;

case delq:

X = DeleteQ(Q);

if ( X!=ERROR ) printf("%d is out\n", X);

break;

case end:

while (Q->Count) printf("%d ", DeleteQ(Q));

done = 1;

break;

}

}

return 0;

}

/\* 你的代码将被嵌在这里 \*/

### 输入样例：

4

Del

Add 5

Add 4

Add 3

Del

Del

Add 2

Add 1

Add 0

Add 10

End

### 输出样例：

Queue Empty

5 is out

4 is out

Queue Full

3 2 1 0

# 习题3.13 双端队列（25 分）

双端队列（deque，即double-ended queue的缩写）是一种具有队列和栈性质的数据结构，即可以（也只能）在线性表的两端进行插入和删除。若以顺序存储方式实现双端队列，请编写例程实现下列操作：

* Push(X,D)：将元素X插入到双端队列D的头；
* Pop(D)：删除双端队列D的头元素，并返回；
* Inject(X,D)：将元素X插入到双端队列D的尾部；
* Eject(D)：删除双端队列D的尾部元素，并返回。

函数接口定义：

bool Push( ElementType X, Deque D );

ElementType Pop( Deque D );

bool Inject( ElementType X, Deque D );

ElementType Eject( Deque D );

其中Deque结构定义如下：

typedef int Position;

typedef struct QNode \*PtrToQNode;

struct QNode {

ElementType \*Data; /\* 存储元素的数组 \*/

Position Front, Rear; /\* 队列的头、尾指针 \*/

int MaxSize; /\* 队列最大容量 \*/

};

typedef PtrToQNode Deque;

注意：Push和Inject应该在正常执行完操作后返回true，或者在出现非正常情况时返回false。当Front和Rear相等时队列为空，Pop和Eject必须返回由裁判程序定义的ERROR。

裁判测试程序样例：

#include <stdio.h>

#include <stdlib.h>

#define ERROR -1

typedef int ElementType;

typedef enum { push, pop, inject, eject, end } Operation;

typedef enum { false, true } bool;

typedef int Position;

typedef struct QNode \*PtrToQNode;

struct QNode {

ElementType \*Data; /\* 存储元素的数组 \*/

Position Front, Rear; /\* 队列的头、尾指针 \*/

int MaxSize; /\* 队列最大容量 \*/

};

typedef PtrToQNode Deque;

Deque CreateDeque( int MaxSize )

{ /\* 注意：为区分空队列和满队列，需要多开辟一个空间 \*/

Deque D = (Deque)malloc(sizeof(struct QNode));

MaxSize++;

D->Data = (ElementType \*)malloc(MaxSize \* sizeof(ElementType));

D->Front = D->Rear = 0;

D->MaxSize = MaxSize;

return D;

}

bool Push( ElementType X, Deque D );

ElementType Pop( Deque D );

bool Inject( ElementType X, Deque D );

ElementType Eject( Deque D );

Operation GetOp(); /\* 裁判实现，细节不表 \*/

void PrintDeque( Deque D ); /\* 裁判实现，细节不表 \*/

int main()

{

ElementType X;

Deque D;

int N, done = 0;

scanf("%d", &N);

D = CreateDeque(N);

while (!done) {

switch(GetOp()) {

case push:

scanf("%d", &X);

if (!Push(X, D)) printf("Deque is Full!\n");

break;

case pop:

X = Pop(D);

if ( X==ERROR ) printf("Deque is Empty!\n");

else printf("%d is out\n", X);

break;

case inject:

scanf("%d", &X);

if (!Inject(X, D)) printf("Deque is Full!\n");

break;

case eject:

X = Eject(D);

if ( X==ERROR ) printf("Deque is Empty!\n");

else printf("%d is out\n", X);

break;

case end:

PrintDeque(D);

done = 1;

break;

}

}

return 0;

}

/\* 你的代码将被嵌在这里 \*/

输入样例：

3

Pop

Inject 1

Pop

Eject

Push 2

Push 3

Eject

Inject 4

Inject 5

Inject 6

Push 7

Pop

End

输出样例：

Deque is Empty!

1 is out

Deque is Empty!

2 is out

Deque is Full!

Deque is Full!

3 is out

Inside Deque: 4 5

# 习题3.14 另类堆栈（15 分）

在栈的顺序存储实现中，另有一种方法是将Top定义为栈顶的上一个位置。请编写程序实现这种定义下堆栈的入栈、出栈操作。如何判断堆栈为空或者满？

### 函数接口定义：

bool Push( Stack S, ElementType X );

ElementType Pop( Stack S );

其中Stack结构定义如下：

typedef int Position;

typedef struct SNode \*PtrToSNode;

struct SNode {

ElementType \*Data; /\* 存储元素的数组 \*/

Position Top; /\* 栈顶指针 \*/

int MaxSize; /\* 堆栈最大容量 \*/

};

typedef PtrToSNode Stack;

注意：如果堆栈已满，Push函数必须输出“Stack Full”并且返回false；如果队列是空的，则Pop函数必须输出“Stack Empty”，并且返回ERROR。

### 裁判测试程序样例：

#include <stdio.h>

#include <stdlib.h>

#define ERROR -1

typedef int ElementType;

typedef enum { push, pop, end } Operation;

typedef enum { false, true } bool;

typedef int Position;

typedef struct SNode \*PtrToSNode;

struct SNode {

ElementType \*Data; /\* 存储元素的数组 \*/

Position Top; /\* 栈顶指针 \*/

int MaxSize; /\* 堆栈最大容量 \*/

};

typedef PtrToSNode Stack;

Stack CreateStack( int MaxSize )

{

Stack S = (Stack)malloc(sizeof(struct SNode));

S->Data = (ElementType \*)malloc(MaxSize \* sizeof(ElementType));

S->Top = 0;

S->MaxSize = MaxSize;

return S;

}

bool Push( Stack S, ElementType X );

ElementType Pop( Stack S );

Operation GetOp(); /\* 裁判实现，细节不表 \*/

void PrintStack( Stack S ); /\* 裁判实现，细节不表 \*/

int main()

{

ElementType X;

Stack S;

int N, done = 0;

scanf("%d", &N);

S = CreateStack(N);

while ( !done ) {

switch( GetOp() ) {

case push:

scanf("%d", &X);

Push(S, X);

break;

case pop:

X = Pop(S);

if ( X!=ERROR ) printf("%d is out\n", X);

break;

case end:

PrintStack(S);

done = 1;

break;

}

}

return 0;

}

/\* 你的代码将被嵌在这里 \*/

### 输入样例：

4

Pop

Push 5

Push 4

Push 3

Pop

Pop

Push 2

Push 1

Push 0

Push 10

End

### 输出样例：

Stack Empty

3 is out

4 is out

Stack Full

0 1 2 5

# 练习4.1 根据后序和中序遍历输出先序遍历（25 分）

本题要求根据给定的一棵二叉树的后序遍历和中序遍历结果，输出该树的先序遍历结果。

### 输入格式:

第一行给出正整数*N*(≤30)，是树中结点的个数。随后两行，每行给出*N*个整数，分别对应后序遍历和中序遍历结果，数字间以空格分隔。题目保证输入正确对应一棵二叉树。

### 输出格式:

在一行中输出Preorder:以及该树的先序遍历结果。数字间有1个空格，行末不得有多余空格。

### 输入样例:

7

2 3 1 5 7 6 4

1 2 3 4 5 6 7

### 输出样例:

Preorder: 4 1 3 2 6 5 7

# 练习4.2 平衡二叉树的根（25 分）

将给定的一系列数字插入初始为空的AVL树，请你输出最后生成的AVL树的根结点的值。

### 输入格式:

输入的第一行给出一个正整数*N*（≤20），随后一行给出*N*个不同的整数，其间以空格分隔。

### 输出格式:

在一行中输出顺序插入上述整数到一棵初始为空的AVL树后，该树的根结点的值。

### 输入样例1:

5

88 70 61 96 120

### 输出样例1:

70

### 输入样例2:

7

88 70 61 96 120 90 65

### 输出样例2:

88

# 习题4.3 是否二叉搜索树（25 分）

本题要求实现函数，判断给定二叉树是否二叉搜索树。

函数接口定义：

bool IsBST ( BinTree T );

其中BinTree结构定义如下：

typedef struct TNode \*Position;

typedef Position BinTree;

struct TNode{

ElementType Data;

BinTree Left;

BinTree Right;

};

函数IsBST须判断给定的T是否二叉搜索树，即满足如下定义的二叉树：

定义：一个二叉搜索树是一棵二叉树，它可以为空。如果不为空，它将满足以下性质：

* 非空左子树的所有键值小于其根结点的键值。
* 非空右子树的所有键值大于其根结点的键值。
* 左、右子树都是二叉搜索树。

如果T是二叉搜索树，则函数返回true，否则返回false。

裁判测试程序样例：

#include <stdio.h>

#include <stdlib.h>

typedef enum { false, true } bool;

typedef int ElementType;

typedef struct TNode \*Position;

typedef Position BinTree;

struct TNode{

ElementType Data;

BinTree Left;

BinTree Right;

};

BinTree BuildTree(); /\* 由裁判实现，细节不表 \*/

bool IsBST ( BinTree T );

int main()

{

BinTree T;

T = BuildTree();

if ( IsBST(T) ) printf("Yes\n");

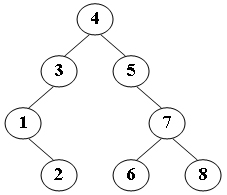
else printf("No\n");

return 0;

}

/\* 你的代码将被嵌在这里 \*/

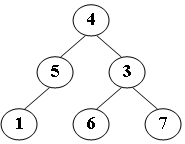
输入样例1：如下图



输出样例1：

Yes

输入样例2：如下图



输出样例2：

No

作者: DS课程组

单位: 浙江大学

时间限制: 400ms

内存限制: 64MB

代码长度限制: 16KB

# 练习4.3 堆中的路径（25 分）

将一系列给定数字插入一个初始为空的小顶堆H[]。随后对任意给定的下标i，打印从H[i]到根结点的路径。

### 输入格式:

每组测试第1行包含2个正整数*N*和*M*(≤1000)，分别是插入元素的个数、以及需要打印的路径条数。下一行给出区间[-10000, 10000]内的*N*个要被插入一个初始为空的小顶堆的整数。最后一行给出*M*个下标。

### 输出格式:

对输入中给出的每个下标i，在一行中输出从H[i]到根结点的路径上的数据。数字间以1个空格分隔，行末不得有多余空格。

### 输入样例:

5 3

46 23 26 24 10

5 4 3

### 输出样例:

24 23 10

46 23 10

26 10

# 习题4.5 顺序存储的二叉树的最近的公共祖先问题（25 分）

设顺序存储的二叉树中有编号为*i*和*j*的两个结点，请设计算法求出它们最近的公共祖先结点的编号和值。

### 输入格式:

输入第1行给出正整数*n*（≤1000），即顺序存储的最大容量；第2行给出*n*个非负整数，其间以空格分隔。其中0代表二叉树中的空结点（如果第1个结点为0，则代表一棵空树）；第3行给出一对结点编号*i*和*j*。

题目保证输入正确对应一棵二叉树，且1≤*i*,*j*≤*n*。

### 输出格式:

如果*i*或*j*对应的是空结点，则输出ERROR: T[x] is NULL，其中x是*i*或*j*中先发现错误的那个编号；否则在一行中输出编号为*i*和*j*的两个结点最近的公共祖先结点的编号和值，其间以1个空格分隔。

### 输入样例1：

15

4 3 5 1 10 0 7 0 2 0 9 0 0 6 8

11 4

### 输出样例1：

2 3

### 输入样例2：

15

4 3 5 1 0 0 7 0 2 0 9 0 0 6 8

12 8

### 输出样例2：

ERROR: T[12] is NULL

# 习题5.10 线性探测法的查找函数（20 分）

试实现线性探测法的查找函数。

### 函数接口定义：

Position Find( HashTable H, ElementType Key );

其中HashTable是开放地址散列表，定义如下：

#define MAXTABLESIZE 100000 /\* 允许开辟的最大散列表长度 \*/

typedef int ElementType; /\* 关键词类型用整型 \*/

typedef int Index; /\* 散列地址类型 \*/

typedef Index Position; /\* 数据所在位置与散列地址是同一类型 \*/

/\* 散列单元状态类型，分别对应：有合法元素、空单元、有已删除元素 \*/

typedef enum { Legitimate, Empty, Deleted } EntryType;

typedef struct HashEntry Cell; /\* 散列表单元类型 \*/

struct HashEntry{

ElementType Data; /\* 存放元素 \*/

EntryType Info; /\* 单元状态 \*/

};

typedef struct TblNode \*HashTable; /\* 散列表类型 \*/

struct TblNode { /\* 散列表结点定义 \*/

int TableSize; /\* 表的最大长度 \*/

Cell \*Cells; /\* 存放散列单元数据的数组 \*/

};

函数Find应根据裁判定义的散列函数Hash( Key, H->TableSize )从散列表H中查到Key的位置并返回。如果Key不存在，则返回线性探测法找到的第一个空单元的位置；若没有空单元，则返回ERROR。

### 裁判测试程序样例：

#include <stdio.h>

#define MAXTABLESIZE 100000 /\* 允许开辟的最大散列表长度 \*/

typedef int ElementType; /\* 关键词类型用整型 \*/

typedef int Index; /\* 散列地址类型 \*/

typedef Index Position; /\* 数据所在位置与散列地址是同一类型 \*/

/\* 散列单元状态类型，分别对应：有合法元素、空单元、有已删除元素 \*/

typedef enum { Legitimate, Empty, Deleted } EntryType;

typedef struct HashEntry Cell; /\* 散列表单元类型 \*/

struct HashEntry{

ElementType Data; /\* 存放元素 \*/

EntryType Info; /\* 单元状态 \*/

};

typedef struct TblNode \*HashTable; /\* 散列表类型 \*/

struct TblNode { /\* 散列表结点定义 \*/

int TableSize; /\* 表的最大长度 \*/

Cell \*Cells; /\* 存放散列单元数据的数组 \*/

};

HashTable BuildTable(); /\* 裁判实现，细节不表 \*/

Position Hash( ElementType Key, int TableSize )

{

return (Key % TableSize);

}

#define ERROR -1

Position Find( HashTable H, ElementType Key );

int main()

{

HashTable H;

ElementType Key;

Position P;

H = BuildTable();

scanf("%d", &Key);

P = Find(H, Key);

if (P==ERROR)

printf("ERROR: %d is not found and the table is full.\n", Key);

else if (H->Cells[P].Info == Legitimate)

printf("%d is at position %d.\n", Key, P);

else

printf("%d is not found. Position %d is returned.\n", Key, P);

return 0;

}

/\* 你的代码将被嵌在这里 \*/

### 输入样例1：（注：-1表示该位置为空。下同。）

11

11 88 21 -1 -1 5 16 7 6 38 10

38

### 输出样例1：

38 is at position 9.

### 输入样例2：

11

11 88 21 -1 -1 5 16 7 6 38 10

41

### 输出样例2：

41 is not found. Position 3 is returned.

### 输入样例3：

11

11 88 21 3 14 5 16 7 6 38 10

41

### 输出样例3：

ERROR: 41 is not found and the table is full.

# 习题5.11 分离链接法的删除操作函数（20 分）

试实现分离链接法的删除操作函数。

### 函数接口定义：

bool Delete( HashTable H, ElementType Key );

其中HashTable是分离链接散列表，定义如下：

typedef struct LNode \*PtrToLNode;

struct LNode {

ElementType Data;

PtrToLNode Next;

};

typedef PtrToLNode Position;

typedef PtrToLNode List;

typedef struct TblNode \*HashTable; /\* 散列表类型 \*/

struct TblNode { /\* 散列表结点定义 \*/

int TableSize; /\* 表的最大长度 \*/

List Heads; /\* 指向链表头结点的数组 \*/

};

函数Delete应根据裁判定义的散列函数Hash( Key, H->TableSize )从散列表H中查到Key的位置并删除之，然后输

出一行文字：Key is deleted from list Heads[i]，其中Key是传入的被删除的关键词，i是Key所在的链表的编

号；最后返回true。如果Key不存在，则返回false。

### 裁判测试程序样例：

#include <stdio.h>

#include <string.h>

#define KEYLENGTH 15 /\* 关键词字符串的最大长度 \*/

typedef char ElementType[KEYLENGTH+1]; /\* 关键词类型用字符串 \*/

typedef int Index; /\* 散列地址类型 \*/

typedef enum {false, true} bool;

typedef struct LNode \*PtrToLNode;

struct LNode {

ElementType Data;

PtrToLNode Next;

};

typedef PtrToLNode Position;

typedef PtrToLNode List;

typedef struct TblNode \*HashTable; /\* 散列表类型 \*/

struct TblNode { /\* 散列表结点定义 \*/

int TableSize; /\* 表的最大长度 \*/

List Heads; /\* 指向链表头结点的数组 \*/

};

Index Hash( ElementType Key, int TableSize )

{

return (Key[0]-'a')%TableSize;

}

HashTable BuildTable(); /\* 裁判实现，细节不表 \*/

bool Delete( HashTable H, ElementType Key );

int main()

{

HashTable H;

ElementType Key;

H = BuildTable();

scanf("%s", Key);

if (Delete(H, Key) == false)

printf("ERROR: %s is not found\n", Key);

if (Delete(H, Key) == true)

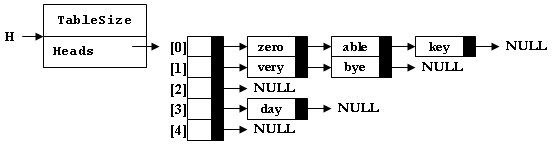
printf("Are you kidding me?\n");

return 0;

}

/\* 你的代码将被嵌在这里 \*/

### 输入样例1：散列表如下图



able

### 输出样例1：

able is deleted from list Heads[0]

### 输入样例2：散列表如样例1图

date

### 输出样例2：

ERROR: date is not found

# 习题5.13 词频统计（30 分）

请编写程序，对一段英文文本，统计其中所有不同单词的个数，以及词频最大的前10%的单词。

所谓“单词”，是指由不超过80个单词字符组成的连续字符串，但长度超过15的单词将只截取保留前15个单词字符。而合法的“单词字符”为大小写字母、数字和下划线，其它字符均认为是单词分隔符。

### 输入格式:

输入给出一段非空文本，最后以符号#结尾。输入保证存在至少10个不同的单词。

### 输出格式:

在第一行中输出文本中所有不同单词的个数。注意“单词”不区分英文大小写，例如“PAT”和“pat”被认为是同一个单词。

随后按照词频递减的顺序，按照词频:单词的格式输出词频最大的前10%的单词。若有并列，则按递增字典序输出。

### 输入样例：

This is a test.

The word "this" is the word with the highest frequency.

Longlonglonglongword should be cut off, so is considered as the same as longlonglonglonee. But this\_8 is different than this, and this, and this...#

this line should be ignored.

### 输出样例：（注意：虽然单词the也出现了4次，但因为我们只要输出前10%（即23个单词中的前2个）单词，而按照字母序，the排第3位，所以不输出。）

23

5:this

4:is

### 感谢武汉理工大学的郭小兵老师修正测试数据！

# 练习6.1 邻接矩阵存储图的深度优先遍历（20 分）

试实现邻接矩阵存储图的深度优先遍历。

### 函数接口定义：

void DFS( MGraph Graph, Vertex V, void (\*Visit)(Vertex) );

其中MGraph是邻接矩阵存储的图，定义如下：

typedef struct GNode \*PtrToGNode;

struct GNode{

int Nv; /\* 顶点数 \*/

int Ne; /\* 边数 \*/

WeightType G[MaxVertexNum][MaxVertexNum]; /\* 邻接矩阵 \*/

};

typedef PtrToGNode MGraph; /\* 以邻接矩阵存储的图类型 \*/

函数DFS应从第V个顶点出发递归地深度优先遍历图Graph，遍历时用裁判定义的函数Visit访问每个顶点。当访问邻接点时，要求按序号递增的顺序。题目保证V是图中的合法顶点。

### 裁判测试程序样例：

#include <stdio.h>

typedef enum {false, true} bool;

#define MaxVertexNum 10 /\* 最大顶点数设为10 \*/

#define INFINITY 65535 /\* ∞设为双字节无符号整数的最大值65535\*/

typedef int Vertex; /\* 用顶点下标表示顶点,为整型 \*/

typedef int WeightType; /\* 边的权值设为整型 \*/

typedef struct GNode \*PtrToGNode;

struct GNode{

int Nv; /\* 顶点数 \*/

int Ne; /\* 边数 \*/

WeightType G[MaxVertexNum][MaxVertexNum]; /\* 邻接矩阵 \*/

};

typedef PtrToGNode MGraph; /\* 以邻接矩阵存储的图类型 \*/

bool Visited[MaxVertexNum]; /\* 顶点的访问标记 \*/

MGraph CreateGraph(); /\* 创建图并且将Visited初始化为false；裁判实现，细节不表 \*/

void Visit( Vertex V )

{

printf(" %d", V);

}

void DFS( MGraph Graph, Vertex V, void (\*Visit)(Vertex) );

int main()

{

MGraph G;

Vertex V;

G = CreateGraph();

scanf("%d", &V);

printf("DFS from %d:", V);

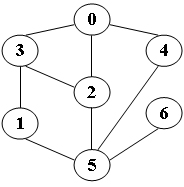
DFS(G, V, Visit);

return 0;

}

/\* 你的代码将被嵌在这里 \*/

### 输入样例：给定图如下



5

### 输出样例：

DFS from 5: 5 1 3 0 2 4 6

注：在题目中建立图的时候，如果V和W之间有边的话Graph->G[V][W]会被初始化为1

# 练习6.2 邻接表存储图的广度优先遍历（20 分）

试实现邻接表存储图的广度优先遍历。

### 函数接口定义：

void BFS ( LGraph Graph, Vertex S, void (\*Visit)(Vertex) );

其中LGraph是邻接表存储的图，定义如下：

/\* 邻接点的定义 \*/

typedef struct AdjVNode \*PtrToAdjVNode;

struct AdjVNode{

Vertex AdjV; /\* 邻接点下标 \*/

PtrToAdjVNode Next; /\* 指向下一个邻接点的指针 \*/

};

/\* 顶点表头结点的定义 \*/

typedef struct Vnode{

PtrToAdjVNode FirstEdge; /\* 边表头指针 \*/

} AdjList[MaxVertexNum]; /\* AdjList是邻接表类型 \*/

/\* 图结点的定义 \*/

typedef struct GNode \*PtrToGNode;

struct GNode{

int Nv; /\* 顶点数 \*/

int Ne; /\* 边数 \*/

AdjList G; /\* 邻接表 \*/

};

typedef PtrToGNode LGraph; /\* 以邻接表方式存储的图类型 \*/

函数BFS应从第S个顶点出发对邻接表存储的图Graph进行广度优先搜索，遍历时用裁判定义的函数Visit访问每个顶点。当访问邻接点时，要求按邻接表顺序访问。题目保证S是图中的合法顶点。

### 裁判测试程序样例：

#include <stdio.h>

typedef enum {false, true} bool;

#define MaxVertexNum 10 /\* 最大顶点数设为10 \*/

typedef int Vertex; /\* 用顶点下标表示顶点,为整型 \*/

/\* 邻接点的定义 \*/

typedef struct AdjVNode \*PtrToAdjVNode;

struct AdjVNode{

Vertex AdjV; /\* 邻接点下标 \*/

PtrToAdjVNode Next; /\* 指向下一个邻接点的指针 \*/

};

/\* 顶点表头结点的定义 \*/

typedef struct Vnode{

PtrToAdjVNode FirstEdge; /\* 边表头指针 \*/

} AdjList[MaxVertexNum]; /\* AdjList是邻接表类型 \*/

/\* 图结点的定义 \*/

typedef struct GNode \*PtrToGNode;

struct GNode{

int Nv; /\* 顶点数 \*/

int Ne; /\* 边数 \*/

AdjList G; /\* 邻接表 \*/

};

typedef PtrToGNode LGraph; /\* 以邻接表方式存储的图类型 \*/

bool Visited[MaxVertexNum]; /\* 顶点的访问标记 \*/

LGraph CreateGraph(); /\* 创建图并且将Visited初始化为false；裁判实现，细节不表 \*/

void Visit( Vertex V )

{

printf(" %d", V);

}

void BFS ( LGraph Graph, Vertex S, void (\*Visit)(Vertex) );

int main()

{

LGraph G;

Vertex S;

G = CreateGraph();

scanf("%d", &S);

printf("BFS from %d:", S);

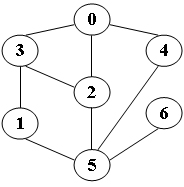
BFS(G, S, Visit);

return 0;

}

/\* 你的代码将被嵌在这里 \*/

### 输入样例：给定图如下



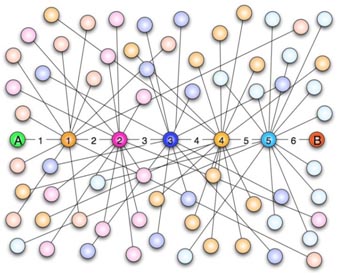
2

### 输出样例：

BFS from 2: 2 0 3 5 4 1 6

# 实例6.1 六度空间（30 分）

“六度空间”理论又称作“六度分隔（Six Degrees of Separation）”理论。这个理论可以通俗地阐述为：“你和任何一个陌生人之间所间隔的人不会超过六个，也就是说，最多通过五个人你就能够认识任何一个陌生人。”如图1所示。

  
图1 六度空间示意图

“六度空间”理论虽然得到广泛的认同，并且正在得到越来越多的应用。但是数十年来，试图验证这个理论始终是许多社会学家努力追求的目标。然而由于历史的原因，这样的研究具有太大的局限性和困难。随着当代人的联络主要依赖于电话、短信、微信以及因特网上即时通信等工具，能够体现社交网络关系的一手数据已经逐渐使得“六度空间”理论的验证成为可能。

假如给你一个社交网络图，请你对每个节点计算符合“六度空间”理论的结点占结点总数的百分比。

### 输入格式:

输入第1行给出两个正整数，分别表示社交网络图的结点数*N*（1<*N*≤10​4​​，表示人数）、边数*M*（≤33×*N*，表示社交关系数）。随后的*M*行对应*M*条边，每行给出一对正整数，分别是该条边直接连通的两个结点的编号（节点从1到*N*编号）。

### 输出格式:

对每个结点输出与该结点距离不超过6的结点数占结点总数的百分比，精确到小数点后2位。每个结节点输出一行，格式为“结点编号:（空格）百分比%”。

### 输入样例:

10 9

1 2

2 3

3 4

4 5

5 6

6 7

7 8

8 9

9 10

### 输出样例:

1: 70.00%

2: 80.00%

3: 90.00%

4: 100.00%

5: 100.00%

6: 100.00%

7: 100.00%

8: 90.00%

9: 80.00%

10: 70.00%

# 练习7.1 排序（25 分）

给定*N*个（长整型范围内的）整数，要求输出从小到大排序后的结果。

本题旨在测试各种不同的排序算法在各种数据情况下的表现。各组测试数据特点如下：

 数据1：只有1个元素；

 数据2：11个不相同的整数，测试基本正确性；

 数据3：103个随机整数；

 数据4：104个随机整数；

 数据5：105个随机整数；

 数据6：105个顺序整数；

 数据7：105个逆序整数；

 数据8：105个基本有序的整数；

 数据9：105个随机正整数，每个数字不超过1000。

### 输入格式:

输入第一行给出正整数*N*（≤10​5​​），随后一行给出*N*个（长整型范围内的）整数，其间以空格分隔。

### 输出格式:

在一行中输出从小到大排序后的结果，数字间以1个空格分隔，行末不得有多余空格。

### 输入样例:

11

4 981 10 -17 0 -20 29 50 8 43 -5

### 输出样例:

-20 -17 -5 0 4 8 10 29 43 50 981

# 习题8.1 银行排队问题之单队列多窗口服务（25 分）

假设银行有*K*个窗口提供服务，窗口前设一条黄线，所有顾客按到达时间在黄线后排成一条长龙。当有窗口空闲时，下一位顾客即去该窗口处理事务。当有多个窗口可选择时，假设顾客总是选择编号最小的窗口。

本题要求输出前来等待服务的*N*位顾客的平均等待时间、最长等待时间、最后完成时间，并且统计每个窗口服务了多少名顾客。

### 输入格式:

输入第1行给出正整数*N*（≤1000），为顾客总人数；随后*N*行，每行给出一位顾客的到达时间T和事务处理时间P，并且假设输入数据已经按到达时间先后排好了顺序；最后一行给出正整数*K*（≤10），为开设的营业窗口数。这里假设每位顾客事务被处理的最长时间为60分钟（意思是如果P大于60直接让P等于60即可，即窗口肯定能提高效率在60分钟内把你的任务处理完）。

### 输出格式:

在第一行中输出平均等待时间（输出到小数点后1位）、最长等待时间、最后完成时间，之间用1个空格分隔，行末不能有多余空格。

在第二行中按编号递增顺序输出每个窗口服务了多少名顾客，数字之间用1个空格分隔，行末不能有多余空格。

### 输入样例：

9

0 20

1 15

1 61

2 10

10 5

10 3

30 18

31 25

31 2

3

### 输出样例：

6.2 17 61

5 3 1

### 感谢浙江财经大学王瑞洲、周甄陶同学修正测试数据！

# 习题8.2 银行排队问题之单队列多窗口加VIP服务（30 分）

假设银行有*K*个窗口提供服务，窗口前设一条黄线，所有顾客按到达时间在黄线后排成一条长龙。当有窗口空闲时，下一位顾客即去该窗口处理事务。当有多个窗口可选择时，假设顾客总是选择编号最小的窗口。

有些银行会给VIP客户以各种优惠服务，例如专门开辟VIP窗口。为了最大限度地利用资源，VIP窗口的服务机制定义为：当队列中没有VIP客户时，该窗口为普通顾客服务；当该窗口空闲并且队列中有VIP客户在等待时，排在最前面的VIP客户享受该窗口的服务。同时，当轮到某VIP客户出列时，若VIP窗口非空，该客户可以选择空闲的普通窗口；**否则一定选择VIP窗口**。

本题要求输出前来等待服务的*N*位顾客的平均等待时间、最长等待时间、最后完成时间，并且统计每个窗口服务了多少名顾客。

### 输入格式:

输入第1行给出正整数*N*（≤1000），为顾客总人数；随后*N*行，每行给出一位顾客的到达时间T、事务处理时间P和是否VIP的标志（1是VIP，0则不是），并且假设输入数据已经按到达时间先后排好了顺序；最后一行给出正整数*K*（≤10）—— 为开设的营业窗口数，以及VIP窗口的编号（从0到*K*−1）。这里假设每位顾客事务被处理的最长时间为60分钟。

### 输出格式:

在第一行中输出平均等待时间（输出到小数点后1位）、最长等待时间、最后完成时间，之间用1个空格分隔，行末不能有多余空格。

在第二行中按编号递增顺序输出每个窗口服务了多少名顾客，数字之间用1个空格分隔，行末不能有多余空格。

### 输入样例：

10

0 20 0

0 20 0

1 68 1

1 12 1

2 15 0

2 10 0

3 15 1

10 12 1

30 15 0

62 5 1

3 1

### 输出样例：

15.1 35 67

4 5 1

# 习题8.3 银行排队问题之单窗口“夹塞”版（30 分）

排队“夹塞”是引起大家强烈不满的行为，但是这种现象时常存在。在银行的单窗口排队问题中，假设银行只有1个窗口提供服务，所有顾客按到达时间排成一条长龙。当窗口空闲时，下一位顾客即去该窗口处理事务。此时如果已知第*i*位顾客与排在后面的第*j*位顾客是好朋友，并且愿意替朋友办理事务的话，那么第*i*位顾客的事务处理时间就是自己的事务加朋友的事务所耗时间的总和。在这种情况下，顾客的等待时间就可能被影响。假设所有人到达银行时，若没有空窗口，都会请求排在最前面的朋友帮忙（包括正在窗口接受服务的朋友）；当有不止一位朋友请求某位顾客帮忙时，该顾客会根据自己朋友请求的顺序来依次处理事务。试编写程序模拟这种现象，并计算顾客的平均等待时间。

### 输入格式:

输入的第一行是两个整数：1≤*N*≤10000，为顾客总数；0≤*M*≤100，为彼此不相交的朋友圈子个数。若*M*非0，则此后*M*行，每行先给出正整数2≤*L*≤100，代表该圈子里朋友的总数，随后给出该朋友圈里的*L*位朋友的名字。名字由3个大写英文字母组成，名字间用1个空格分隔。最后*N*行给出*N*位顾客的姓名、到达时间*T*和事务处理时间*P*（以分钟为单位），之间用1个空格分隔。简单起见，这里假设顾客信息是按照到达时间先后顺序给出的（有并列时间的按照给出顺序排队），并且假设每个事务最多占用窗口服务60分钟（如果超过则按60分钟计算）。

### 输出格式:

按顾客接受服务的顺序输出顾客名字，每个名字占1行。最后一行输出所有顾客的平均等待时间，保留到小数点后1位。

### 输入样例:

6 2

3 ANN BOB JOE

2 JIM ZOE

JIM 0 20

BOB 0 15

ANN 0 30

AMY 0 2

ZOE 1 61

JOE 3 10

### 输出样例:

JIM

ZOE

BOB

ANN

JOE

AMY

75.2

# 习题8.4 畅通工程之最低成本建设问题（30 分）

某地区经过对城镇交通状况的调查，得到现有城镇间快速道路的统计数据，并提出“畅通工程”的目标：使整个地区任何两个城镇间都可以实现快速交通（但不一定有直接的快速道路相连，只要互相间接通过快速路可达即可）。现得到城镇道路统计表，表中列出了有可能建设成快速路的若干条道路的成本，求畅通工程需要的最低成本。

### 输入格式:

输入的第一行给出城镇数目*N* (1<*N*≤1000)和候选道路数目*M*≤3*N*；随后的*M*行，每行给出3个正整数，分别是该条道路直接连通的两个城镇的编号（从1编号到*N*）以及该道路改建的预算成本。

### 输出格式:

输出畅通工程需要的最低成本。如果输入数据不足以保证畅通，则输出“Impossible”。

### 输入样例1:

6 15

1 2 5

1 3 3

1 4 7

1 5 4

1 6 2

2 3 4

2 4 6

2 5 2

2 6 6

3 4 6

3 5 1

3 6 1

4 5 10

4 6 8

5 6 3

### 输出样例1:

12

### 输入样例2:

5 4

1 2 1

2 3 2

3 1 3

4 5 4

### 输出样例2:

Impossible

# 习题8.5 畅通工程之局部最小花费问题（35 分）

某地区经过对城镇交通状况的调查，得到现有城镇间快速道路的统计数据，并提出“畅通工程”的目标：使整个地区任何两个城镇间都可以实现快速交通（但不一定有直接的快速道路相连，只要互相间接通过快速路可达即可）。现得到城镇道路统计表，表中列出了任意两城镇间修建快速路的费用，以及该道路是否已经修通的状态。现请你编写程序，计算出全地区畅通需要的最低成本。

### 输入格式:

输入的第一行给出村庄数目*N* (1≤*N*≤100)；随后的*N*(*N*−1)/2行对应村庄间道路的成本及修建状态：每行给出4个正整数，分别是两个村庄的编号（从1编号到*N*），此两村庄间道路的成本，以及修建状态 — 1表示已建，0表示未建。

### 输出格式:

输出全省畅通需要的最低成本。

### 输入样例:

4

1 2 1 1

1 3 4 0

1 4 1 1

2 3 3 0

2 4 2 1

3 4 5 0

### 输出样例:

3