PAT B Level及其他代码

求两数最大公约数和最小公倍数

#include<stdio.h>

#include<math.h>

int yueshu(int a,int b)

{

int i,c;

for(i=1;i<a&&i<b;i++)

if (a%i == 0)

{

if (b%i == 0)

c = i;

};

return c;

}

int beishu(int a,int b)

{

int i, c;

for (i = a\*b; i >= a&&i >= b; i--)

{

if (i%a == 0)

{

if (i%b == 0)

c = i;

};

};

return c;

}

void main(void)

{

int a, b,x,y;

scanf("%d,%d",&a, &b);

x=beishu(a,b);

y=yueshu(a,b);

printf("最大公倍数为%d\n最小公约数为%d\n", x, y);

}

goto函数的用法

#include<stdio.h>

void main()

{

int a, b, c, d;

begin: scanf ("%d,%d,%d,%d", &a, &b, &c, &d);

if (a > b&&c > d)

goto begin;

}

getch()以及Sleep的用法

#include<stdio.h>

#include<iostream>

#include<windows.h>

#include<conio.h>

void main()

{

int c;

while (1)

{

Sleep(500);

printf("yeah");

c = getch();

if (c == 'n' || c == 'N')

continue;

else break;

}

}

int kbhit(void)函数检查当前是否有键盘键入，若有则返回一个非0值，若没有则返回0

!kbhit误输入返回1，有输入返回0，以下为测试函数。

#include<stdio.h>

#include<iostream>

#include<windows.h>

#include<conio.h>

void main()

{

int c,a;

while (1)

{

getch();

Sleep(500);

a = !kbhit();

printf("yeah\n%d\n",a);

}

}

int getch()的用法测试

#include<stdio.h>

#include<iostream>

#include<windows.h>

#include<conio.h>

void main(void)

{

char c;

while (1)

{

c = getch();

Sleep(1);

printf("%c", c);

}

}

排序（冒泡排序）

#include<stdio.h>

#include<iostream>

#include<windows.h>

#include<conio.h>

int main()

{

int A[5], i, j, a, b;

for (i = 0; i < 5; i++)

scanf("%d",&A[i]);

for (i = 0; i < 5; i++)

{

for (j = i; j < 5; j++)

{

if (A[i] <= A[j])

{ //就特么忘了这个括号

b = A[i];

A[i] = A[j];

A[j] = b;

}

}

}

for (i = 0; i < 5; i++)

printf("%d ",A[i]);

return 0;

}

/\*某城市最高的楼只有一部电梯。

该电梯和一般电梯不同的是它依照输入楼层数的先后次序运行。

电梯最初在0层。

运行完一个输入序列后就停止在该楼层，不返回0层。

编写程序计算电梯运行一个序列的时间。

每次都假设电梯在0层开始，无论上一次运行到几层。

电梯每上1层需要6秒。每下1层需要4秒。

如在某层停留，无论上下人多少，均停留5秒。

输入：程序可以输入多行。每行的第一个数N指示后面将有N个楼层的序列。楼层的值大于等于1，小于100。N = 0表示结束输入。

输出：计算每个序列电梯运行的时间。

例子：

1 2

3 2 3 1

0

17

41\*/

#include<stdio.h>

#include<iostream>

using namespace std;

int a, b=0, m[10], N=1,i=0,j;//a表示

int jisuan()

{

m[10] = 0;

while (1)

{

cin >> N;

if (N == 0)

break;

for (j = 0; j < N; j++)

{

cin >> a;

if (b > a)

m[i] = m[i] + 4 \* (b - a);

else if (b < a)

m[i] = m[i] + 6 \* (a - b);

b = a;

}

b = 0;

m[i] += (5 \* N);

i++;

}

return 0;

}

int main()

{

jisuan();

for (j = 0; j < i; j++)

cout << m[j] << endl;

return 0;

}

/\*猜数字游戏是gameboy最喜欢的游戏之一。游戏的规则是这样的：计算机随机产生一个四位数，然后玩家猜这个四位数是什么。

每猜一个数，计算机都会告诉玩家猜对几个数字，其中有几个数字在正确的位置上。比如计算机随机产生的数字为1122。

如果玩家猜1234, 因为1, 2这两个数字同时存在于这两个数中，而且1在这两个数中的位置是相同的，所以计算机会告诉玩

家猜对了2个数字，其中一个在正确的位置。如果玩家猜1111, 那么计算机会告诉他猜对2个数字，有2个在正确的位置。

现在给你一段gameboy与计算机的对话过程，你的任务是根据这段对话确定这个四位数是什么。 输入：输入数据有多组。

每组的第一行为一个正整数N(1 <= N <= 100)，表示在这段对话中共有N次问答。在接下来的N行中，每行三个整数A, B, C。

gameboy猜这个四位数为A，然后计算机回答猜对了B个数字，其中C个在正确的位置上。当N = 0时，输入数据结束。

输出：每组输入数据对应一行输出。如果根据这段对话能确定这个四位数，则输出这个四位数，若不能，则输出"Not sure"。

例子：

6

4815 2 1

5716 1 0

7842 1 0

4901 0 0

8585 3 3

8555 3 2

2

4815 0 0

2999 3 3

0

3585

Not sure\*/

#include<stdio.h>

#include<iostream>

using namespace std;

struct G { int a; int b; int c; };//使用结构数组

int judge(int i, int A, int B, int C)

{

int Q[4], W[4], x[4], y[4], p, q, k = 0, l = 0;

Q[0] = i % 10; //Q存放的是待判定的数

Q[1] = (i / 10) % 10;

Q[2] = (i / 100) % 10;

Q[3] = (i / 1000);

W[0] = A % 10; //W存放的是输入的数

W[1] = (A / 10) % 10;

W[2] = (A / 100) % 10;

W[3] = (A / 1000);

for (p = 0; p < 4; p++)

{

if (Q[p] == W[p])

k++;

}

for (p = 0; p < 4; p++) //如此机智的算法

{

for (q = 0; q < 4; q++)

{

if (Q[p] == W[q])

{

l++;

Q[p] = -1;

W[q] = -2;

}

}

}

if (k == C&&l == B)

return 1;

else

return 0;

}

int main()

{

struct G X[99];

int N, m = 1, n, i, j, k,u;

while(1)

{

cin >> N;

n = 0;

struct G X[99];

for (i = 0; i < N; i++)

{

cin >> X[i].a >> X[i].b >> X[i].c;

}

for (j = 1000; j <= 9999; j++)

{

m = 1;

for (i = 0; i < N; i++) //他奶奶的，多打了个分号，草泥马

{

m = m\*judge(j, X[i].a, X[i].b, X[i].c);//这里，只有每次执行都能通过，才能最终通过

}

if (m == 1) //若通过了，则说明这个数符合整个N的要求

{

n++; //计有多少个符合整个N要求的数

k = j; //把这个数赋值给k

}

}

if (n == 1) //如果只有一个符合的数，则输出

{

cout << k << endl;

}

else

{

cout << "Not sure" << endl;

}

}

} //没有完全符合题目要求，可以改进

/\*进行一次独木舟的旅行活动，独木舟可以在港口租到，并且之间没有区别。一条独木舟最多只能乘坐两人，且乘客的

总重量不能超过独木舟的最大的承载量。我们要尽可能的减少这次活动中的花销，所以要找出可以安置所有旅客的最少

的独木舟数。现在请写出程序，读入独木舟的最大承载量、旅客数目和每位旅客的重量。根据给出的规则计算安置所有

旅客必须的最少的独木舟条数。输入2个整数w和n，80 <= w <= 200, 1 <= n <= 300，w表示一条独木舟的最大承载量，

n为人数。接下来输入每个人的重量（不能大于承载量）输出独木舟的条数，例如：

85 6

5 84 85 80 84 83

结果为5\*/

#include<iostream>

#include<stdio.h>

using namespace std;

int main()

{

int w, n, x[300];

while (1)

{

cin >> w;

cin >> n;

int i, j, a;

for (i = 0; i < n; i++)

{

cin >> x[i];

}

for (i = 0; i < n; i++) //将乘客重量从小到大进行排序

{

for (j = i; j < n; j++)

{

if (x[i] < x[j])

{

a = x[i];

x[i] = x[j];

x[j] = a;

}

}

}

int M=0, N=n-1, l=0;

while (M < N)

{

i = x[M] + x[N]; //M从第一个，N从最后一个进行相加比较

if (i > w) //若大于每个船的载重，则重人一个人一个船

{

M++;

l++;

}

else //或者两个人一个船

{

M++;

N--;

l++;

}

}

cout << l << endl;

}

return 0;

}

/\*(HDU 1003)给你一个序列a[1],a[2],a[3]......a[n],计算其最大和子序列。比如(6,-1,5,4,-7),它的最大和子串为(6,-1,5,4)

其中6 + (-1) + 5 + 4 = 14.注意这里是子串，在序列中是连续的元素。

输入：

首先输入一个正整数n，代表序列的长度为n，其中1<=n<=100000。另起一行，输入n个整数，作为序列,每个整数

大于等于-1000小于等于1000。

输出：

输出一行，包括三个数字，分别为：子串的最大和，子串在序列中的起始位置，子串在序列中的终止位置。

样例输入：

5

6 -1 5 4 -7

样例输出：

14 1 4

\*/

#include<iostream> //方法1，单次循环

using namespace std;

int main()

{

int i, j=1, x[100000], n, k;

while (1)

{

cin >> n;

for (i = 0; i < n; i++) //键入数据

{

cin >> x[i];

}

int thissum = 0, maxsum = 0;

for (i = 0; i < n; i++)

{

thissum += x[i]; //每次录入一个，加起来与maxsum比较

if (thissum > maxsum)

{

maxsum = thissum;

k = i + 1;

}

if (thissum < 0) //thissum小于0直接舍弃

{

thissum = 0;

j = i + 2;

}

}

cout << maxsum <<" "<<j<<" "<<k<< endl;

}

}

#include<iostream> //方法2，在线处理

using namespace std;

int main()

{

int i, ca = 1, s, e, n, x, now, before, max;

cin >> n;

for (i = 1; i <= n; i++)

{

cin >> now;

if (i == 1)//初始化

{

max = before = now;//max保留之前算出来的最大和，before存储目前在读入数据前保留的和，now保留读入数据

x = s = e = 1;//x用来暂时存储before保留的和的起始位置，当before>max时将赋在s位置，s，e保留最大和的start和end位置

}

else

{

if (now > now + before)//如果之前存储的和加上现在的数据比现在的数据小，就把存储的和换成现在的数据，反之就说明数据在递增，可以直接加上

{

before = now;

x = i;//预存的位置要重置

}

else

before += now;

}

if (before>max)//跟之前算出来的最大和进行比较，如果大于，位置和数据就要重置

max = before, s = x, e = i;

}

cout << max << ' ' << s << ' ' << e << endl;

return 0;

}

/\*(HDU2602)许多年前，有一个被称为“骨收藏家”的人。这人喜欢收集不同的骨头，如狗，牛的，当然他也去坟墓…

集骨者有一个大袋子，体积为V，他此行遇到N个骨头，不同的骨具有不同的价值和不同的体积，现在给出每个骨头的价值和体积，

你能计算出骨收藏家此行能收集到的骨头的最大的总价值吗？

输入:

输入包含三行，首先第一行包括两个数字，第一个数字N，代表骨头数，第二个数字V代表袋子的容量。第二行输入N个数字，代表

N个骨头的价值。第三行输入N个数字，代表N个骨头的体积。

输出:

可以收集的骨头的最大价值。

样例输入:

5 10

1 2 3 4 5

5 4 3 2 1

样例输出:

14

\*/

#include<iostream>

using namespace std;

struct G { int a; int b; }; //结构数组

int main()

{

struct G x[100];

int i, j, m, n, N, V; //第一个数字N，代表骨头数，第二个数字V代表袋子的容量

cin >> N >> V;

for (i = 0; i < N; i++)

{

cin >> x[i].a; //输入价值

}

for (i = 0; i < N; i++)

{

cin >> x[i].b; //输入体积

}

for (i = 0; i < N; i++) //按体积从大到小排序

{

for (j = i+1; j < N; j++)

{

if (x[i].b < x[j].b)

{

m = x[i].a;

n = x[i].b;

x[i].a = x[j].a;

x[i].b = x[j].b;

x[j].a = m;

x[j].b = n;

}

}

}

int thissum, maxsum=0, sum;

m = 0;

n = N - 1;

while (m < n) //完成的意外的顺利！

{

n = N - 1; //n指向体积最小的一个

sum = x[m].b ;

while (sum < V)

{

n--;

sum += x[n].b;

}

j = (N - 1) - n; //判断所有情况，符合的小体积的个数j

thissum = x[m].a;

for (i = 1; i <= j; i++)

{

thissum += x[N - i].a;//累加求总价值，放在thissum里

}

if (thissum > maxsum)

{

maxsum = thissum;

}

m++;

}

cout << maxsum << endl;

}

/\*让我们用字母B来表示“百”、字母S表示“十”，用“12...n”来表示个位数字n（<10），换个格式来输出任一个不超过3位的正整数。

例如234应该被输出为BBSSS1234，因为它有2个“百”、3个“十”、以及个位的4。

输入格式：每个测试输入包含1个测试用例，给出正整数n（<1000）。

输出格式：每个测试用例的输出占一行，用规定的格式输出n。

输入样例1：

234

输出样例1：

BBSSS1234

输入样例2：

23

输出样例2：

SS123\*/

#include<stdio.h>

#include<iostream>

using namespace std;

int main()

{

int a, b, c, i,j;

cin >> a;

b = (a / 100) % 10;

c = (a / 10) % 10;

i = a % 10;

for (j = 0; j < b; j++)

cout << 'B';

for (j = 0; j < c; j++)

cout << 'S';

for (j = 1; j <= i; j++)

cout << j;

return 0;

}

/\*让我们定义 dn 为：dn = pn+1 - pn，其中 pi 是第i个素数。显然有 d1=1 且对于n>1有 dn 是偶数。

“素数对猜想”认为“存在无穷多对相邻且差为2的素数”。

现给定任意正整数N (< 105)，请计算不超过N的满足猜想的素数对的个数。

输入格式：每个测试输入包含1个测试用例，给出正整数N。

输出格式：每个测试用例的输出占一行，不超过N的满足猜想的素数对的个数。

输入样例：

20

输出样例：

4\*/

#include <stdio.h>

#include <iostream>

#include <math.h>

using namespace std;

int main()

{

int a, c=0,b=0, i, j, fun[10000], k=0; //魔性的b=0???

cin >> a;

for (i = 2; i <= a; i++)

{

for (j = 2; j <= sqrt(i); j++)

{

b = 0;

if (i%j == 0)

{

b = 1;

break;

}

}

if (b == 0)

{

fun[k] = i;

if (fun[k] - fun[k - 1] == 2)

c++;

k++;

}

}

cout << c;

return 0;

}

/\*一个数组A中存有N（N>0）个整数，在不允许使用另外数组的前提下，将每个整数循环向右移M（M>=0）个位置，

即将A中的数据由（A0 A1……AN-1）变换为（AN-M …… AN-1 A0 A1……AN-M-1）（最后M个数循环移至最前面的M个位置）。

如果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：每个输入包含一个测试用例，第1行输入N ( 1<=N<=100)、M（M>=0）；第2行输入N个整数，之间用空格分隔。

输出格式：在一行中输出循环右移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：

6 2

1 2 3 4 5 6

输出样例：

5 6 1 2 3 4\*/

#include <stdio.h>

#include <iostream>

#include <math.h>

using namespace std;

int main()

{

int N, M, i, j, a[1000], b[1000];

cin >> N >> M;

for (i = 0; i < N; i++)

{

cin >> a[i];

}

for (j = 0; j < M; j++)

{

b[N - 1 - j] = a[N - 1 - j];

}

for (i = N - M - 1; i >= 0; i--)

{

a[i + M] = a[i];

}

for (i = 0; i < M; i++)

{

a[i] = b[N - M + i];

}

for (i = 0; i < N - 1; i++)

{

cout << a[i] << ' ';

}

cout << a[N - 1];

return 0;

}

/\*给定一句英语，要求你编写程序，将句中所有单词的顺序颠倒输出。

输入格式：测试输入包含一个测试用例，在一行内给出总长度不超过80的字符串。字符串由若干单词和若干空格组成，

其中单词是由英文字母（大小写有区分）组成的字符串，单词之间用1个空格分开，输入保证句子末尾没有多余的空格。

输出格式：每个测试用例的输出占一行，输出倒序后的句子。

输入样例：

Hello World Here I Come

输出样例：

Come I Here World Hello\*/

#include <stdio.h>

#include <iostream>

#include <math.h>

#define maxsize 50

using namespace std;

int main()

{

int i=0, j=0;

char a[80][80],b=NULL;

while(b!='\n')

{

cin >> a[j++];

b=getchar(); //getchar()的用法

i++;

}

for (i = j - 1; i >= 0; i--)

{

if (i == j - 1)

cout << a[i];

else

cout << ' '<<a[i];

}

return 0;

}

/\*小易准备去魔法王国采购魔法神器, 购买魔法神器需要使用魔法币,

但是小易现在一枚魔法币都没有, 但是小易有两台魔法机器可以通过投入x(x可以为0)个魔法币产生更多的魔法币。

魔法机器1 : 如果投入x个魔法币, 魔法机器会将其变为2x + 1个魔法币

魔法机器2 : 如果投入x个魔法币, 魔法机器会将其变为2x + 2个魔法币

小易采购魔法神器总共需要n个魔法币, 所以小易只能通过两台魔法机器产生恰好n个魔法币,

小易需要你帮他设计一个投入方案使他最后恰好拥有n个魔法币。\*/

#include <stdio.h>

#include <iostream>

#include <math.h>

#define maxsize 50

using namespace std;

int main()

{

int i=0, j, a[50], b, n;

cin >> n;

while (n>=1)

{

if (n % 2 == 0)

{

n = (n - 2) / 2;

a[i]=2;

i++;

}

else

{

n = (n - 1) / 2;

a[i] = 1;

i++;

}

}

for (j = i - 1; j >= 0; j--)

cout << a[j];

return 0;

}

/\*设计函数求一元多项式的导数。（注：xn（n为整数）的一阶导数为n\*xn-1。）

输入格式：以指数递降方式输入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：以与输入相同的格式输出导数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。

注意“零多项式”的指数和系数都是0，但是表示为“0 0”。

输入样例：

3 4 -5 2 6 1 -2 0

输出样例：

12 3 -10 1 6 0\*/

#include <stdio.h>

#include <iostream>

#include <math.h>

#define maxsize 100

using namespace std;

int main()

{

int x[maxsize][maxsize], i=0, j=0, n=0, b=0;

while (b != '\n')

{

cin >> x[0][i];

i++;

b = getchar(); //getchar()的用法

}

for (j = 0; j < i; j+=2)

{

if (x[0][j] != 0 && x[0][j + 1] != 0)

{

x[1][n] = x[0][j] \* (x[0][j + 1] );

x[1][n + 1] = x[0][j + 1] - 1;

n += 2;

}

}

for (i = 0; i < n; i++)

{

if (i == 0)

{

cout << x[1][i];

}

else

cout <<' '<< x[1][i];

}

return 0;

}

/\*给定区间[-231, 231]内的3个整数A、B和C，请判断A+B是否大于C。

输入格式：

输入第1行给出正整数T(<=10)，是测试用例的个数。随后给出T组测试用例，每组占一行，顺序给出A、B和C。整数间以空格分隔。

输出格式：

对每组测试用例，在一行中输出“Case #X: true”如果A+B>C，否则输出“Case #X: false”，其中X是测试用例的编号（从1开始）。

输入样例：

4

1 2 3

2 3 4

2147483647 0 2147483646

0 -2147483648 -2147483647

输出样例：

Case #1: false

Case #2: true

Case #3: true

Case #4: false\*/

#include <stdio.h>

#include <iostream>

#include <math.h>

#define maxsize 100

using namespace std;

typedef struct

{

long int a, b, c;

string m;

}G;

int main()

{

int i, j, n;

G x[maxsize];

cin >> n;

for (i = 0; i < n; i++)

{

cin >> x[i].a >> x[i].b >> x[i].c;

}

for (i = 0; i < n; i++)

{

if (i == n - 1)

{

if (x[i].a + x[i].b > x[i].c)

cout << "Case #" << i + 1 << ": " << "true" ;

else

cout << "Case #" << i + 1 << ": " << "false" ;

}

else

{

if (x[i].a + x[i].b > x[i].c)

cout << "Case #" << i + 1 << ": " << "true" << endl;

else

cout << "Case #" << i + 1 << ": " << "false" << endl;

}

}

return 0;

}

/\*！！给定一系列正整数，请按要求对数字进行分类，并输出以下5个数字：

A1 = 能被5整除的数字中所有偶数的和；

A2 = 将被5除后余1的数字按给出顺序进行交错求和，即计算n1-n2+n3-n4...；

A3 = 被5除后余2的数字的个数；

A4 = 被5除后余3的数字的平均数，精确到小数点后1位；

A5 = 被5除后余4的数字中最大数字。

输入格式：

每个输入包含1个测试用例。每个测试用例先给出一个不超过1000的正整数N，随后给出N个不超过1000的待分类的正整数。数字间以空格分隔。

输出格式：

对给定的N个正整数，按题目要求计算A1~A5并在一行中顺序输出。数字间以空格分隔，但行末不得有多余空格。

若其中某一类数字不存在，则在相应位置输出“N”。

输入样例1：

13 1 2 3 4 5 6 7 8 9 10 20 16 18

输出样例1：

30 11 2 9.7 9

输入样例2：

8 1 2 4 5 6 7 9 16

输出样例2：

N 11 2 N 9\*/

#include <stdio.h>

#include <iostream>

#include <math.h>

#define maxsize 100

using namespace std;

int main()

{

int x[1000], a = 0, b = 0, i = 0, j = 1, A[5] = { 0 },m;

double k, n = 0;

cin >> m;

for (i = 0; i < m; i++)

{

cin >> x[i];

}

while(a<m)

{

switch (x[a] % 5)

{

case 0:

{

if (x[a] % 2 == 0)

A[0] += x[a];

a++;

break;

}

case 1:

{

A[1] += j\*x[a];

j = j\*-1;

a++;

break;

}

case 2:

{

A[2]++;

a++;

break;

}

case 3:

{

A[3] += x[a];

a++;

n++;

break;

}

case 4:

{

if (x[a] > A[4])

A[4] = x[a];

a++;

break;

}

}

}

k = A[3] / n; //整型变量相除没有小数部分，整型除以浮点型变量为浮点型变量

if (n == 1)

A[3] = 0;

for (i = 0; i < 5; i++)

{

if (A[i] != 0 && i == 0)

cout << A[0];

else if (A[i] == 0 && i == 0)

cout << "N";

else if (A[i] != 0 && i == 3 && n>1)

{

cout << " ";

printf("%.1lf", k);//输出一位小数这样写

}

else if (A[i] != 0 && i != 0)

cout <<" "<< A[i];

else if (A[i] == 0 && i != 0)

cout << " "<<"N";

}

return 0;

}

/\*另一种指针变量与分函数的做法\*/

#include **<stdio.h>**

**int** N;

**void** A1(**int**,**int** \*);

**void** A2(**int**,**int** \*,**int** \*);

**void** A4(**int**,**int** \*);

**void** A5(**int**,**int** \*);

**int** main()

{

**int** i,j;

**int** arr[1000];

**int** a1,a2,a3,a4,a5;

**int** count1,count2,count4,count5;

a1=a2=a3=a4=a5=0;

count1=count2=count4=count5=0;

scanf("%d", &N);

**for** (i = 0; i < N; i++)

{

scanf("%d",&arr[i]);

}

**for** (i = 0; i < N; i++)

{

j = arr[i]%5+1;

**switch** (j)

{

**case** 1: a1++;A1(arr[i],&count1);**break**;

**case** 2: a2++;A2(arr[i],&count2,&a2);**break**;

**case** 3: a3++;**break**;

**case** 4: a4++;A4(arr[i],&count4);**break**;

**case** 5: a5++;A5(arr[i],&count5);**break**;

}

}

**if**(a1!=0&&count1!=0)

printf("%d",count1);

**else**

printf("N");

**if**(a2!=0)

printf(" %d",count2);

**else**

printf(" N");

**if**(a3!=0)

printf(" %d",a3);

**else**

printf(" N");

**if**(a4!=0)

printf(" %.1lf",(**double**)count4/a4);

**else**

printf(" N");

**if**(a5!=0)

printf(" %d",count5);

**else**

printf(" N");

**return** 0;

}

**void** A1(**int** a,**int** \*count)

{

**if** ((a&1) == 0)

\*count += a;

}

**void** A2(**int** a,**int** \*count,**int** \*flag)

{

**if** (((\*flag)&1)==0)

(\*count) -= a;

**else**

(\*count) += a;

}

**void** A4(**int** a,**int** \*count)

{

\*count += a;

}

**void** A5(**int** a,**int** \*max)

{

**if**(\*max<a)

\*max=a;

}

/\*令Pi表示第i个素数。现任给两个正整数M <= N <= 104，请输出PM到PN的所有素数。

输入格式：

输入在一行中给出M和N，其间以空格分隔。

输出格式：

输出从PM到PN的所有素数，每10个数字占1行，其间以空格分隔，但行末不得有多余空格。

输入样例：

5 27

输出样例：

11 13 17 19 23 29 31 37 41 43

47 53 59 61 67 71 73 79 83 89

97 101 103\*/

#include <stdio.h>

#include <iostream>

#include <math.h>

#define maxsize 100

using namespace std;

int main()

{

int fun[10000], a, b, i, j, m, n, k=1, l=0, o, p;

cin >> m >> n;

for (i = 2; ; i++)

{

o = 1;

for (j = 2; j <= sqrt(i); j++)

{

if (k > n)

return 0;

if (i%j == 0)

o = 0;

}

if (o == 1)

{

fun[k] = i;

if (k >= m)

{

cout << fun[k];

l++;

if (l % 10 == 0)

cout << endl;

else if(k!=n)

cout << " ";

}

k++;

}

}

}

/\*大侦探福尔摩斯接到一张奇怪的字条：“我们约会吧！ 3485djDkxh4hhGE 2984akDfkkkkggEdsb s&hgsfdk d&Hyscvnm”。

大侦探很快就明白了，字条上奇怪的乱码实际上就是约会的时间“星期四 14:04”，因为前面两字符串中

第1对相同的大写英文字母（大小写有区分）是第4个字母'D'，代表星期四；

第2对相同的字符是'E'，那是第5个英文字母，代表一天里的第14个

钟头（于是一天的0点到23点由数字0到9、以及大写字母A到N表示）；

后面两字符串第1对相同的英文字母's'出现在第4个

位置（从0开始计数）上，代表第4分钟。现给定两对字符串，请帮助福尔摩斯解码得到约会的时间。

输入格式：

输入在4行中分别给出4个非空、不包含空格、且长度不超过60的字符串。

输出格式：

在一行中输出约会的时间，格式为“DAY HH:MM”，其中“DAY”是某星期的3字符缩写，即MON表示星期一，TUE表示星期二，WED表示星期三，

THU表示星期四，FRI表示星期五，SAT表示星期六，SUN表示星期日。题目输入保证每个测试存在唯一解。

输入样例：

3485djDkxh4hhGE

2984akDfkkkkggEdsb

s&hgsfdk

d&Hyscvnm

输出样例：

THU 14:04\*/

#include <stdio.h>

#include <iostream>

#include <string>

using namespace std;

int main()

{

string one, two, three, four;

cin >> one >> two >> three >> four;

string week[7] = { "MON","TUE","WED","THU","FRI","SAT","SUN" };

int count=1, i=0;

int length1 = one.length() < two.length() ? one.length() : two.length();

int length2 = three.length() < four.length() ? three.length() : four.length();//此处

for (i = 0; i < length1; i++)

{

if (count == 1 && one[i] == two[i] && one[i] >= 'A' && one[i] <= 'G')

{

cout << week[one[i] - 'A'] <<" ";

count++;

continue; //此处continue

}

if (count == 2 && one[i] == two[i] && (('0'<=two[i]&&two[i]<='9') || ('A' <= two[i]&&two[i] <= 'N')))

{

if ('0' <= two[i] && two[i] <= '9')

cout << '0' << one[i] << ":";

/\*cout<<" "表示在程序运行时在那个黑框子显示出你" "内的你所键入的内容。

而cout << ' '表示在程序运行时在那个黑框子显示出' '内的字符型变量所代表的内容。

例如，cout << "hello world"; 在黑框子上就会显示hello world，

而cout << ‘h';他会输出字符型h，而当单引号里的内容多余一个字符时他就会输出一个数，而不是你键入的内容了。\*/

else

cout << one[i] - 'A' + 10 << ":"; //输出格式

break;

}

}

for (i = 0; i < length2; i++)

{

if (three[i] == four[i] && (('a' <= three[i]&&three[i] <= 'z') || ('A' <= three[i] &&three[i]<= 'Z')))

{

if (i < 10)

cout << '0' << i;

else

cout << i;

break;

}

}

}

/\*正整数A的“DA（为1位整数）部分”定义为由A中所有DA组成的新整数PA。

例如：给定A = 3862767，DA = 6，

则A的“6部分”PA是66，因为A中有2个6。

现给定A、DA、B、DB，请编写程序计算PA + PB。

输入格式：

输入在一行中依次给出A、DA、B、DB，中间以空格分隔，其中0 < A, B < 1010。

输出格式：

在一行中输出PA + PB的值。

输入样例1：

3862767 6 13530293 3

输出样例1：

399

输入样例2：

3862767 1 13530293 8

输出样例2：

0\*/

#include <iostream>

using namespace std;

int main()

{

int A, DA, B, DB, k[2] = { 0 }, count, j = 1;

cin >> A >> DA >> B >> DB;

for (int i = 10; A >= 1;) //重复计数了，靠

{

count = A%i;

if (count == DA)

{

k[0] += DA\*j;

j = j \* 10;

}

A = A / 10;

}

j = 1;

for (int i = 10; B >= 1;)

{

count = B%i;

if (count == DB)

{

k[1] += DB\*j;

j = j \* 10;

}

B = B / 10;

}

cout << k[0]+k[1] ;

}

/\*本题要求计算A/B，其中A是不超过1000位的正整数，B是1位正整数。

你需要输出商数Q和余数R，使得A = B \* Q + R成立。

输入格式：

输入在1行中依次给出A和B，中间以1空格分隔。

输出格式：

在1行中依次输出Q和R，中间以1空格分隔。

输入样例：

123456789050987654321 7

输出样例：

17636684150141093474 3\*/

#include <iostream>

#include <string>

using namespace std;

int main()

{

string a;

int b, c[1000], count = 0 ;

cin >> a >> b;

for (int i = 0; i < a.length(); i++) //i完全临时变量

{

count = count \* 10 + a[i] - '0'; //机智的算法

c[i] = count / b;

count = count % b;

}

for (int i = 0; i < a.length(); i++)

{

if (c[i] == 0 && i == 0)

{

if (c[0] == 0 && (c[1] > 9 || c[1] < 0))

{

cout << "0";

continue;

}

else continue;

}

cout << c[i];

}

cout << " " << count;

}

/\*大家应该都会玩“锤子剪刀布”的游戏：两人同时给出手势，胜负规则如图所示：

现给出两人的交锋记录，请统计双方的胜、平、负次数，并且给出双方分别出什么手势的胜算最大。

输入格式：

输入第1行给出正整数N（<=105），即双方交锋的次数。随后N行，每行给出一次交锋的信息，

即甲、乙双方同时给出的的手势。C代表“锤子”、J代表“剪刀”、B代表“布”，第1个字母代表甲方，

第2个代表乙方，中间有1个空格。

输出格式：

输出第1、2行分别给出甲、乙的胜、平、负次数，数字间以1个空格分隔。第3行给出两个字母，

分别代表甲、乙获胜次数最多的手势，中间有1个空格。如果解不唯一，则输出按字母序最小的解。

输入样例：

10

C J

J B

C B

B B

B C

C C

C B

J B

B C

J J

输出样例：

5 3 2

2 3 5

B B\*/

#include <iostream>

#include <string>

using namespace std;

int main()

{

int n, x[2][3] = { 0 }, p[2][3] = { 0 };

char a[10000], b[10000];

cin >> n;

for (int i = 0; i < n; i++)

{

cin >> a[i] >> b[i];

if (a[i] == b[i])

{

x[0][1]++;

x[1][1]++;

}

if ((a[i] == 'C'&&b[i] == 'J')|| (a[i] == 'J'&&b[i] == 'B') || (a[i] == 'B'&&b[i] == 'C'))//甲胜

{

x[0][0]++;

x[1][2]++;

if (a[i] == 'C')

{

p[0][0]++;

}

if (a[i] == 'J')

{

p[0][1]++;

}

if (a[i] == 'B')

{

p[0][2]++;

}

}

if ((a[i] == 'C'&&b[i] == 'B') || (a[i] == 'J'&&b[i] == 'C') || (a[i] == 'B'&&b[i] == 'J'))//乙胜

{

x[0][2]++;

x[1][0]++;

if (b[i] == 'C')

{

p[1][0]++;

}

if (b[i] == 'J')

{

p[1][1]++;

}

if (b[i] == 'B')

{

p[1][2]++;

}

}

}

for (int i = 0; i < 2; i++)

{

cout << x[i][0] ;

cout << " " << x[i][1] << " "<<x[i][2]<<endl;

}

for (int i = 0; i < 2; i++)

{

x[0][0] = p[i][0] > p[i][1] ? p[i][0] : p[i][1];

x[0][0] = x[0][0] > p[i][2] ? x[0][0] : p[i][2];

if (x[0][0] == p[i][2])

a[i] = 'B';

else if (x[0][0] == p[i][0])

a[i] = 'C';

else if (x[0][0] == p[i][1])

a[i] = 'J';

}

cout << a[0] <<" " << a[1];

}

/\*给定任一个各位数字不完全相同的4位正整数，如果我们先把4个数字按非递增排序，再按非递减排序，

然后用第1个数字减第2个数字，将得到一个新的数字。一直重复这样做，我们很快会停在有“数字黑洞”之称的6174，

这个神奇的数字也叫Kaprekar常数。

例如，我们从6767开始，将得到

7766 - 6677 = 1089

9810 - 0189 = 9621

9621 - 1269 = 8352

8532 - 2358 = 6174

7641 - 1467 = 6174

... ...

现给定任意4位正整数，请编写程序演示到达黑洞的过程。

输入格式：

输入给出一个(0, 10000)区间内的正整数N。

输出格式：

如果N的4位数字全相等，则在一行内输出“N - N = 0000”；否则将计算的每一步在一行内输出，

直到6174作为差出现，输出格式见样例。注意每个数字按4位数格式输出。

输入样例1：

6767

输出样例1：

7766 - 6677 = 1089

9810 - 0189 = 9621

9621 - 1269 = 8352

8532 - 2358 = 6174

输入样例2：

2222

输出样例2：

2222 - 2222 = 0000\*/

#include <iostream>

#include <string>

using namespace std;

int sort1(int &a) //引用子函数的时候改变了原参数的值，保留了下来。但这个引用的a在主函数里无效。

{

int str[4], i ,k = a; //子函数中定义的所有变量，在主函数中均无效，只有在该主函数里有效。

for (i = 3; i >= 0; i--)

{

str[i] = k % 10;

k = k / 10;

}

int q = 0, p = 0, l;

for (p = 0; p < 4; p++)

{

for (q = p + 1; q < 4; q++)

{

if (str[p] < str[q])

{

l = str[p];

str[p] = str[q];

str[q] = l;

}

}

}

for (i = 0; i < 4; i++)

{

cout << str[i];

}

l = str[3] + str[2] \* 10 + str[1] \* 100 + str[0] \* 1000;

return l;

}

int sort2(int &a)

{

int str[4], i,k=a;

for (i = 3; i >= 0; i--)

{

str[i] = k % 10;

k = k / 10;

}

int q = 0, p = 0, l;

for (p = 0; p<4; p++)

for (q = p + 1; q<4; q++)

{

if (str[p] > str[q])

{

l = str[p];

str[p] = str[q];

str[q] = l;

}

}

for (i = 0; i < 4; i++)

{

cout << str[i];

}

l = str[3] + str[2] \* 10 + str[1] \* 100 + str[0] \* 1000;

return l;

}

int main()

{

int h[2], str[4];

int b, i;

cin >> b;

if (b % 1111 == 0)

{

cout << b << " - " << b << " = " << "0000" << endl;

return 0;

}

while (b != 6174)

{

h[0] = sort1(b); //先构思整体框架，再构造主函数，再构造子函数

cout << " - " ;

h[1] = sort2(b);

cout << " = " ;

b = h[0] - h[1];

int k = b;

for (i = 3; i >= 0; i--)

{

str[i] = k % 10;

k = k / 10;

}

for (i = 0; i < 4; i++)

{

cout << str[i];

}

cout << endl;

}

}

/\*月饼是中国人在中秋佳节时吃的一种传统食品，不同地区有许多不同风味的月饼。

现给定所有种类月饼的库存量、总售价、以及市场的最大需求量，请你计算可以获得的最大收益是多少。

注意：销售时允许取出一部分库存。样例给出的情形是这样的：

假如我们有3种月饼，其库存量分别为18、15、10万吨，

总售价分别为75、72、45亿元。如果市场的最大需求量只有20万吨，那么我们最大收益策略应该是卖出

全部15万吨第2种月饼、以及5万吨第3种月饼，

获得 72 + 45/2 = 94.5（亿元）。

输入格式：

每个输入包含1个测试用例。每个测试用例先给出一个不超过1000的正整数N表示月饼的种类数、以及

不超过500（以万吨为单位）的正整数D表示市场最大需求量。随后一行给出N个正数表示每种月饼的库存

量（以万吨为单位）；最后一行给出N个正数表示每种月饼的总售价（以亿元为单位）。数字间以空格分隔。

输出格式：

对每组测试用例，在一行中输出最大收益，以亿元为单位并精确到小数点后2位。

输入样例：

3 20

18 15 10

75 72 45

输出样例：

94.50\*/

#include <iostream>

#include <string>

using namespace std;

typedef struct

{

double a;

double b;

double c;

}G;

int main()

{

G x[1000], b;

int n, i=0, j, a, q=0;

double k=0;

cin >> n >> a; //有n种商品，市场需求为a万吨

for (i = 0; i < n; i++)

{

cin >> x[i].a;

}

for (i = 0; i < n; i++)

{

cin >> x[i].b;

}

for (i = 0; i < n; i++)

{

x[i].c = x[i].b / x[i].a;

}

for (i = 0; i < n; i++)

for (j = i + 1; j < n; j++)

{

if (x[i].c < x[j].c)

{

b.a = x[i].a;

x[i].a = x[j].a;

x[j].a = b.a;

b.b = x[i].b;

x[i].b = x[j].b;

x[j].b = b.b;

b.c = x[i].c;

x[i].c = x[j].c;

x[j].c = b.c;

}

}

/\*cout << x[0].a << x[1].a<<x[2].a;\*/

i = 0;

while (q <= a)

{

if (q + x[i].a < a)

{

q += x[i].a;

k += x[i].b;

}

else

{

q = a - q;

k += x[i].c\*q;

printf("%.2lf", k);

return 0;

}

i++;

}

}

/\*给定一个k位整数N = dk-1\*10k-1 + ... + d1\*101 + d0 (0<=di<=9, i=0,...,k-1, dk-1>0)，

请编写程序统计每种不同的个位数字出现的次数。例如：给定N = 100311，则有2个0，3个1，和1个3。

输入格式：

每个输入包含1个测试用例，即一个不超过1000位的正整数N。

输出格式：

对N中每一种不同的个位数字，以D:M的格式在一行中输出该位数字D及其在N中出现的次数M。要求按D的升序输出。

输入样例：

100311

输出样例：

0:2

1:3

3:1\*/

#include <iostream>

#include <string>

using namespace std; //c++大数的处理方法

int r[10] = { 0 };

int main()

{

string a; //字符串里面的元素均由char形式存储

int r[10] = { 0 };

cin >> a;

char x[10] = { '0','1','2','3','4','5','6','7','8','9' };

for (int i = 0; i < a.length(); i++) //字符串里的用法

for (int j = 0; j < 10; j++)

{

if (a[i] == x[j])

{

r[j]++;

}

}

for (int i = 0; i < 10; i++)

{

if (r[i] != 0)

{

cout << i << ":" << r[i]<<endl;

}

}

return 0;

}

进制的转化/\*输入两个非负10进制整数A和B(<=230-1)，输出A+B的D (1 < D <= 10)进制数。

输入格式：

输入在一行中依次给出3个整数A、B和D。

输出格式：

输出A+B的D进制数。

输入样例：

123 456 8

输出样例：

1103\*/

#include <iostream>

#include <string>

#include <math.h>

using namespace std;

int main()

{

long int a, b, c; //最大取到2的31次方

int d, i, j;

cin >> a >> b >> d;

c = a + b;

long int x[100] = { 0 }, y[100];

for (i = 0; i < 33; i++)

{

if(pow(d, i)<c)

x[i] = pow(d,i); //取次方的方法

else break;

}

a = 0;

for (j = i-1; j >= 0; j--)

{

y[a] = c / x[j];

cout << y[a];

c = c - x[j] \* y[a];

a++;

}

return 0;

}

/\*给定数字0-9各若干个。你可以以任意顺序排列这些数字，但必须全部使用。

目标是使得最后得到的数尽可能小（注意0不能做首位）。例如：给定两个0，两个1，

三个5，一个8，我们得到的最小的数就是10015558。

现给定数字，请编写程序输出能够组成的最小的数。

输入格式：

每个输入包含1个测试用例。每个测试用例在一行中给出10个非负整数，

顺序表示我们拥有数字0、数字1、……数字9的个数。整数间用一个空格分隔。

10个数字的总个数不超过50，且至少拥有1个非0的数字。

输出格式：

在一行中输出能够组成的最小的数。

输入样例：

2 2 0 0 0 3 0 0 1 0

输出样例：

10015558\*/

#include <iostream>

#include <string>

#include <math.h>

using namespace std;

int main()

{

int y[10], x[50], i = 0, j = 0, a=0 , b;

for (i = 0; i < 10; i++)

{

cin >> y[i];

}

for (i = 0; i < 10; i++)

{

for (j = 0; j < y[i]; j++)

{

x[a] = i;

a++;

}

}

for (i = 0; i < a; i++)

{

if (x[i] != 0)

{

b = x[i];

x[i] = x[0];

x[0] = b;

break;

}

}

for (i = 0; i < a; i++)

cout << x[i];

}

字符串！！/\*科学计数法是科学家用来表示很大或很小的数字的一种方便的方法，

其满足正则表达式[+-][1-9]"."[0-9]+E[+-][0-9]+，即数字的整数部分只有1位，

小数部分至少有1位，该数字及其指数部分的正负号即使对正数也必定明确给出。

现以科学计数法的格式给出实数A，请编写程序按普通数字表示法输出A，并保证所有有效位都被保留。

输入格式：

每个输入包含1个测试用例，即一个以科学计数法表示的实数A。该数字的存储长度不超过9999字节，

且其指数的绝对值不超过9999。

输出格式：

对每个测试用例，在一行中按普通数字表示法输出A，并保证所有有效位都被保留，包括末尾的0。

输入样例1：

+1.23400E-03

输出样例1：

0.00123400

输入样例2：

-1.2E+10

输出样例2：

-12000000000\*/

#include<stdio.h>

#include<iostream>

#include<string>

using namespace std;

int main()

{

int i, j = 0, k = 0;

string x, y, z, a, b;

cin >> x;

for (i = 0; i < x.length(); i++)

{

if (x[i] == '+' || x[i] == '-')

{

z = z + x[i];//可以这样写

}

else

y = y + x[i];

}

int flag = 1;

//cout << z<<endl<<y;

for (i = 0; i < y.length(); i++)

{

if (y[i] == 'E')

{

break;

}

}

for (j = y.length() - 1; j >i; j--)

{

k += (y[j] - '0')\*flag;

flag \*= 10;

} //读出k

if (k == 0)

{

if (z[0] == '-')

cout << z[0];

i = 0;

while (y[i] != 'E')

{

cout << y[i];

i++;

}

return 0;

}

//cout<<k<<endl;

//j = y.find(".");

//cout << j;

j = 1;

char n;

int sum=0;

if (z[1] == '+')

{

if(z[0]=='-')

cout << z[0];

cout << y[0];//y="+1.20000000E+05"; k=5; 输出样例:120000.000

for (i = 0; i < k; i++)

{

if (y[j + 1] != 'E')

{

cout << y[j + 1];

n = y[j + 1]; //可以这样写

y[j + 1] = y[j];

y[j] = n;

j++; //最终输出120000; y=120000.000E;j=6

}

else

{

cout << '0';

sum++; //立一个sum，出口不同接下来的处理不同

}

}

if (sum == 0)

{

while (y[j] != 'E')

{

cout << y[j];

j++;

}

}

return 0;

}

if (z[1] == '-')//y="1.23400E03"; k=3; 输出样例1：0.00123400

{

if (z[0] == '-')

cout << z[0];

cout << "0.";

for (i = 0; i < k - 1; i++)

{

cout << '0';

}

for (i = 0; i < y.length(); i++)

{

if (y[i] == '.')

continue;

if (y[i] == 'E')

break;

cout << y[i];

}

return 0;

}

}

数组的重构，利用序号技巧 & 输出格式！！！/\*(B1025.反转链表)给定一个常数K以及一个单链表L，请编写程序将L中每K个结点反转。

例如：给定L为1→2→3→4→5→6，K为3，则输出应该为3→2→1→6→5→4；

如果K为4，则输出应该为4→3→2→1→5→6，即最后不到K个元素不反转。

输入格式：

每个输入包含1个测试用例。每个测试用例第1行给出第1个结点的地址、结点总个

数正整数N(<= 105)、以及正整数K(<=N)，即要求反转的子链结点的个数。结点的

地址是5位非负整数，NULL地址用-1表示。

接下来有N行，每行格式为：

Address Data Next

其中Address是结点地址，Data是该结点保存的整数数据，Next是下一结点的地址。

输出格式：

对每个测试用例，顺序输出反转后的链表，其上每个结点占一行，格式与输入相同。

输入样例：

00100 6 4

00000 4 99999

00100 1 12309

68237 6 -1

33218 3 00000

99999 5 68237

12309 2 33218

输出样例：

00000 4 33218

33218 3 12309

12309 2 00100

00100 1 99999

99999 5 68237

68237 6 -1\*/

#include<stdio.h>

#include<iostream>

#include<algorithm>

using namespace std;

typedef struct

{

int data;

int address;

}G;

int main()

{

int n, k, addr, first, next[200], res[200];//res存储“编号”

//尽量避免复杂的数据交换，使用了结构数组的“排号”，避重就轻

G x[200];

cin >> first >> n >> k;

for (int i = 0; i < n; i++)

{

cin >> x[i].address >> x[i].data >> next[i];

}

int sum = 0;

addr = first;

while (addr != -1)//由address和next排出res[]

{

for (int i = 0; i < n; i++)

{

if (x[i].address == addr)

{

res[sum++] = i;//sum作用完后+1

addr = next[i];

break;

}

}

}

int j = 0;

for (int i = 0; i < sum / k; i ++)//res[]前每n个直接调转

{

reverse(res + j, res + j + k);//这里定义了一个j来实现多次反转中每次的计数

j += k;

}

for (int i = 0; i < sum - 1; i++)//重新定义next[]地址数组

{

next[res[i]] = x[res[i+1]].address;

}

for (int i = 0; i < sum - 1; i++)//注意输出格式

{

printf("%05d %d %05d\n", x[res[i]].address, x[res[i]].data, next[res[i]]);//输出5位整数

}

printf("%05d %d -1\n", x[res[sum-1]].address, x[res[sum-1]].data);

return 0;

}

四舍五入！！/\*(B1026.程序运行时间)要获得一个C语言程序的运行时间，常用的方法是调用头文件time.h，其中提供了clock()函数，

可以捕捉从程序开始运行到clock()被调用时所耗费的时间。这个时间单位是clock tick，即“时钟打点”。

同时还有一个常数CLK\_TCK，给出了机器时钟每秒所走的时钟打点数。于是为了获得一个函数f的运行时间，

我们只要在调用f之前先调用clock()，获得一个时钟打点数C1；在f执行完成后再调用clock()，

获得另一个时钟打点数C2；两次获得的时钟打点数之差(C2-C1)就是f运行所消耗的时钟打点数，

再除以常数CLK\_TCK，就得到了以秒为单位的运行时间。

这里不妨简单假设常数CLK\_TCK为100。现给定被测函数前后两次获得的时钟打点数，请你给出被测函数运行的时间。

输入格式：

输入在一行中顺序给出2个整数C1和C2。注意两次获得的时钟打点数肯定不相同，即C1 < C2，并且取值在[0, 107]。

输出格式：

在一行中输出被测函数运行的时间。运行时间必须按照“hh:mm:ss”（即2位的“时:分:秒”）

格式输出；不足1秒的时间四舍五入到秒。

输入样例：

123 4577973

输出样例：

12:42:59\*/

#include<stdio.h>

#include<iostream>

#include<algorithm>

#include<time.h>

using namespace std;

int main()

{

long long c1, c2;

cin >> c1 >> c2;

long long q = c2 - c1;//总共秒

int n = (q+50) / 100;//喵的，是“四舍五入”不是“全入”

int m = n / 60;//总共分

int k = n % 60;//秒

int h = m / 60;//时

int a = m % 60;//分

//cout << h << ':' << a << ':' << k;

printf("%02d:%02d:%02d", h, a, k);

return 0;

}

沙漏右边的空格不能打！！/\*(B1027.打印沙漏)本题要求你写个程序把给定的符号打印成沙漏的形状。例如给定17个“\*”，要求按下列格式打印

\*\*\*\*\*

\*\*\*

\*

\*\*\*

\*\*\*\*\*

所谓“沙漏形状”，是指每行输出奇数个符号；各行符号中心对齐；相邻两行符号数差2；

符号数先从大到小顺序递减到1，再从小到大顺序递增；首尾符号数相等。

给定任意N个符号，不一定能正好组成一个沙漏。要求打印出的沙漏能用掉尽可能多的符号。

输入格式：

输入在一行给出1个正整数N（<=1000）和一个符号，中间以空格分隔。

输出格式：

首先打印出由给定符号组成的最大的沙漏形状，最后在一行中输出剩下没用掉的符号数。

输入样例：

19 \*

输出样例：

\*\*\*\*\*

\*\*\*

\*

\*\*\*

\*\*\*\*\*

2\*/

#include<stdio.h>

#include<iostream>

#include<algorithm>

#include<time.h>

using namespace std;

int main()

{

int n;

char s;

cin >> n >> s;

int a = 1, b = 1, c = 0;

if (n == 1)

{

cout << s << endl << '0';//右边的空格不能打！！！

return 0;

}

while (a <= n)

{

b += 2;

c++;//c是几层（包括基层）

a += 2 \* b;

}

a = a - 2 \* b;//a是总共利用了几个

int q = n - a;

//cout << a << ' ' << n << ' ' << c;

b = 0;

for (int i = 0; i < c; i++)

{

for (int j = 2\*c-1; j > 0; j--)

{

if (2 \* c - j <= i)

cout << ' ';

else if (j <= i)

continue;

else

cout << s;

}

cout << endl;

}

for (int i = 0; i < c - 1; i++)

{

for (int j = 0; j < 2 \* c - 1; j++)

{

if (j < c - 2 - i)

cout << ' ';

else if (j > 2 \* c - 2 - (c - 2 - i))

continue;

else

cout << s;

}

cout << endl;

}

cout << q;

return 0;

}

结构体数组的排序、sort函数！！(没全对，原因未知)/\*(B1028.人口普查)某城镇进行人口普查，得到了全体居民的生日。现请你写个程序，找出镇上最年长和最年轻的人。

这里确保每个输入的日期都是合法的，但不一定是合理的——假设已知镇上没有超过200岁的老人，而今天是2014年9月6日，

所以超过200岁的生日和未出生的生日都是不合理的，应该被过滤掉。

输入格式：

输入在第一行给出正整数N，取值在(0, 105]；随后N行，每行给出1个人的姓名（由不超过5个英文字母组成的字符串）、

以及按“yyyy/mm/dd”（即年/月/日）格式给出的生日。题目保证最年长和最年轻的人没有并列。

输出格式：

在一行中顺序输出有效生日的个数、最年长人和最年轻人的姓名，其间以空格分隔。

输入样例：

5

John 2001/05/12

Tom 1814/09/06

Ann 2121/01/30

James 1814/09/05

Steve 1967/11/20

输出样例：

3 Tom John\*/

#include<stdio.h>

#include<iostream>

#include<algorithm>

#include<time.h>

#include<string>

using namespace std;

typedef struct

{

string name;

int year, month, day, sum;

}G;

bool cmp(G a,G b) //需自定义函数

{

return a.sum < b.sum;

}

int main()

{

int n, c;

cin >> n;

G x[10000];

for (int i = 0; i < n; i++)

{

cin >> x[i].name;

cin >> x[i].year;

c = getchar();

if (c == '/')

{

c = 0;

cin >> x[i].month;

c = getchar();

if (c == '/')

{

cin >> x[i].day;

continue;

}

}

}

int q[105];//q存放“序号”

for (int i = 0; i < n; i++)

{

x[i].sum = x[i].year \* 10000 + x[i].month \* 100 + x[i].day;//数组与原结构体的对应

}

sort(x, x + n, cmp);//结构体数组的排序

int f = n;

for (int i = 0; i < n; i++)

{

if (x[i].sum < 18140906||x[i].sum>20140906)

{

f--;

x[i].sum = -1;

}

}

cout << f;

if (f == 0)

{

return 0;

}

for (int i = 0; i < n; i++)//从前往后

{

if (x[i].sum != -1)

{

cout <<' '<< x[i].name;

break;

}

}

for (int i = n-1; i >0; i--)

{

if (x[i].sum != -1)

{

cout << ' ' << x[i].name;

break;

}

}

return 0;

}

ASCII码的应用，string的处理find，length的应用/\*(B1029.旧键盘)旧键盘上坏了几个键，于是在敲一段文字的时候，对应的字符就不会出现。

现在给出应该输入的一段文字、以及实际被输入的文字，请你列出肯定坏掉的那些键。

输入格式：

输入在2行中分别给出应该输入的文字、以及实际被输入的文字。每段文字是不超过80个字符的串，

由字母A-Z（包括大、小写）、数字0-9、以及下划线“\_”（代表空格）组成。题目保证2个字符串均非空。

输出格式：

按照发现顺序，在一行中输出坏掉的键。其中英文字母只输出大写，每个坏键只输出一次。题目保证至少有1个坏键。

输入样例：

7\_This\_is\_a\_test

\_hs\_s\_a\_es

输出样例：

7TI\*/

#include<stdio.h>

#include<iostream>

#include<algorithm>

#include<time.h>

#include<string>

using namespace std;

int main()

{

string should, now, x;

cin >> should >> now;

int j = 0;

for (int i = 0; i < should.length(); i++)

{

if (should[i] < 58) //非字母

{

if (should[i] == now[j])

{

j++;

continue;

}

if (should[i] != now[j])

{

if(x.find(should[i])==string::npos)//没找到！！

x += should[i];

}

}

if (should[i] > 58)//字母

{

if (( should[i] == now[j])|| (should[i] + 32 == now[j])|| (should[i] == now[j] + 32 ))//相同

{

j++;

continue;

}

else //不相同

{

if (should[i] < 97)//should[i]大写，下划线的ASCII码在大写65-90，小写97-122之间！！

{

if (x.find(should[i]) == string::npos)

{

x += should[i];

}

}

else//should[i]小写

if (x.find(should[i] - 32) == string::npos)

{

x += should[i] - 32;

}

}

}

}

cout << x;

}

排序，遍历，程序优化，以及变量类型，如何避免超时？/\*(B1030.完美数列)给定一个正整数数列，和正整数p，设这个数列中的最大值是M，

最小值是m，如果M <= m \* p，则称这个数列是完美数列。

现在给定参数p和一些正整数，请你从中选择尽可能多的数构成一个完美数列。

输入格式：

输入第一行给出两个正整数N和p，其中N（<= 105）是输入的正整数的个数，

p（<= 109）是给定的参数。第二行给出N个正整数，每个数不超过109。

输出格式：

在一行中输出最多可以选择多少个数可以用它们组成一个完美数列。

输入样例：

10 8

2 3 20 4 5 1 6 7 8 9

输出样例：

8\*/

#include<stdio.h>

#include<iostream>

#include<algorithm>

#include<time.h>

#include<string>

using namespace std;

int main()

{

long long n, p, j;

cin >> n >> p;

long long x[10000], y[10000];//100000就定义数组超限了。另外10的9次方居然int超限，需要long long型。不能用n定义数组，换IDE了。

for (int i = 0; i < n; i++)

cin >> x[i];

sort(x, x + n);

int sum = 0 ;

for (int i = 0; i < n; i++)

{

for (j = i + sum; j < n; j++)//精髓，j从i+sum开始查找，利用了之前的结果，加快了运行速度。

{

if (x[j] > x[i] \* p)

break;

sum++;

}

}

cout << sum;

return 0;

}

字符串的处理，字符与整数之间的转换与加减，取模=求余数。/\*(B1031.查验身份证)一个合法的身份证号码由17位地区、日期编号和顺序编号加1位校验码组成。校验码的计算规则如下：

首先对前17位数字加权求和，权重分配为：{7，9，10，5，8，4，2，1，6，3，7，9，10，5，8，4，2}；

然后将计算的和对11取模得到值Z；最后按照以下关系对应Z值与校验码M的值：

Z：0 1 2 3 4 5 6 7 8 9 10

M：1 0 X 9 8 7 6 5 4 3 2

现在给定一些身份证号码，请你验证校验码的有效性，并输出有问题的号码。

输入格式：

输入第一行给出正整数N（<= 100）是输入的身份证号码的个数。随后N行，每行给出1个18位身份证号码。

输出格式：

按照输入的顺序每行输出1个有问题的身份证号码。这里并不检验前17位是否合理，只检查前17位是

否全为数字且最后1位校验码计算准确。如果所有号码都正常，则输出“All passed”。

输入样例1：

4

320124198808240056

12010X198901011234

110108196711301866

37070419881216001X

输出样例1：

12010X198901011234

110108196711301866

37070419881216001X

输入样例2：

2

320124198808240056

110108196711301862

输出样例2：

All passed\*/

#include<stdio.h>

#include<iostream>

#include<algorithm>

#include<time.h>

#include<string>

using namespace std;

int main()

{

int n; //n个身份证

string x[100];//存储各身份证号

cin >> n;

for (int i = 0; i<n; i++)//输入每一个身份证

{

cin >> x[i];

}

int a[17] = { 7,9,10,5,8,4,2,1,6,3,7,9,10,5,8,4,2 };

int b[100]={0};//存储每一个身份证号的Z值

for (int i = 0; i<n; i++)

{

for (int j = 0; j<17; j++)

{

if (x[i][j] - '0' < 0 || x[i][j] - '9'>0)

{

b[i] = -1;

break;

}

else

b[i] += a[j] \* (x[i][j]-'0');

}

if (b[i] == -1)

{

continue;

}

b[i] = b[i] % 11;//取模就是求余数。。

}

int m[11] = { 1, 0, 10, 9, 8, 7, 6, 5, 4, 3, 2 };

int result[100] = { 0 };//存储合理/不合理身份证序号

int j = 0;

for (int i = 0; i < n; i++)

{

if (x[i][17] == 'X' && m[b[i]] == 10)//合理

{

result[i] = 0;//竟然把m和result搞混了

continue;

}

if ((x[i][17] - '0') == m[b[i]])//合理

{

result[i] = 0;

continue;

}

result[i] = 1;

j++;

}

if (j == 0)

{

cout << "All passed";

return 0;

}

for (int i = 0; i<n; i++)

{

if (result[i] == 1)

cout << x[i] << endl;

}

return 0;

}

不错，但是超时了，而且算法很蠢。

数组中find的用法。大数量结构体的排序必然有超时的可能。

（从1开始连续编号）好好读题，不要给自己人为增加难度/\*(B1031.查验身份证)为了用事实说明挖掘机技术到底哪家强，PAT组织了一场挖掘机技能大赛。

现请你根据比赛结果统计出技术最强的那个学校。

输入格式：

输入在第1行给出不超过105的正整数N，即参赛人数。随后N行，每行给出一位参赛者的信息和成绩，

包括其所代表的学校的编号（从1开始连续编号）、及其比赛成绩（百分制），中间以空格分隔。

输出格式：

在一行中给出总得分最高的学校的编号、及其总分，中间以空格分隔。题目保证答案唯一，没有并列。

输入样例：

6

3 65

2 80

1 100

2 70

3 40

3 0

输出样例：

2 150

\*/

#include<stdio.h>

#include<iostream>

#include<algorithm>

#include<time.h>

#include<string>

using namespace std;

struct G

{

int num;

int score;

};

int main()

{

int n;

cin >> n;

struct G x[n], max;//又被所谓的105坑了

for (int i = 0; i < n; i++)

{

cin >> x[i].num >> x[i].score;

}

int p[n], q[n];

int j = 0;

for (int i = 0; i < n; i++)//每一位选手筛选

{

int \*l = find(p, p + n, x[i].num);//find的用法，找到了\*l返回该被找数，没找到指针指向范围中的最后一个。

if (l == p + n)//没找到

{

p[j] = x[i].num;

q[j] = x[i].score;

j++;

if (x[i].score > max.score)

{

max.num = x[i].num;

max.score = x[i].score;

}

}

else

{

int m;

for ( m = 0; m < j; m++)

{

if (\*l == p[m])

{

q[m] += x[i].score;

break;

}

}

if (q[m] > max.score)

{

max.num = p[m];

max.score = q[m];

}

}

}

cout << max.num << ' ' << max.score;

return 0;

}

看看这个版本，相比起来就机智的多了

1. 看清题目（从1开始连续编号）
2. 输入环节善于利用数组的下标直接进行累加
3. 利用副本，和原对应起来解决问题

#include<stdio.h>

#include<iostream>

#include<algorithm>

#include<time.h>

#include<string>

using namespace std;

int main()

{

int n, i;

int x[100000] = { 0 }, y[100000];//如果用n的话就不能这样直接赋值

cin >> n;

for (i = 0; i < n; i++)

{

int num, result;

cin >> num >> result;

x[num-1]+=result;

}

for (i = 0; i < n; i++)//副本，将副本进行排序，与原进行比较。

{

y[i] = x[i];

}

sort(y, y + n);

for (i = 0; i < n; i++)

{

if (x[i] == y[n - 1])

{

cout << i + 1 << ' ' << y[n - 1];

return 0;

}

}

}

大小写转换，ASCII码的运用，或用int troupper()记得初始化变量(小写转化为大写)！！尤其是分两种情况的时候，每种情况边量在用之前都要初始化！！（有一处错误，未知）/\*(B1033.旧键盘打字)旧键盘上坏了几个键，于是在敲一段文字的时候，对应的字符就不会出现。现在给出应该输入

的一段文字、以及坏掉的那些键，打出的结果文字会是怎样？

输入格式：

输入在2行中分别给出坏掉的那些键、以及应该输入的文字。其中对应英文字母的坏键以大写给出；每段文字是不超

过105个字符的串。可用的字符包括字母[a-z, A-Z]、数字0-9、以及下划线“\_”（代表空格）、“,”、“.”、

“-”、“+”（代表上档键）。题目保证第2行输入的文字串非空。

注意：如果上档键坏掉了，那么大写的英文字母无法被打出。

输出格式：

在一行中输出能够被打出的结果文字。如果没有一个字符能被打出，则输出空行。

输入样例：

7IE.

7\_This\_is\_a\_test.

输出样例：

\_hs\_s\_a\_tst\*/

#include<stdio.h>

#include<iostream>

#include<algorithm>

#include<time.h>

#include<string>

using namespace std;

int main()

{

string x, y;//x代表坏掉的键，y代表应该输入的语句

cin >> x >> y;

int tag = 0, i, j=0;//没有+号时，tag的初始化！！！靠

if (x.find('+') != string::npos)

{

tag = -1;

}

if (tag == -1)//上档键坏掉

{

for (i = 0; i < y.length(); i++)

{

if ('A' <= y[i]&&y[i] <= 'Z')//不能65<=y[i]<=90这样写！！

{

continue;

}

else if ('a' <= y[i]&&y[i] <= 'z')//小写

{

if (x.find(toupper(y[i])) == string::npos)//加减，toupper(a)将a的小写变大写，若a不是小写字母则返回a

{

cout << y[i];

j++;

}

}

else

{

if (x.find(y[i]) == string::npos)

{

cout << y[i];

j++;

}

}

}

}

else

{

for (i = 0; i < y.length(); i++)

{

if ('a' <= y[i] && y[i] <= 'z')//小写

{

if (x.find(y[i] - 32) == string::npos)//没找到

{

cout << y[i];

j++;

}

}

else

{

if (x.find(y[i]) == string::npos)//其他没找到

{

cout << y[i];

j++;

}

}

}

}

return 0;

}

好啰嗦的一道题，可能写的太长最后运行超时了。其中涉及到string下find函数的运用，子函数与引用“出现了一个问题 导致程序停止正常工作”通常是 数组超限or出现了除以0的运算。真的好心累/\*(B1034.有理数的四则运算)本题要求编写程序，计算2个有理数的和、差、积、商。

输入格式：

输入在一行中按照“a1/b1 a2/b2”的格式给出两个分数形式的有理数，其中分子和分母全是整型范围内的整数，

负号只可能出现在分子前，分母不为0。

输出格式：

分别在4行中按照“有理数1 运算符 有理数2 = 结果”的格式顺序输出2个有理数的和、差、积、商。注意输出

的每个有理数必须是该有理数的最简形式“k a/b”，其中k是整数部分，a/b是最简分数部分；若为负数，则须

加括号；若除法分母为0，则输出“Inf”。题目保证正确的输出中没有超过整型范围的整数。

输入样例1：

2/3 -4/2

输出样例1：

2/3 + (-2) = (-1 1/3)

2/3 - (-2) = 2 2/3

2/3 \* (-2) = (-1 1/3)

2/3 / (-2) = (-1/3)

输入样例2：

5/3 0/6

输出样例2：

1 2/3 + 0 = 1 2/3

1 2/3 - 0 = 1 2/3

1 2/3 \* 0 = 0

1 2/3 / 0 = Inf\*/

#include<stdio.h>

#include<iostream>

#include<algorithm>

#include<time.h>

#include<string>

using namespace std;

long long beishu(long long a, long long b)

{

long long i, c;

for (i = a\*b; i >= a&&i >= b; i--)

{

if (i%a == 0)

{

if (i%b == 0)

c = i;

}

}

return c;

}

long long yueshu(long long a, long long b)//没考虑负数范围

{

long long i, c;

if (a<0)

{

a = -a;

}

for (i = 1; i <= a&&i <= b; i++)

if (a%i == 0)

{

if (b%i == 0)

c = i;

};

return c;

}

int shuchu(long long a, long long b)//不会影响到原参数 ，加\*&有影响，表示引用

{

// cout<<a<<' '<<b<<' '<<c<<' ';

if (a == 0)

{

printf("0");

return 0;

}

long long c = yueshu(a, b);

a = a / c;

b = b / c;

if (a < 0)

{

if ((-a) % b == 0)//先判断这个

{

printf("(-%d)", -a / b);

}

else if (-a > b)

{

printf("(-%d %d/%d)", -a / b, -a - (-a / b)\*b, b);//这里

}

else

printf("(-%d/%d)", -a, b);

}

else if (a%b == 0)

{

printf("%d", a / b);

}

else if (a > b)

{

printf("%d %d/%d", a / b, a - (a / b)\*b, b);//这里

}

else

printf("%d/%d", a, b);

return 0;

}

int main()

{

string a, b;

cin >> a >> b;

long long x = a.find('/');//分别找到a和b中除号的位置

long long y = b.find('/');

long long flag = 0; //记录正负号

long long c[3] = { 0 }, i, count = 1;//忘记c数组初始化！！！

for (i = a.length() - 1; i > x; i--)//忘记减一！！ length是长度，不是下标！！！

{

c[1] += count\*(a[i] - '0');

count \*= 10;

}

count = 1;

for (i = b.length() - 1; i > y; i--)

{

c[2] += count\*(b[i] - '0');

count \*= 10;

}

c[0] = beishu(c[1], c[2]);//c[0]为两数分母的最小公倍数

//cout << c[1] << ' ' << c[2] << ' ' << c[0];

count = 1;

long long p[3] = { 0 }, q[3] = { 0 };//p[1],p[2]分别记录两数分子，p[0]记录其和

for (i = x - 1; a[i] != '-' && i >= 0; i--)//忘记-1。

{

p[1] += count\*(a[i] - '0');

count \*= 10;

}

if (a[0] == '-')

p[1] = -1 \* p[1];

p[1] = p[1] \* c[0] / c[1];//p[1]储存最小公倍数为分母下的a分子,cnm写成p[1]了。。

count = 1; //c[1]是第一个数的分母，c[0]是两个数的最小公倍数。

for (i = y - 1; b[i] != '-'&&i >= 0; i--)

{

p[2] += count\*(b[i] - '0');

count \*= 10;

}

if (b[0] == '-')

p[2] = -1 \* p[2];

p[2] = p[2] \* c[0] / c[2];//p[2]储存最小公倍数为分母下的b分子

// cout<<p[1]<<' '<<p[2]<<' '<<c[0];

for (i = 0; i < 4; i++)

{

// cout << endl << p[1] << ' ' << p[2] << ' ' << c[0] << endl;

// cout<<i<<endl;

switch (i)

{

case 0:

{

p[0] = p[1] + p[2];//p[0]储存最小公倍数为分母下的分子之和

shuchu(p[1], c[0]);

printf(" + ");

shuchu(p[2], c[0]);

printf(" = ");

shuchu(p[0], c[0]);

printf("\n");

break;

}

case 1:

{

p[0] = p[1] - p[2];//p[0]储存最小公倍数为分母下的分子之差

shuchu(p[1], c[0]);

printf(" - ");

shuchu(p[2], c[0]);

printf(" = ");

shuchu(p[0], c[0]);

printf("\n");

break;

}

case 2:

{

p[0] = p[1] \* p[2];//p[0]储存最小公倍数为分母下的分子之积

int h = c[0] \* c[0];

shuchu(p[1], c[0]);

printf(" \* ");

shuchu(p[2], c[0]);

printf(" = ");

shuchu(p[0], h);

printf("\n");

break;

}

case 3:

{

shuchu(p[1], c[0]);

printf(" / ");

shuchu(p[2], c[0]);

printf(" = ");

if (p[2] == 0)

{

printf("Inf");

return 0;

}

if (p[2]<0)

{

int p1 = -p[1];

int p2 = -p[2];

shuchu(p1, p2);

break;

}

shuchu(p[1], p[2]);

break;

}

}

}

}

排序算法考虑不周全，merge sort遗漏了结尾的排序，导致出错。有一个自己写的引用指针的merge()函数不能用，但用指针表示数组的judge函数还是可以用的。排序。algorithm下数组排序sort()函数的具体用法。/\*(B1035.插入与归并)根据维基百科的定义：

插入排序是迭代算法，逐一获得输入数据，逐步产生有序的输出序列。每步迭代中，算法从输入序列中取出

一元素，将之插入有序序列中正确的位置。如此迭代直到全部元素有序。

归并排序进行如下迭代操作：首先将原始序列看成N个只包含1个元素的有序子序列，然后每次迭代归并两个

相邻的有序子序列，直到最后只剩下1个有序的序列。

现给定原始序列和由某排序算法产生的中间序列，请你判断该算法究竟是哪种排序算法？

输入格式：

输入在第一行给出正整数N (<=100)；随后一行给出原始序列的N个整数；最后一行给出由某排序算法产生的

中间序列。这里假设排序的目标序列是升序。数字间以空格分隔。

输出格式：

首先在第1行中输出“Insertion Sort”表示插入排序、或“Merge Sort”表示归并排序；然后在第2行中输

出用该排序算法再迭代一轮的结果序列。题目保证每组测试的结果是唯一的。数字间以空格分隔，且行末不得有多余空格。

输入样例1：

10

3 1 2 8 7 5 9 4 6 0

1 2 3 7 8 5 9 4 6 0

输出样例1：

Insertion Sort

1 2 3 5 7 8 9 4 6 0

输入样例2：

10

3 1 2 8 7 5 9 4 0 6

1 3 2 8 5 7 4 9 0 6

输出样例2：

Merge Sort

1 2 3 8 4 5 7 9 0 6\*/

#include <stdio.h>

#include <math.h>

#include <algorithm>

#include <iostream>

using namespace std;

bool judge(int\*a, int\*b, int N)

{

for (int i = 0; i < N; i++)

{

if (\*(a + i) != \*(b + i))

{

return false;

}

}

return true;

}

void merge(int \*&a, int i, int &N)//为啥不能用。

{

int i1 = pow(2, i);

int h = 1;

N = N / 2;

for (int j = 0; j < N; j++)

{

sort(a + h - 1, a + i1 + h);

h = h + i1;

}

}

int main()

{

int N, a[100], b[100], a1[100], a2[100];//这里写成N，则必须写在cin后面。

cin >> N;

int N1 = N;

for (int i = 0; i < N; i++)

{

cin >> a[i];

}

for (int i = 0; i < N; i++)

{

cin >> b[i];

}

for (int i = 0; i < N; i++)

{

a1[i] = a[i];

a2[i] = a[i];

}

for (int i = 0; i < N; i++)

{

sort(a1, a1 + i + 2);

int i1 = pow(2, i + 1);//特喵的，sort的用法套路这么深.前一个指针（不参加）之后到后一个指针（参加）

int h = 0;

N1 = N1 / 2;

for (int j = 0; j < N1; j++)

{

sort(a2 + h, a2 + i1 + h);//基指针指向数组的第一个元素前

h = h + i1;

}

sort(a2 + h, a2 + N);//一个成功的debug发现了算法的漏洞，在这里应该把余下的未排序的数加以排序。

if (judge(a1, b, N))

{

cout << "Insertion Sort" << endl;

sort(a1, a1 + i + 3);

for (int j = 0; j < N - 1; j++)

{

cout << a1[j] << ' ';

}

cout << a1[N - 1];

return 0;

}

else if (judge(a2, b, N))

{

cout << "Merge Sort" << endl;

int i1 = pow(2, i + 2);

int h = 0;

N1 = N1 / 2;

for (int j = 0; j < N1; j++)

{

sort(a2 + h, a2 + i1 + h);

h = h + i1;

}

sort(a2 + h, a2 + N);

for (int j = 0; j < N - 1; j++)

{

cout << a2[j] << ' ';

}

cout << a2[N - 1];

return 0;

}

}

}

画图形，没难度。/\*(B1036.跟奥巴马一起学编程)美国总统奥巴马不仅呼吁所有人都学习编程，甚至以身作则编写代码，成为美国历史上首位

编写计算机代码的总统。2014年底，为庆祝“计算机科学教育周”正式启动，奥巴马编写了很简单的计算机代码：在屏幕

上画一个正方形。现在你也跟他一起画吧！

输入格式：

输入在一行中给出正方形边长N（3<=N<=20）和组成正方形边的某种字符C，间隔一个空格。

输出格式：

输出由给定字符C画出的正方形。但是注意到行间距比列间距大，所以为了让结果看上去更像正方形，我们输出的行数实际上

是列数的50%（四舍五入取整）。

输入样例：

10 a

输出样例：

aaaaaaaaaa

a a

a a

a a

aaaaaaaaaa\*/

#include <stdio.h>

#include <math.h>

#include <algorithm>

#include <iostream>

using namespace std;

int main()

{

int n;

char a;

cin >> n >> a;

for (int i = 0; i < (n + 1) / 2; i++)//i表示行数，这里犯了错。

{

for (int j = 0; j < n; j++)//j表示每一列

{

if (i == 0 || i == (n + 1) / 2-1)//减一，以输出最后一行

cout << a;

else if (j == 0 || j == n - 1)

cout << a;

else

cout << ' ';

}

if (i == (n + 1) / 2)

return 0;

cout << endl;

}

}

特定的输入格式下用getchar()输入，找零钱，求余数/\*(B1037.在霍格沃茨找零钱)如果你是哈利·波特迷，你会知道魔法世界有它自己的货币系统 —— 就如海格告诉哈

利的：“十七个银西可(Sickle)兑一个加隆(Galleon)，二十九个纳特(Knut)兑一个西可，很容易。”现在，

给定哈利应付的价钱P和他实付的钱A，你的任务是写一个程序来计算他应该被找的零钱。

输入格式：

输入在1行中分别给出P和A，格式为“Galleon.Sickle.Knut”，其间用1个空格分隔。这里Galleon是[0, 107]区间

内的整数，Sickle是[0, 17)区间内的整数，Knut是[0, 29)区间内的整数。

输出格式：

在一行中用与输入同样的格式输出哈利应该被找的零钱。如果他没带够钱，那么输出的应该是负数。

输入样例1：

10.16.27 14.1.28

输出样例1：

3.2.1

输入样例2：

14.1.28 10.16.27

输出样例2：

-3.2.1\*/

#include <stdio.h>

#include <math.h>

#include <algorithm>

#include <iostream>

using namespace std;

int main()

{

long long x[3], y[3], z[3], sum[3];

for (int i = 0; i < 3; i++)//特定的输入格式下用getchar()输入

{

cin >> x[i];

int b = getchar();

if (b == '.')

continue;

}

for (int i = 0; i < 3; i++)

{

cin >> y[i];

int b = getchar();

if (b == '.')

continue;

}

sum[0] = x[0] \* 17 \* 29 + x[1] \* 29 + x[2];//实际带的钱，单位Knut

sum[1] = y[0] \* 17 \* 29 + y[1] \* 29 + y[2];//价格，单位Knut

sum[2] = sum[1] - sum[0];

int flag = 0;

if (sum[2] < 0)

{

flag = 1;

sum[2] = -sum[2];

}

z[2] = sum[2] % 29;

sum[2] = sum[2] / 29;

z[1] = sum[2] % 17;

sum[2] = sum[2] / 17;

z[0] = sum[2];

if (flag == 1)

cout << '-';

for (int i = 0; i < 2; i++)

{

cout << z[i] << '.';

}

cout << z[2];

return 0;

}

数组上界问题，“以数组下标为记成绩，数组的数为记次数的算法”，大大节省了时间。一道经典的debug。/\*(B1038.统计同成绩的学生)本题要求读入N名学生的成绩，将获得某一给定分数的学生人数输出。

输入格式：

输入在第1行给出不超过105的正整数N，即学生总人数。随后1行给出N名学生的百分制整数成绩，中间以空格分隔。

最后1行给出要查询的分数个数K（不超过N的正整数），随后是K个分数，中间以空格分隔。

输出格式：

在一行中按查询顺序给出得分等于指定分数的学生人数，中间以空格分隔，但行末不得有多余空格。

输入样例：

10

60 75 90 55 75 99 82 90 75 50

3 75 90 88

输出样例：

3 2 0\*/

#include <stdio.h>

#include <math.h>

#include <algorithm>

#include <iostream>

using namespace std;

int main()

{

int n, num;

cin >> n;

int a[101] = { 0 };//存储所有学生成绩，这里，101。因为0到100一共101个数。。若是100则定义的是a[0]到a[99]。。

for (int i = 0; i < n; i++)

{

cin >> num;

a[num]++;

}

int m;//存储要找的学生成绩。草，就是这里出错了

cin >> m;

int b[m];//写在这个后面。m可以很多，不只有100个。注意审题。

for (int i = 0; i < m; i++)

{

cin >> b[i];

}

for (int i = 0; i < m - 1; i++)

{

cout << a[b[i]] << ' ';

}

cout << a[b[m-1]];

return 0;

}

字符串的遍历，字符串find()的用法，防止重复遍历置一个其他值的思想/\*(B1039.到底买不买)小红想买些珠子做一串自己喜欢的珠串。卖珠子的摊主有很多串五颜六色的珠串，

但是不肯把任何一串拆散了卖。于是小红要你帮忙判断一下，某串珠子里是否包含了全部自己想要的珠子？如果是，

那么告诉她有多少多余的珠子；如果不是，那么告诉她缺了多少珠子。

为方便起见，我们用[0-9]、[a-z]、[A-Z]范围内的字符来表示颜色。例如在图1中，第3串是小红想做的珠串；那么

第1串可以买，因为包含了全部她想要的珠子，还多了8颗不需要的珠子；第2串不能买，因为没有黑色珠子，并且

少了一颗红色的珠子。

图 1

输入格式：

每个输入包含1个测试用例。每个测试用例分别在2行中先后给出摊主的珠串和小红想做的珠串，两串都不超过1000个珠子。

输出格式：

如果可以买，则在一行中输出“Yes”以及有多少多余的珠子；如果不可以买，则在一行中输出“No”以及缺了多少珠子。

其间以1个空格分隔。

输入样例1：

ppRYYGrrYBR2258

YrR8RrY

输出样例1：

Yes 8

输入样例2：

ppRYYGrrYB225

YrR8RrY

输出样例2：

No 2\*/

#include <stdio.h>

#include <math.h>

#include <algorithm>

#include <iostream>

#include <string>

using namespace std;

int main()

{

string a, b;//a表示现有的，b表示需要的

cin >> a >> b;

int no = 0, more = 0;

for (int i = 0; i < b.length(); i++)//需要b[i]，现有a[j]

{

if (a.find(b[i]) == string::npos)//没找到

{

no++;

}

else//找到了，避免重复查找，找到后置’.’。

{

int num = a.find(b[i]);

a[num] = '.';

}

}

//cout << a << endl << no;

int sum = 0;

if (no == 0)

{

cout << "Yes" << ' ';

for (int i = 0; i < a.length(); i++)

{

if (a[i] != '.')

sum++;

}

cout << sum;

return 0;

}

else

{

cout << "No" << ' ';

cout << no;

return 0;

}

}

算法的优化，long long数据类型的使用，解题的策略/\*(B1040.有几个PAT)字符串APPAPT中包含了两个单词“PAT”，其中第一个PAT是第2位(P),第4位(A),第6位(T)；

第二个PAT是第3位(P),第4位(A),第6位(T)。

现给定字符串，问一共可以形成多少个PAT？

输入格式：

输入只有一行，包含一个字符串，长度不超过105，只包含P、A、T三种字母。

输出格式：

在一行中输出给定字符串中包含多少个PAT。由于结果可能比较大，只输出对1000000007取余数的结果。

输入样例：

APPAPT

输出样例：

2\*/

#include <stdio.h>

#include <math.h>

#include <algorithm>

#include <iostream>

#include <string>

using namespace std;

int main()

{

string a;

cin >> a;

long long sum = 0;//错误：这里要用long long

int p1 = 0, a1 = 0, P = 0, T = 0;

/\*while (a.find('A') != string::npos) //error：这个方法超时了，每次遇见A统计P,A,T的个数，导致繁琐。所以尽量在线处理，减少循环次数。

{

int i = a.find('A');//找到A的位置

int t1 = i + 1;

a[i] = '.';

for (int j = p1; j < i; j++)

{

if (a[j] == 'P')

{

P++;

p1 = i;

}

}

T = 0;

for (int j = i+1; j < a.length(); j++)

{

if (a[j] == 'T')

{

T++;

}

}

sum += P\*T;

}

cout << sum;\*/

int count = 0;

for (int i = 0; i < a.length(); i++)//数出T的数量

{

if (a[i] == 'T')

count++;

}

T = count;

for (int i = 0; i < a.length(); i++)

{

if (a[i] == 'P')

P++;

else if (a[i] == 'T')

T--;

else if (a[i] == 'A')

{

sum += P\*T;

}

}

sum = sum % 1000000007; //错误：按题目要求要求余。

cout << sum;

return 0;

}

另一种更机智的算法：直接P,PA,PAT处理过去。

int main()

{

string a;

cin >> a;

int p = 0, pa = 0, pat = 0;

for (int i = 0; i < a.length(); i++) {

if (a[i] == 'P') {

p++;

}

else if (a[i] == 'A') {

pa += p;

}

else if (a[i] == 'T') {

pat += pa;

pat = pat % 1000000007;

}

}

cout << pat;

}

结构体数组的应用，比较简单/\*(B1041.考试座位号)每个PAT考生在参加考试时都会被分配两个座位号，一个是试机座位，一个是考试座位。正常情况下，考生在入场时

先得到试机座位号码，入座进入试机状态后，系统会显示该考生的考试座位号码，考试时考生需要换到考试座位就座。但有些考生迟到了，

试机已经结束，他们只能拿着领到的试机座位号码求助于你，从后台查出他们的考试座位号码。

输入格式：

输入第一行给出一个正整数N（<=1000），随后N行，每行给出一个考生的信息：“准考证号 试机座位号 考试座位号”。其中准考证号由14位

数字组成，座位从1到N编号。输入保证每个人的准考证号都不同，并且任何时候都不会把两个人分配到同一个座位上。

考生信息之后，给出一个正整数M（<=N），随后一行中给出M个待查询的试机座位号码，以空格分隔。

输出格式：

对应每个需要查询的试机座位号码，在一行中输出对应考生的准考证号和考试座位号码，中间用1个空格分隔。

输入样例：

4

10120150912233 2 4

10120150912119 4 1

10120150912126 1 3

10120150912002 3 2

2

3 4

输出样例：

10120150912002 2

10120150912119 1\*/

#include <stdio.h>

#include <math.h>

#include <algorithm>

#include <iostream>

#include <string>

using namespace std;

struct G

{

string num;

int kaoshi;

};

int main()

{

struct G a[1000];

int shiji[1000];

int n, m;

cin >> n;

for (int i = 0; i < n; i++)

{

cin >> a[i].num >> shiji[i] >> a[i].kaoshi ;

}

cin >> m;

int x[1000];

for (int i = 0; i < m; i++)

{

cin >> x[i];

}

for (int i = 0; i < m; i++)

{

for (int j = 0; j < n; j++)

{

if (shiji[j] == x[i])

{

cout << a[j].num << ' ' << a[j].kaoshi << endl;

break;

}

}

}

return 0;

}

ASCII码的运用，直接以ASCII码做数组的下标，注意字符串不包含空格/\*(B1042.字符统计)请编写程序，找出一段给定文字中出现最频繁的那个英文字母。

输入格式：

输入在一行中给出一个长度不超过1000的字符串。字符串由ASCII码表中任意可见字符及空格组成，至少包含1个英文字母，以

回车结束（回车不算在内）。

输出格式：

在一行中输出出现频率最高的那个英文字母及其出现次数，其间以空格分隔。如果有并列，则输出按字母序最小的那个字母。

统计时不区分大小写，输出小写字母。

输入样例：

This is a simple TEST. There ARE numbers and other symbols 1&2&3...........

输出样例：

e 7\*/

#include <stdio.h>

#include <math.h>

#include <algorithm>

#include <iostream>

#include <string>

using namespace std;

int main()

{

string b[10000], a; //字符串不包含空格，所以加了衔接环节。

int flag = 0, p = 0;

while (flag != '\n')

{

cin >> b[p];

a += b[p];

p++;

flag = getchar();

}

int x[1000] = { 0 }, max=0;

char c;

for (int i = 0; i < a.length(); i++)

{

if (a[i] <= 'Z'&&a[i] >= 'A')

{

a[i] += 32;

}

if (a[i] <= 'z'&&a[i] >= 'a')

{

x[a[i]]++; //直接以ASCII码做数组的下标

if (x[a[i]] >= max)

{

if (x[a[i]] == max) //按题目要求增加比较环节

{

if (a[i] < c)

{

max = x[a[i]];

c = a[i];

}

}

else

{

max = x[a[i]];

c = a[i];

}

}

}

}

cout << c << ' ' << max;

}

字符串的处理/\*(B1043.输出PATest)给定一个长度不超过10000的、仅由英文字母构成的字符串。请将字符重新调整顺序，按“PATestPATest....”

这样的顺序输出，并忽略其它字符。当然，六种字符的个数不一定是一样多的，若某种字符已经输出完，则余下的字符仍按PATest的

顺序打印，直到所有字符都被输出。

输入格式：

输入在一行中给出一个长度不超过10000的、仅由英文字母构成的非空字符串。

输出格式：

在一行中按题目要求输出排序后的字符串。题目保证输出非空。

输入样例：

redlesPayBestPATTopTeePHPereatitAPPT

输出样例：

PATestPATestPTetPTePePee\*/

#include <stdio.h>

#include <math.h>

#include <algorithm>

#include <iostream>

#include <string>

using namespace std;

int main()

{

string a;

cin >> a;

int P=0, A=0, T=0, e=0, s=0, t=0;

for (int i = 0; i < a.length(); i++)

{

if (a[i] == 'P')

P++;

else if (a[i] == 'A')

A++;

else if (a[i] == 'T')

T++;

else if (a[i] == 'e')

e++;

else if (a[i] == 's')

s++;

else if (a[i] == 't')

t++;

}

while (1)

{

if (P > 0)

{

cout << 'P';

P--;

}

if (A > 0)

{

cout << 'A';

A--;

}

if (T > 0)

{

cout << 'T';

T--;

}

if (e > 0)

{

cout << 'e';

e--;

}

if (s > 0)

{

cout << 's';

s--;

}

if (t > 0)

{

cout << 't';

t--;

}

if (P == 0 && A == 0 && T == 0 && e == 0 && s == 0 && t == 0)

return 0;

}

}

妈卖批，就是不知道哪错了，题目的意思是输入一个，紧接着输出一个。/\*(B1044.火星数字)火星人是以13进制计数的：

地球人的0被火星人称为tret。

地球人数字1到12的火星文分别为：jan, feb, mar, apr, may, jun, jly, aug, sep, oct, nov, dec。

火星人将进位以后的12个高位数字分别称为：tam, hel, maa, huh, tou, kes, hei, elo, syy, lok, mer, jou。

例如地球人的数字“29”翻译成火星文就是“hel mar”；而火星文“elo nov”对应地球数字“115”。为了方便交流，

请你编写程序实现地球和火星数字之间的互译。

输入格式：

输入第一行给出一个正整数N（<100），随后N行，每行给出一个[0, 169)区间内的数字 —— 或者是地球文，或者是火星文。

输出格式：

对应输入的每一行，在一行中输出翻译后的另一种语言的数字。

输入样例：

4

29

5

elo nov

tam

输出样例：

hel mar

may

115

13\*/

#include <stdio.h>

#include <math.h>

#include <algorithm>

#include <iostream>

#include <string>

using namespace std;

int main()

{

string a[13] = { "tret", "jan", "feb", "mar", "apr", "may", "jun", "jly", "aug", "sep", "oct", "nov", "dec" };//字符串数组的初始化

string b[13] = { "tret", "tam", "hel", "maa", "huh", "tou", "kes", "hei", "elo", "syy", "lok", "mer", "jou" };

int n;

string in[101];

cin >> n;

int h = 0;

int count = 0;

for (int i = 0; i < n; i++)

{

cin >> in[i];

h = getchar();

if (h == ' ')

{

string qq;

cin >> qq;

in[i] += qq;

}

if (in[i][0] < '9')//数字变火星文

{

int sum = 0;

int q = 1;

for (int j = in[i].length() - 1; j >= 0; j--)//减一，和 j >= 0 。

{

sum += (in[i][j] - '0')\*q;

q \*= 10;

}

int w[10] = { 0 }, e = 0;

while (sum > 0)

{

w[e] = sum % 13;

sum = sum / 13;

e++;//e大时存的是高位。

}

if (e == 0)

{

cout << a[0] << endl;

}

else if (e == 1)

{

cout << a[w[0]] << endl;

}

else if (e == 2)

{

if (w[0] == 0)

{

cout << b[w[1]] << endl;

}

else

cout << b[w[1]] << ' ' << a[w[0]] << endl;

}

}

else//火星文变数字

{

int sum = 0;

if (in[i].length() > 4)//两位火星文

{

for (int j = 1; j < 13; j++)//噢，这里输入tam tret先找到了tret直接sum = 0了。

{

if (in[i].find(b[j]) != string::npos)

{

sum += 13 \* j;

break;

}

}

for (int j = 0; j < 13; j++)

{

if (in[i].find(a[j]) != string::npos)

{

sum += j;

break;

}

}

cout << sum << endl;

}

else//一位火星文

{

sum = 0;

for (int j = 0; j < 13; j++)

{

if (in[i].find(a[j]) != string::npos)

{

sum += j;

break;

}

}

for (int j = 0; j < 13; j++)//高位

{

if (in[i].find(b[j]) != string::npos)

{

sum += j \* 13;

break;

}

}

cout << sum << endl;

}

}

}

return 0;

}

算法的优化，心好累，编译器不允许数组取到超限或者-1位，坑的是每行末尾都要输出空行，在0的末尾输出必须两个空行，所以注意结尾的换行。

一个高效的算法，总体思想是先把最值的数列处理好，再一遍比较。/\*(B1044.火星数字)著名的快速排序算法里有一个经典的划分过程：我们通常采用某种方法取一个元素作为主元，通过交换，把比主元小的

元素放到它的左边，比主元大的元素放到它的右边。 给定划分后的N个互不相同的正整数的排列，请问有多少个元素可能是划分前选取的主元？

例如给定N = 5, 排列是1、3、2、4、5。则：

1的左边没有元素，右边的元素都比它大，所以它可能是主元；

尽管3的左边元素都比它小，但是它右边的2它小，所以它不能是主元；

尽管2的右边元素都比它大，但其左边的3比它大，所以它不能是主元；

类似原因，4和5都可能是主元。

因此，有3个元素可能是主元。

输入格式：

输入在第1行中给出一个正整数N（<= 105）； 第2行是空格分隔的N个不同的正整数，每个数不超过109。

输出格式：

在第1行中输出有可能是主元的元素个数；在第2行中按递增顺序输出这些元素，其间以1个空格分隔，行末不得有多余空格。

输入样例：

5

1 3 2 4 5

输出样例：

3

1 4 5\*/

#include <stdio.h>

#include <math.h>

#include <algorithm>

#include <iostream>

#include <string>

using namespace std;

int main()

{

int n;

cin >> n;

int x[n];

for (int i = 0; i < n; i++)

{

cin >> x[i];

}

int leftmax = 0;

int y[n], count = 0;

for (int i = 0; i < n; i++)

{

if (x[i] > leftmax)

{

leftmax = x[i];

int c = 0;

for (int j = i + 1; j < n; j++)

{

if (x[j] < x[i])

{

c++;

break;

}

}

if (c == 0)

{

y[count++] = x[i];

}

}

else continue;

}

sort(y, y + count);

if (count == 0)

{

cout << "0" << endl;

return 0;

}

cout << count << endl;

cout << y[0];

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

{

cout << ' ' << y[i];

}

return 0;

}

高效算法：

#include <stdio.h>

#include <math.h>

#include <algorithm>

#include <iostream>

#include <string>

using namespace std;

int main()

{

int n;

cin >> n;

int x[n], leftmax[n], rightmin[n], y[n];

for (int i = 0; i < n; i++)

{

cin >> x[i];

}

int count = 0;

leftmax[0] = x[0];

rightmin[n - 1] = x[n - 1];//为避免取到数组的-1位。

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

{

leftmax[i] = max(x[i], leftmax[i - 1]);

}

for (int i = n - 2; i >= 0; i--)

{

rightmin[i] = min(x[i], rightmin[i + 1]);

}

for (int i = 0; i < n; i++)

{

if (x[i] == leftmax[i] && x[i] == rightmin[i])

{

y[count++] = x[i];

}

}

if (count == 0)

{

cout << "0" << endl << endl;//哦，这个脑残，非要我为0时输出两个换行，平常却不需要输出换行，呵呵~

return 0;

}

cout << count << endl;

cout << y[0];

for (int i = 1; i < count; i++)//可以改进一下输出方式，把y[0]也写在里面

{

cout << ' ' << y[i];

}

return 0;

}//我这里允许取数组的负一位，他不允许啊。