A 小明的压岁钱

难度	考点
2	循环

解题思路

记录每一年小明银行中钱的总数与此时他存入银行的钱的总数，注意小明前一年的利润会并入下一年的本金，但是不计入存入钱的总数。

示例程序

```
#include <stdio.h>

int main()
{
    double a;
    int b,c,n,m,i,j,t,tmax;
    scanf("%lf%d",&a,&n);//a为现在银行中钱的总数
    b=a;//b为存入的钱的总数
    for(i=1;i<=n;i++){
        scanf("%d",&c);//读入今年存入的钱
        a+=c;//a和b都要改变
        b+=c;
        tmax=0;
        scanf("%d",&m);//读入可选银行的数目
        for(j=1;j<=m;j++){
        scanf("%d",&t);
        if(t>tmax)//判断是否为最大利率
            tmax=t;
        }
        a*=(1.+0.01*tmax);
    }
    printf("%.2f",a-b);
    return 0;
}
```

B Switch真香

难度	考点
2	switch使用

解题思路

分别读取算数、计算符，以计算符为t判断依据写switchx循环，当除数为0时需另外判断。

示例代码

```
#include<stdio.h>
int main(){
    double a,b,c;
    char s;
    while (scanf("%lf",&a)!=EOF){
        scanf("%c%lf",&s,&b);
        switch(s){
            case '+':c=a+b;
                     break;
            case '-':c=a-b;
                     break;
            case '*':c=a*b;
                     break;
            case '/':if (b!=0)
                         c=a/b;
                     else{
                         printf("Wrong Input\n");
                         continue;
                     }
                     break;
        }
        printf("%.2f\n",c);
    }
    return 0;
}
```

c Random Star

难度	考点
2	循环

题目分析

比较简单的无穷级数求和&找规律&循环打印字符串题，如果大家在循环的写法上不熟练可以先在草稿纸上写出图案与行号、列号间的关系，如果那个N不会求，那数分考试要小心了（危）。

示例代码

```
#include<stdio.h>
int main()
{
    int x, n, i, j;
    while(~scanf("%d", &x)) //学了位运算应该能看懂这行啥意思了
    {
        n = 2 * x + 1; //N = ln(e^(2x+1)) = 2x+1
        for(i = 0; i < n; i++)
        {
            for(j = 0; j < n; j++)
            {
                if(i == n / 2 || j == n - i - 1 || j == i)
                    printf("*"); //根据图形的规律和对称性易得所有'*'的位置
                else
                    printf(" ");
            }
            printf("\n");//记得换行
        }
        printf("\n");//记得换行
    }
    return 0;
}
```

D 小胖吃自助

难度	考点
3	多重循环

这道题目对应着经典的"最大连续子序列和"问题，它的解决方法非常多，在这里提供三种解法。

特别注意：只有第一种暴力解法是所有人必须掌握的！法二与法三供感兴趣的同学思考学习。

法一 暴力循环法

暴力方法的思路非常清晰：我们要求最大连续子序列和，也就是所有的连续子列和中的最大值。因此我们可以枚举整个数组中全部的连续子列，对每个子列进行求和，然后记录下这些和当中的最大值。

枚举所有的连续子列，也就是枚举所有的起始位置和结束位置，我们可以采用一个二重循环来实现。对于一个已经确定好的区间$[i,j]$，可以再用一个循环去求这个下标区间内的元素和。因此这里是三重循环嵌套。

另外题目还要求在找最大连续子列和的同时，记录下这个子列的起点和终点。这里我们同样采用变量记录最大和对应的子列的起点终点，当更新最大和时顺带更新起点终点的记录。另外如果有多种子列可以达到最大和，我们要记录下长度最长的子列，也就是$j-i$最大，在这里我们需要在某个子列和与当前的最大和相等时更新记录的起点终点下标。

最后就是几件事，一是别忘了判断数组所有元素均为负的情况(最大连续子列和就是什么也不选)，二是看清数据范围，100个1e9相加已经超过 `int` 了，请用 `long long` 保存连续子列的和。

代码解释

在这里统一对代码中出现的一些变量进行说明：

- `nega`：记录整个数组的所有元素是否均为负数。其用法：一开始先假定所有元素均为负数，即令其等于1，如果遇到非负数则将其置零。
- `sum`：记录从 `i` 到 `j` 的连续子序列的和。
- `maxS`：记录连续子列和的最大值。
- `st`， `ed`：记录对应最大和的连续子列的起点下标和终点下标(两边均包含)

另外，在示例代码中更新最大值和位置记录的部分使用了*逗号运算符*，其作用是将多个表达式连接成一个表达式(可以省略大括号并且不破坏代码的可读性)，这种写法可以常用于同时更新多个变量的值。

示例代码

```c
#include <stdio.h>
int n, a[105];
int main()
{
    int i, j, k;
    long long maxS = 0, sum;
    int st, ed, nega;
    while (scanf("%d", &n) != EOF)
    {
        nega = 1;
        for (i = 1; i <= n; i++)
        {
            scanf("%d", &a[i]);
            if (a[i] >= 0) nega = 0; // 在输入过程中顺便完成对负数的判断
        }
        if (nega) printf("I\'m hungry!\n");
        else
        {
            maxS = st = ed = 0; // 将记录的最大和假定为零
            for (i = 1; i <= n; i++)
            {
                for (j = i; j <= n; j++) // 始终有 j >= i
                {
                    sum = 0; // 每次求区间和之前先将求和变量清零
                    for (k = i; k <= j; k++)
                        sum += a[k];
                    if (sum > maxS) // 更新最大和，用逗号运算符可以省掉大括号
                        maxS = sum,
                        st = i, ed = j;
                    else if (sum == maxS) // 达到最大和，此时更新区间长度
                    {
                        if (j - i > ed - st)
                            st = i, ed = j;
                    }
                }
            }
            printf("%lld %d %d\n", maxS, st, ed);
        }
    }
    return 0;
}
```

法二 前缀和法

法二和法一的本质是一样的，只是这里对数组采取了前缀和优化，从而在想要获取一个区间所有元素和时可以省去法一当中的 `for k` 这层用来求和的循环，只需一次计算即可达到。

所谓的前缀和有些类似于数分当中学的级数的"部分和"，其本质就是"前n项的和"。在以下的示例代码中， `sum` 数组就表示数组 `a` 的前缀和，其中$\forall k:1\leq k\leq n$均有 `sum[k] = a[0]+a[1]+...+a[k]` 。计算前缀和的方法为 `s[i] = s[i - 1] + a[i]` ，在遍历数组 `a` 的同时即可求出前缀和，这个过程也可以在输入过程中完成。（由于题目规定下标从1开始，所以 `a[0]` 和 `s[0]` 都是0，这样在循环 `i` 从1开始时 `s[i-1]` 直接是零， `s[1]=a[1]` ）。

在有了前缀和以后，想要查询区间$[i,j]$的所有元素和只需计算 `sum[j] - sum[i - 1]` 即可，这里直接给出结论，感兴趣的同学可以自己写几个数组尝试一下。(为什么是 `i-1` 而不是 `i`)

其他步骤和上面的代码基本类似，只是少了一层 `for` 循环，以及少了一个记录连续序列和的 `sum` 变量（但是多了记录前缀和的 `sum` 数组）

代码说明

在下面的示例代码中有一个新见到的函数 `memset` ，其用途是将某一段连续的内存按字节赋为某个值。其在本例中的应用为将代表前缀和的数组 `sum` 清零（因为前缀和是累加求出来的，新来一组数据不可能在上一组数据的基础上继续累加，因此需要重新初始化）。

函数 `memset` 一共有三个参数。对于初始化一个数组，其第一个参数为数组名，第二个参数为要赋的值(通常为0)，第三个参数一般为 `sizeof(数组名)` ，表示要连续赋值的内存长度。
例： `memset(a, 0, sizeof(a));` 。

在应用此函数时需要注意几个点：

- **非常重要**：由于 `memset` 函数是按字节直接对内存写入的，因此第二个参数通常我们只会选取 `0` 或者 `-1` ，可达到将整个数组赋值为 `0` 或 `-1` 的效果（想想上节课学过的补码， `0` 在内存中是 `00000000` ， `-1` 在内存中是 `11111111` ）。不要轻易尝试其他值(例如说1)，否则会得到错误的结果（如 `a` 是 `int` 型数组，则 `memset(a, 1, sizeof(a));` 并不能将 `a[i]` 变为1，而是变为 `0x01010101` ，也就是 `16843009` ）
- 对一个数组快速初始化为 `0` 或 `-1` 请优先选用 `memset` ，它的执行速度比 `for` 循环快得多。
- `memset` 函数位于头文件 `string.h` 中，如需要用，请 `#include <string.h>` 。

示例代码

```c
#include <stdio.h>
#include <string.h>
#define MAXN 100

int a[MAXN+5];
long long sum[MAXN+5]; // sum数组表示a数组的前缀和

int main()
{
    int n, i, j;
    int st, ed;
    long long maxS = 0;
    int nega = 1; // 所有元素是否全为负数

    while (scanf("%d", &n) != EOF)
    {
        maxS = 0; nega = 1;
        memset(sum, 0, sizeof(sum)); // 清空sum数组，也可以用下面的for循环
        /*
        for (i = 1; i <= n; i++)
            s[i] = 0;
        */
        for (i = 1; i <= n; i++)
        {
            scanf("%lld", &a[i]);
            if (a[i] >= 0) nega = 0;
            sum[i] = sum[i - 1] + a[i];
        }
        if (nega) printf("I\'m hungry!\n");
        else
        {
            for (i = 1; i <= n; i++)
            {
                for (j = i; j <= n; j++)
                {
                    if (sum[j] - sum[i - 1] > maxS)
                        maxS = sum[j] - sum[i - 1],
                        st = i, ed = j;
                    else if (sum[j] - sum[i - 1] == maxS)
                    {
                        if (j - i > ed - st)
                            st = i, ed = j;
                    }
                }
            }
            printf("%lld %d %d\n", maxS, st, ed);
        }
    }
    return 0;
}
```

法三 动态规划

这个"求最大连续子序列和"的题目有着一个简单且经典的动态规划做法。如果不知道动态规划是什么可以上网上搜索一下，其大致的思想就是将一个规模较大的问题拆分成若干个规模较小的性质类似的子问题，先解决规模小的问题进而解决规模较大的问题。用动态规划方法解题的关键在于选取合适的状态并且找到状态转移方程，通过状态转移方程递推得到问题的最优解。

在以下的例程当中， `sum[i]` 表示"以 `a[i]` 作为结尾(必须含 `a[i]`)的最大连续子序列的和"（也就是要dp的状态）。这里先直接给出结论，状态转移方程为 `sum[i] = max(sum[i - 1] + a[i], a[i])` ，我们以题目中给出的样例2来试着推出这个关系。

样例2数组共8个数，为 `[1,-9,2,6,0,-8,1,-7]` 。
首先，我们可以显而易见地分析得出 `sum[1] = a[1] = 1` ， `sum[2] = sum[1] + a[2] = -8` (表示选取 `[1,-9]` 两个元素)，接下来我们考虑 `a[3]` 也就是 `2` 。
以 `2` 结尾的连续子列一共有 `[1,-9,2]` ， `[-9,2]` 以及它自身 `[2]` 。我们可以直接排除掉 `[-9,2]` ，原因是我们已经求出了 `sum[2]` ， `sum[2] = -8` 意味着所有以 `-9` 结束的连续子列中，和最大的是 `[1,-9]` 也就是 `-8` ， `[-9]` 一定比不过 `[1,-9]` （对于 `sum[3]` 而言只需要记住 `sum[2]` 的结论，不需要关心 `sum[2]` 怎么来的）
所以我们每一步其实只有两种选择：接上前面的子列，即 `[1,-9,2]` ；或者重新开始，即 `[2]` 。
很显然在这一步，切断与前面的联系、重新开始是正确的答案，即 `sum[3] = 2` 。原因很简单，因为 `sum[2]<0` ，一个负数一定会拖累后面的求和（比如说 `[1,-9,2,6]` 一定不如 `[2,6]` 的和更大）。
接下来就是求后续的 `sum[4]` 到 `sum[8]` ，同理，如果上一个 `sum[i - 1]` 是正数，就累加(`sum[i] = sum[i - 1] + a[i]`)；如果是负数，就重新开始(`sum[i] = a[i]`)。至此我们得到了原始问题的若干个子问题，即以每个数作为结尾的最大和连续子列。

i	0	1	2	3	4	5	6	7	8
a[i]	0	1	-9	2	6	0	-8	1	-7
sum[i]	0	1	-8	2	8	8	0	1	-6

与前两种方法类似，在求出每个 `sum[i]` 的同时记录一下最大值。另外这里同样补充了边界 `a[0]=sum[0]=0` 。

到这里这道题还没有结束，我们还需要记录和最大的连续子序列的起点和终点，这个过程可以在更新最大和时完成。我们类似地令状态 `start[i]` 表示以 `a[i]` 结尾的和最大的连续子列的起点，那么更新 `sum[i]` 的同时即可更新 `start[i]` ：连接上前面的子列(`start[i]=start[i-1]`)或者从自己开始 `start[i]=i` 。这里补充一个初始条件 `start[0] = 1` ，这样可以无需特殊处理 `i=1` ，简化循环。

示例代码

```c
#include <stdio.h>
#include <string.h>

#define MAXN 100

int a[MAXN+5];
long long sum[MAXN+5]; // 记录最大连续子列和
int start[MAXN+5]; // 记录起始终止位置

int main()
{
    int n, i, j;
    int nega;
    long long maxS;
    int st, ed;
    while (scanf("%d", &n) != EOF)
    {
        memset(sum, 0, sizeof(sum));
        memset(start, 0, sizeof(start));
        nega = 1; maxS = 0;
        for (i = 1; i <= n; i++)
        {
            scanf("%d", &a[i]);
            if (a[i] >= 0) nega = 0;
        }
        if (nega) printf("I\'m hungry!\n");
        else
        {
            start[0] = 1; // 和以下for循环第一个else同理，方便递推
            for (i = 1; i <= n; i++)
            {
                // 状态转移
                if (sum[i - 1] + a[i] >= 0)
                    sum[i] = sum[i - 1] + a[i],
                    start[i] = start[i - 1];
                else
                    sum[i] = 0,
                    start[i] = i + 1; // 这样做的目的在于方便后续的递推

                // 更新最大值
                if (sum[i] > maxS)
                    maxS = sum[i], st = start[i], ed = i;
                else if (sum[i] == maxS)
                {
                    if (i - start[i] > ed - st)
                        st = start[i], ed = i;
                }
            }
            printf("%lld %d %d\n", maxS, st, ed);
        }
    }
```

```
    return 0;
}
```

E 人肉指南针

难度	考点
2	选择语句

题目分析

主要考察利用if实现不重不漏的分类讨论。根据目的地在直角坐标系的位置，可以分成9种情况：原点、正x轴、负x轴、正y轴、负y轴以及四个象限。
注意浮点型数据判断相等关系时会有误差，建议使用 `fabs(x) < eps` 语句。

示例代码

```
#include <stdio.h>
#include <math.h>

#define PI 3.14159265//圆周率，math.h中也定义了PI，我们这里和它取的不一样

int main()
{
    double x, y;
    scanf("%lf%lf", &x, &y);

    double angle = atan(y / x) * 180 / PI;//计算以角度表示的反正切

    if(fabs(x) < 1e-6 && fabs(y) < 1e-6) printf("Bingo");//原点
    else if(fabs(y) < 1e-6 && x > 0) printf("E");//正x轴
    else if(fabs(y) < 1e-6 && x < 0) printf("W");//负x轴
    else if(fabs(x) < 1e-6 && y > 0) printf("N");//正y轴
    else if(fabs(x) < 1e-6 && y < 0) printf("S");//负y轴
    else if(x > 0 && y > 0) printf("NE%.2f", 90 - angle);//第一象限
    else if(x < 0 && y > 0) printf("NW%.2f", 90 + angle);//第二象限
    else if(x < 0 && y < 0) printf("SW%.2f", 90 - angle);//第三象限
    else if(x > 0 && y < 0) printf("SE%.2f", 90 + angle);//第四象限

    return 0;
}
```

F 拔剑吧少年

难度	考点
1	数学公式

题目分析

这就是一道公式题啦，为了避免一行AC，我加了一个泰勒展开阶数的判定，但是仍然没办法卡掉打表，很无奈。注意题中是泰勒展开的阶数，不是展几项，因为数学上没有几阶马青公式这种说法，为了数学上严谨一点，采取了这种说法，真的不是为了卡大家。不过如果没看懂这块的话，应该过不了样例的。

示例代码

```
//方法一：公式法
#include<stdio.h>
double f[10000];
int main()
{
    int n,x1=5,x2=239,i;
    double ans1,ans2;
    while(scanf("%d",&n)!=EOF)
    {
        n=(n+1)/2;
        ans1=0;ans2=0;
        f[0]=1;//预处理f为5的i次方
        for(i=1;i<2*n;++i) f[i]=f[i-1]*x1;
        for(i=0;i<n;++i)
            ans1+=1.0/(f[i*2+1]*(i*2+1))*(i%2?-1:1);//计算泰勒展开
        f[0]=1;//预处理f为239的i次方
        for(i=1;i<2*n;++i) f[i]=f[i-1]*x2;
        for(i=0;i<n;++i)
            ans2+=1.0/(f[i*2+1]*(i*2+1))*(i%2?-1:1);//计算泰勒展开
        printf("%.9lf\n",16.0*ans1-4.0*ans2);
    }
    return 0;
}
```

```
//方法二：打表法（不建议，仅提供作为参考）
#include<stdio.h>
double a[7]={3.183263598,3.140597029,3.141621029,3.141591772,3.141592682,3.141592653,3.141592654
int main()
{
    int i;
    while(scanf("%d",&i)!=EOF)
        printf("%.9lf\n",i>12?a[6]:a[(i-1)/2]);
    return 0;
}
```

G 质量弹簧阻尼

难度	考点
2	循环与迭代

题目分析

本题采用数值分析法，通过初值进行有限次迭代，利用计算机的计算能力，得出在误差范围内的近似解；

第一次迭代的 $x^{(0)}$ 为输入的初始位置 x_0，$z^{(0)}$ 为初速度 0，经过迭代公式的一次迭代可得到 $x^{(1)}, z^{(1)}$，再依次迭代下去可得到方程的近似解；

关于迭代的次数，应满足 $n = \frac{t_0}{h}$，步长 $h = 1e-6, 1e-7$ 都可以达到精度要求。

示例代码

```c
#include <stdio.h>
#define H 0.000005 //每次迭代的步长

int main()
{
   double m,b,k;
   double t;
   double x0,z=0;//声明变量
   int i,n;
   double x,x1;
   double pre_x,pre_z;

   scanf("%lf%lf%lf",&m,&b,&k);
   scanf("%lf%lf",&x0,&t);
   n=(int)(t/H); //迭代次数


   x1=x0;z=0;
   pre_x=x1;pre_z=z;
   for(i=0;i<n;i++)
   {
      z=pre_z-H*(b*pre_z+k*pre_x)/m;
      x1=pre_x+H*pre_z;
      pre_z=z;pre_x=x1;
   }
   printf("%.8f\n",x1);
   return 0;
}
```

H 找零件

难度	考点
5	位运算

题目分析

本题考查位运算中异或的作用：异或运算有结合、交换律，相同的两个数异或得 0，0 与任何数异或都得原数。相同的数异或之后两两相消，留下来的就是唯一一个出现次数是奇数的数。

示例代码

```c
#include <stdio.h>
int main()
{
    int num=0,i,n,k,size;
    scanf("%d",&n);
    size=2*n-1;
    for(i=0;i<size;i++)
    {
        scanf("%d",&k);
        num=num^k;
    }
    printf("%d",num);
    return 0;
}
```

I jqe的卡组1

难度	考点
3	循环、位运算

题目分析

我们可以想到用二层循环来判断每一对整数是否相同，找到相同的就可以输出答案并退出。

也可以使用位运算(异或)。将 $n+1$ 个数异或结果与 1 到 n 异或的结果再做异或，得到的值就是出现了 2 次的数。
设出现了 2 次的数为 a，其余 $n-1$ 个数异或结果为 b。这个操作实质上是计算 $(a \wedge a \wedge b) \wedge (a \wedge b)$。由于异或运算满足交换律和结合律，且 $x \wedge x = 0$，$x \wedge 0 = x$，所以 $(a \wedge a \wedge b) \wedge (a \wedge b) = a \wedge ((a \wedge b) \wedge (a \wedge b)) = a \wedge 0 = a$

这道题还有一些其他方法，比如求 $n+1$ 个数的和再减去 $n \times (n+1)/2$、排序后再顺序查找、用数组记录每个数字出现次数等，均可通过本题。

示例代码

```c
/*二层循环*/
#include<stdio.h>

#define M 1111

int n,a[M];

int main()
{
    int i,j;
    scanf("%d",&n);
    for(i = 1;i <= n + 1;i++)
        scanf("%d",&a[i]);
    for(i = 1;i <= n;i++)
    {
        for(j = i + 1;j <= n + 1;j++)
        {
            if(a[i] == a[j])
            {
                printf("%d",a[i]);
                return 0;
            }
        }
    }
    return 0;
}


/*时间O(n)空间O(1)的位运算(异或)解法*/
#include<stdio.h>

int n,a,k = 0;

int main(int argc,char** argv)
{
    int i,j;
    scanf("%d",&n);
    for(i = 1;i <= n + 1;i++)
    {
        scanf("%d",&a);
        k ^= a;
    }
    for(i = 1;i <= n;i++)
        k ^= i;
    printf("%d",k);
    return 0;
}
```

J 1992 年 9 月 29 日

难度	考点
4	控制语句综合、日期处理

解题思路

只要根据题意进行模拟即可。另外，处理每个月份和处理闰年可以采用一些技巧加速程序编写，详见下面标程的注释。

示例程序

```c
#include <stdio.h>

// cntDay[i] 表示非闰年第 i 个月份的日期
int cndDay[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

int main()
{
    int q, yy, mm, dd, y, m, d;
    int cnt2, cnt9;
    int mUpp, dUpp; // 表示循环时的 month 上界、day 上界
    int isLeapYear;
    int date, ans;

    scanf("%d", &q);
    while (q--)
    {
        scanf("%d%d%d", &yy, &mm, &dd);
        ans = 0;
        for (y = 1; y <= yy; y++)
        {
            mUpp = 12;
            if (y == yy)
                mUpp = mm;
            isLeapYear = (y % 4 == 0 && y % 100 != 0) || (y % 400 == 0); // 判断闰年

            for (m = 1; m <= mUpp; m++)
            {
                dUpp = cndDay[m];
                if (m == 2 && isLeapYear)
                    dUpp++; // 特殊处理闰月的情况
                if (y == yy && m == mm)
                    dUpp = dd;

                for (d = 1; d <= dUpp; d++)
                {
                    date = y * 10000 + m * 100 + d; // date 是一个 yyyymmdd 形式的八位数
                    cnt2 = 0, cnt9 = 0;
                    while (date)
                    {
                        if (date % 10 == 2)
                            cnt2++;
                        if (date % 10 == 9)
                            cnt9++;
                        date /= 10;
                    }
                    if (cnt2 && cnt9)
                        ans++;
                }
            }
        }
```

```
        printf("%d\n", ans);
    }

    return 0;
}
```

K 摔跤比赛

难度	考点
3	暴力枚举

题目解析

改编自PPT中乒乓球赛和最大公约数。将参赛人数扩充为了4对，故要有四重循环，判断是否有一个队员和同时和多个队员比赛的情况也要相应修改。另外求最大公约数是不要忘记判0和负数。

示例代码

```c
#include <stdio.h>

int main()
{
    int a[4], b[4], flag, x, y, r, judge, i, j, k, l;
    while (~scanf("%d%d%d%d%d%d%d%d", &a[0], &a[1], &a[2], &a[3], &b[0], &b[1], &b[2], &b[3]))
    {
        flag = 0;
        for (i = 0; i < 4; i++)
        {
            if (a[i] == 1 || b[i] == 1 || a[i] == -1 || b[i] == -1)
            {
                flag = 1;
            }
        }
        if (flag == 1)
        {
            printf("As we can't.\n");
            continue;
        }
        for (i = 0; i < 4; i++)
        {
            for (j = 0; j < 4; j++)
            {
                for (k = 0; k < 4; k++)
                {
                    for (l = 0; l < 4; l++)
                    {
                        if (i == j || i == k || i == l || j == k || j == l || k == l)
                        {
                            continue;
                        }
                        if (a[0] != 0 && b[i] != 0)
                        {
                            x = a[0];
                            y = b[i];
                            for (r = x % y; r != 0; r = x % y)
                            {
                                x = y;
                                y = r;
                            }
                            judge = y < 0 ? -y : y;
                            if (judge == 1)
                            {
                                continue;
                            }
                        }
                        if (a[1] != 0 && b[j] != 0)
                        {
                            x = a[1];
                            y = b[j];
```

```
                    for (r = x % y; r != 0; r = x % y)
                    {
                        x = y;
                        y = r;
                    }
                    judge = y < 0 ? -y : y;
                    if (judge == 1)
                    {
                        continue;
                    }
                }
                if (a[2] != 0 && b[k] != 0)
                {
                    x = a[2];
                    y = b[k];
                    for (r = x % y; r != 0; r = x % y)
                    {
                        x = y;
                        y = r;
                    }
                    judge = y < 0 ? -y : y;
                    if (judge == 1)
                    {
                        continue;
                    }
                }
                if (a[3] != 0 && b[l] != 0)
                {
                    x = a[3];
                    y = b[l];
                    for (r = x % y; r != 0; r = x % y)
                    {
                        x = y;
                        y = r;
                    }
                    judge = y < 0 ? -y : y;
                    if (judge == 1)
                    {
                        continue;
                    }
                }
                flag = 1;
                printf("%dvs%d\n%dvs%d\n%dvs%d\n%dvs%d\n", a[0], b[i], a[1], b[j], a[2],
            }
        }
    }
}
if (flag == 0)
{
    printf("As we can't.\n");
}
```

```
    }
    return 0;
}
```

L 卖口罩

难度	考点
4	简单贪心

题目分析

简单的贪心，优先使用2元进行找零，大家都不可以插队，依次往下判断就行。

示例代码

```c
#include <stdio.h>
int q[100007];

int main() {

    int n;
    while(~scanf("%d", &n))
    {
        int a = 0, b = 0, m, i; //a:1元存量 b:2元存量 m:当前顾客随身携带的金额数
        for (i = 0; i < n; i++)
        {
            scanf("%d", &q[i]);
        }
        for (i = 0; i < n; i++)
        {
            m = q[i];
            if (m == 1)
            {
                a++;
            }
            else if (m == 2)
            {
                b++;
                a -= 1;
                if (a < 0)
                {
                    break;
                }
            }
            else
            {
                m--;
                while (m >= 2 && b > 0) //优先使用两元找零
                {
                    b--;
                    m -= 2;
                }
                while (m >= 1) {
                    a--;
                    m -= 1;
                }
                if (a < 0)
                {
                    break;
                }
            }
        }

        if (i == n)
        {
            printf("Survived.\n");
```

```
        }
        else
        {
            printf("Bankrupted.\n");
        }
    }
    return 0;
}
```