E1 solution

A 简单的计算器

题目解读

实现一个 ~~输出更友好的~~ 计算器

思路

可以发现 op 决定了输出时的符号和进行的计算，则可以通过分支语句对 op 的值进行判断，然后分别输出相应的语句即可

代码结构

```
if(op==0)printf("%d + %d = %d",a,b,a+b);
else if(op==1)printf("%d - %d = %d",a,b,a-b);
else if(op==2)printf("%d * %d = %d",a,b,a*b);
else if(op==3)printf("%d / %d = %d",a,b,a/b);
```

B ljh想对齐

难度	考点
1	输出

问题分析

题目要求我们输入六个整数，并按照特定宽度输出每个整数，同时还有一次换行操作。

有**HINT**可知，我们可以直接通过 `%6d` 的方式实现特定宽度整数的输出，后面的工作就很容易了。

参考代码

```
#include<stdio.h>
int main()
{
    int a,b,c,d,e,f; //定义六个int类型的变量（整型变量）
    scanf("%d%d%d%d%d%d",&a,&b,&c,&d,&e,&f); //依次输入这六个整数
    printf("%6d %6d %6d\n%6d %6d %6d",a,b,c,d,e,f);
    //依次输出，每个整数后有一个空格，同时第三个整数后为换行符 \n
    return 0;
}
```

C 山童的黑市

难度	考点
1	输入，计算

问题分析

题意较好理解，我们需要先读入第一行的四个整数，然后再读入每行的一个字符和一个整数共 n 次，用 if-else 或 switch 的结构判断商品的类型，然后根据题目描述进行计算即可。

本题唯一需要注意的一个小知识点是字符读入的问题，在 hint 中已经给出详细解释。

参考代码

```
#include <stdio.h>

int main()
{
    int n, a1, a2, a3, ans = 0, t;
    char ch;
    scanf("%d%d%d%d", &n, &a1, &a2, &a3);
    while (n--)
    {
        scanf(" %c%d", &ch, &t);
        if (ch == 'G')
            ans += a1 * t;
        else if (ch == 'L')
            ans += a2 * t;
        else
            ans += a3 * t;
        }
    }
    printf("%d", ans);
    return 0;
}
```

D 派蒙喜欢吃日落果

问题分析

题目还是挺简单的，但是要注意**细节**!

首先，派蒙吃了s/t个日落果，那么问题就来了，第一个坑：如果t是0，那么派蒙能以迅雷不及掩耳之势吃完所有日落果，输出0。但是0又不能做除数，这种情况要特判。

接着往下想，如果ss是tt的整数倍，那么派蒙就吃了s/t个完整的日落果，剩下$m-s/t$个完整的日落果；如果s并不是t的整数倍，派蒙就吃了s/t个完整的日落果和一个残缺的日落果，那么就有$s/t+1$个日落果不完整了，剩下$m-s/t-1$个日落果。可是，第二个坑：说不定她在s秒内就吃完了这堆日落果了呢，那么就没有剩下的日落果，输出0。

参考代码

```
#include<stdio.h>
int main()
{
    int m,t,s;
    scanf("%d%d%d",&m,&t,&s);
    if(t==0){
        printf("0");//t=0，派蒙能光速吃完，一个不剩，但0不能做除数，需要特判。
    }
    else if(s%t==0){//判断s是不是t的整数倍
        int ans = m-s/t;
        if(ans < 0) ans=0;//判断s时间内能不能吃完，能就输出0，否则输出剩下的
        printf("%d",ans);
    }
    else{//s不是t的整数倍，会有一个没吃完的日落果
        int ans = m-s/t-1 > 0 ? m-s/t-1 : 0;//判断s时间内能不能吃的只剩下0个完整的日落
果。
        printf("%d",ans);
    }
        return 0;
}
```

E 空白的黑白棋-1

题目解读

输入三个 $[1, 13]$ 内的范围，保证互不包含，求数 x 是否在某个范围内，或者不被任何范围包含。

解题思路

使用if-else结构依次对输入的范围进行判断，如果三次判断都判断不在范围内，else中输出None。

代码结构

首先对输入数据进行读入，定义六个指定范围的变量分别为 $x_1, x_2, x_3, x_4, x_5, x_6$ ，定义三个范围对应的标记数为 $s_1, s_2. s_3$，求解的数为 x 。

注：这里scanf的部分双引号内不需要额外打空格，只读入数字的时候scanf会自动按照空格分割输入内容。

```
int x1,x2,x3,x4,x5,x6,x,s1,s2,s3;
scanf("%d%d%d%d%d%d%d%d%d%d",
       &x1,&x2,&s1,&x3,&x4,&s2,&x5,&x6,&s3,&x);
```

使用if-else结构依次对输入的范围进行判断，如果三次判断都判断不在范围内，else中输出None。

```
if (x >= x1 && x <= x2){
    printf("%d",s1);
}
else if (x >= x3 && x <= x4){
    printf("%d",s2);
}
else if (x >= x5 && x <= x6){
    printf("%d",s3);
}
else {
    printf("None");
}
```

F O! 平均分鸽

难度	考点
1	数学计算，循环

问题分析

题意要求在输入的数中找到最大的两个数和最小的一个数。我们可以预定义出最大最小变量，利用 **"擂台法"** 找出最大最小的值，对最大的两数求和，除以最小的一个数，具体步骤如下代码所示

题目中的数据范围限制在 `int` 中，但可能出现除数为 `0` 的情况，因而需要考虑特殊情况，使用条件判断

`oh O!` 请复制粘贴，不要自己输入

参考代码

```
#include <stdio.h>

int main()
{   //由于我们只需要利用最大最小的值，故可以使用“擂台法”：
    //预定义出变量，并于循环中不断更新。

    //定义变量
    int max1, max2, min, num;
    //初始化，确保max1, max2初始值最小，min初始值最大
    max1 = -1;
    max2 = -1;
    min = 2147483647;
    //读入
    scanf("%d", &num);
    //循环
    while (num != -1)
    {
        //读入的数比当前第一最大值大，则替换第一最大值并更新第二最大值
        if (num >= max1)
        {
            max2 = max1;
```

```
            max1 = num;
        } //读入的数仅仅比当前第二最大值大，则替换第二最大值
        else if (num >= max2)
        {
            max2 = num;
        }
        //读入的数比当前最小值小，则替换最小值
        if (num <= min)
        {
            min = num;
        }
        //继续读取确保更新循环
        scanf("%d", &num);
    }
    //容错处理，确保除数非0
    if (min == 0)
    {
        printf("oh o!");
    }
    else
    {
        printf("%d", (max1 + max2) / min);
    }
    return 0;
}
```

G 字符处理机器

难度	考点
2	分支结构，输出

问题分析

本质上是对每一个字符进行分类处理。需要注意的三点：

- 当遇到数字时需要对数字进行储存，在遇到英文字母后又要对储存的数字进行“清除”，如果连续遇到数字还需要对储存的数字进行更新。
- 输出转义字符时需要在该字符前添加`\`。
- 读入第一个字符前需要先处理整数`n`后的空格。

参考代码

```
#include <stdio.h>

int main() {
    int n, i, j;// i记录输出次数
    char c;
    scanf("%d", &n);
    while (n--) {
        scanf(" %c", &c);
        if (c >= '0' && c <= '9')
```

```
            i = c - '0';
        else if (c >= 'A' && c <= 'Z')
            c = c - 'A' + 'a';
        else if (c >= 'a' && c <= 'z')
            c = c - 'a' + 'A';
        else {
            printf("? *&\\! _//\\a@\\\\\\r\\n! //\\\\"_\"/\\\\^! ~zZ\n");
        }
        if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z')) {
            for (j = 0; j < i; j++)
                printf("%c", c);
            printf("\n");
            i = 1;
        }
    }
    return 0;
}
```

H 用代码来写歌！

题目分析

这道题本质上就是通过 p 计算出 f，再通过 l 和 s 计算出 t，然后循环打印即可。

通过 p 计算 f 时，可以使用判断，也可以使用数组。

示例代码

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


int main() {
    printf("#include<stdio.h>\n#include<stdlib.h>\n\nint main(){\n");
    int n, s;
    scanf("%d%d", &n, &s);
    for (int i = 0;i < n;i++) {
        int p, l, f;
        scanf("%d%d", &p, &l);
        if(p==0) f=0;
        else if(p==1) f=523;
        else if(p==2) f=587;
        else if(p==3) f=659;
        else if(p==4) f=698;
        else if(p==5) f=784;
        else if(p==6) f=880;
        else if(p==7) f=988;
        printf("    _beep(%d, %d);\n", f, l * s);
    }
    printf("}");
}
```

I 组三角形

难度	考点
2	博弈论

问题分析

若总计有偶数条木棒，先手显然可以保证长度为 1 ~ $(\frac{n}{2}-1)$ 的木棒被取完，这时显然可以拼成三角形。

若总计有奇数条木棒，称长度为1到$\frac{n+1}{2}$的木棒为短木棒，其余为长木棒。先手取长木棒，后手就拿短木棒中最长的；先手取短木棒，后手就拿长木棒中最短的。这样，最后会剩下两根短棒，一根长棒，且最长棒与最短棒长度差一定大于$\frac{n+1}{2}$，可以保证无法拼成三角形。

于是我们只要判断 n 为奇数还是偶数，再输出即可。

参考代码

```
#include <stdio.h>
int main()
{
    int n, t;
    scanf("%d", &t);
    while (t--) //重复t次读入数据并处理
    {
        scanf("%d", &n);
        if (n % 2 == 1)//判断奇偶，奇数则输出No，偶数则输出Yes
            printf("No\n");
        else
            printf("Yes\n");
    }
    return 0;
}
```

J 有理数

题意解释与分析

通过观察题意，可以注意到每一个对角线上的分子分母的和是一致的，我们记这个和为 s，题目的要求的顺序正好是按照 s 的大小为第一关键字排列的，所以我们将 s 放在第一层循环。

对于第二层循环，依据题意，每个对角线内部，分母从 1 到 $s-1$ 递增，所以把分母 j 作为第二层循环变量。那么分子 $i=s-j$ 就自然而然了。

至此，我们构建出参考代码：

```
int a, b, x = 0, flag = 0;
scanf("%d%d", &a, &b);
for(int s = 2; ; s++) // s 表示 i+j 的和
```

```
{
    for(int j = 1; j < s; j++)
    {
        int i = s - j; // 遍历到分数 i/j
        int g; // g = gcd(i,j)

        // do something

        if(g == 1) x++; // 说明这是一个符合要求的分数
        if(i == a && j == b) // 找到了!
        {
            flag = 1; break; // flag置1，退出循环
        }
    }
    if(flag) break;
}
printf("%d\n", x);
```

代码中的`// do something`处显然应该用于计算 i 和 j 的*最大公因数*即 gcd。计算 gcd 有多种做法。当然最简单的，可以考虑使用ppt中的代码：

```
int g = i;
if(j < i) g = j;
while((i % g != 0) || (j % g != 0)) g--; // 使用课件上的方式求gcd
```

当然ppt上也给出了一个思考题：有没有什么更快速的方法计算 gcd 呢？当然有。可以考虑辗转相减法，不过辗转相减法的速度也不是特别快，于是进一步的考虑辗转相除法，不理解这两个算法的同学可以点击链接查看百度百科。于是我们有以下代码：

```
int ii = i, jj = j; // 备份 i 和 j 用来算 gcd
int g = ii; // 储存 gcd(i,j)
while(jj) // 当 jj == 0 时 有 gcd = ii 成立，故退出循环
{
    // 辗转相除法
    g = jj;
    jj = ii % jj;
    ii = g;
}
```

参考代码

40分参考代码

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

int main()
{
    int a, b, x = 0, flag = 0;
    scanf("%d%d", &a, &b);
    for(int s = 2; ; s++) // s 表示 i+j 的和
    {
```

```
        for(int j = 1; j < s; j++)
        {
            int i = s - j; // 遍历到分数 \frac{i}{j}
            int g = i;
            if(j < i) g = j;
            while((i % g != 0) || (j % g != 0)) g--; // 使用课件上的方式求gcd
            if(g == 1) x++; // 符合要求的分数，计数
            if(i == a && j == b) // 找到了！
            {
                flag = 1; break; // flag置1，退出循环
            }
        }
        if(flag) break; // flag置1，退出循环
    }
    printf("%d\n", x);
    return 0;
}
```

80分参考代码

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

int main()
{
    int a, b, x = 0, flag = 0;
    scanf("%d%d", &a, &b);
    for(int s = 2; ; s++) // s 表示 i+j 的和
    {
        for(int j = 1; j < s; j++)
        {
            int i = s - j; // 遍历到分数 \frac{i}{j}
            int ii = i, jj = j; // 备份 i 和 j 用来算 gcd
            int g = ii; // 储存 gcd(i,j)
            while(jj) // 当 jj == 0 时 有 gcd = ii 成立，故退出循环
            {
                // 辗转相除法
                g = jj;
                jj = ii % jj;
                ii = g;
            }
            if(g == 1) x++; // 说明这是一个符合要求的分数
            if(i == a && j == b) // 找到了！
            {
                flag = 1; break; // flag置1，退出循环
            }
        }
        if(flag) break; // flag置1，退出循环
    }
    printf("%d\n", x);
    return 0;
}
```

更深入的题目解析

前言

80分拿到，说明你不仅具备了程序设计入门的水平，也具备了必备的思维能力。

如果你是0基础入门，并且在不复制粘贴参考代码的情况下，独立获得本题的80分，说明的具有编程的天赋，假以时日，必将超过有基础的同学。

如果你觉得学有余力，或者只是单纯想要了解数论知识，我都建议你将题解继续看下去，当然如果你没有编程基础，可能无法写出最终的代码，但是接下来的分析，也将对你的思维的开拓有所帮助。

我们需要什么

由于满足 $a+b \leq 10^6$ 的分数，在题目给出的矩阵中，大约 $\frac{(a+b)^2}{2}$ 个，这样的数量级，是不可能通过遍历在比较短的时间内完成的。所以我们需要更快的方法。

如果我们可以快速计算出一个对角线的 $s-1$ 个数（我们沿用之前的描述，记一个对角线上的数分子分母的和为 s ）中有多少个数是既约分数（也就是 $gcd(a,b)=1$ 或称 a 与 b 互质），那么就可以快速的计算出前 n 个对角线中有多少个既约分数了，也就可以快速求出当前分数的编号了。

用数学一点的话来说，就是：对于给定的正整数 s $(s \geq 2)$，正整数 a 从 1 取到 $s-1$，满足 a 与 $s-a$ 互质的 a 的个数。

逆用辗转相减法，我们有 $\gcd(a, s-a) = \gcd(a, s)$，于是题目变为求在所有小于 s 的正整数中，有多少个正整数与 s 互质。这个题目答案有一个名字，叫做欧拉函数，符号是 $\phi(s)$，表示“所有不超过 s 的正整数中，与 s 互质的数的个数”。显然，只要 $s \neq 1$，“不超过”和“小于”对答案就没有影响（因为一个大于 1 的数不可能与自己互质）。

换句话说，我们只需要求出 $\phi(s)$。

欧拉函数与素数筛

可以证明，欧拉函数 $\phi(s)$ 的计算如下：

$$\phi(s) = s\prod_{p|s}^{p\text{是质数}}(1-\frac{1}{p})$$

由于这个证明全是数论，所以就不写在题解里了，有兴趣的同学参考：[欧拉函数]

既然欧拉函数于素数密切相关，所以我们需要快速选出素数，因此有必要介绍素数筛。*注意：素数筛在程设后面的学习中会提到，或者在某些题中可能可以用到，如果没有基础，可以只看思路，实现可以以后在看。*

埃拉托斯特尼筛法

埃拉托斯特尼筛法，简称**埃氏筛**，原理是从 2 开始遍历，每次遍历的时候，把当前数的倍数筛掉（即标记为不是素数），并且如果当前数之前没有被筛掉，那么当前数是素数。代码实现如下：

```
int notPrime[N+1]; // N 是需要筛的数的个数, notPrime[i] = 1 表示 i 不是素数
int prime[N], cnt = 0; // prime 用来储存素数, cnt 用来记录素数个数
for(int i = 2; i <= N; i++) notPrime[i] = 0; // 初始化为都是素数

for(int i = 2; i <= N; i++)   // 遍历 2 到 N 里面所有数
    if(!notPrime[i])      // 如果 i 没有被筛掉
    {
        prime[cnt++] = i; // 把素数记录下来
        // i 是素数的话, 那么 i 的倍数肯定就不是素数
        for(int j = 2; j <= n / i; j++)
            // i*j 是 i 的倍数, 肯定不是素数, 将 i*j 筛掉
            notPrime[i*j] = 1;
    }
```

欧拉筛法

欧拉筛法，简称**欧氏筛**，速度会比**埃氏筛**更快，当然理解起来就更加的复杂。**埃氏筛**不够快的原因是，对于一个数，比如 40，由于 $40 = 2 \times 20$ 而且 $40 = 5 \times 8$，所以会被筛掉两次，然而我们只希望它筛掉一次就够了，因此筛掉多次这种情况相当于做了一部分无用功。为了解决这个问题，我们只希望一个合数，只被它最小的质因子筛掉，由于一个数的最小质因子有且仅有一个，这样就不可能出现无用功了。

因此，考虑对于每个数 x，筛掉 px，其中 p 是质数，且不超过 x 的最小质因子。这样就保证了 p 是 px 的最小质因子，保证了 px 只会被 p 筛掉。代码实现如下：

```
int prime[N], cnt, notPrime[N+1]; // 意义同上
for(int i = 2; i < N; i++)
{
    if(!notPrime[i]) prime[cnt++] = i; // 记录素数
    for(int j = 0; j < cnt && prime[j] <= n / i; j++) // j 是一个编号而已, prime[j] 才是上文说的 p
    {
        not_prime[i*prime[j]] = 1; // 筛掉
        if(i % prime[j] == 0) break; // 如果 prime[j] | i, 说明这是 i 的最小质因子, 更大的素数就不需要了
    }
}
```

计算欧拉函数

埃氏筛

对于埃氏筛，显然每个数都会被自己的每一个质因子筛掉一遍，正好和欧拉函数计算切合，所以代码可用这样写：

```
int notPrime[N+1];
int prime[N], cnt = 0;
int phi[N+1]; // 记录欧拉函数的值
for(int i = 2; i <= N; i++) notPrime[i] = 0;
for(int i = 2; i <= N; i++) notPrime[i] = i; // 初始化为自己

for(int i = 2; i <= N; i++)
    if(!notPrime[i])
    {
```

```
        prime[cnt++] = i;
        for(int j = 2; j <= n / i; j++)
        {
            notPrime[i*j] = 1;
            phi[i*j] = phi[i*j] / p * (p-1); // 套用公式
        }
    }
```

欧氏筛

对于欧氏筛，略微有些复杂。根据公式，可以得到以下两个规则：

$$\text{如果 } p \text{ 不是 } x \text{ 的因子：} \phi(px) = px(1-\frac{1}{p})\prod_{q|x}^{q\text{是质数}}(1-\frac{1}{q}) = (p-1)\phi(x)$$

$$\text{如果 } p \text{ 是 } x \text{ 的因子：} \phi(px) = px\prod_{q|x}^{q\text{是质数}}(1-\frac{1}{q}) = p\phi(x)$$

于是有以下代码：

```
int prime[N], cnt, notPrime[N+1], phi[N+1]; // 意义同上
for(int i = 2; i < N; i++)
{
    if(!notPrime[i])
    {
        prime[cnt++] = i; // 记录素数
        phi[i] = i-1; // 素数的 phi 值肯定为 自己-1
    }
    for(int j = 0; j < cnt && prime[j] <= n / i; j++) // j 是一个编号而已，prime[j]
才是上文说的 p
    {
        not_prime[i*prime[j]] = 1; // 筛掉
        if(i % prime[j] == 0)
        {
            phi[i*prime[j]] = phi[i] * prime[j]; // phi(pi) = phi(i) * p
            break; // 欧拉筛的操作
        }
        else phi[i*prime[j]] = phi[i] * (prime[j] - 1); // phi(pi) = phi(i) *
(p-1)
    }
}
```

统计答案

最后的最后，对于给定的 a 和 b，我们只需要首先计算 $\sum_{i=2}^{a+b-1}\phi(i)$ 的值，然后再从 $\frac{a+b-1}{1}$ 开始遍历 $\frac{a}{b}$ 所在对角线的分数即可。

100分参考代码

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

#define N 1000002
```

```
// N 是需要筛的数的个数，not_prime[i] = 1 表示 i 不是素数
// prime 用来储存素数，cnt 用来记录素数个数
// phi 是欧拉函数的值
int prime[N], cnt;
int not_prime[N], phi[N];

int a, b;
long long ans; // 这里 最后一个测试点 的答案会超过 int 的表示范围，请使用 long long 或者
unsigned int
int gcd(int, int);

int main()
{
    scanf("%d%d", &a, &b);
    int n = a + b; phi[1] = 1;
    for(int i = 2; i < n; i++) // 欧拉筛，详细讲解请参考上文
    {
        if(!not_prime[i])
        {
            phi[i] = i - 1; // 素数的 phi 值肯定为 自己-1
            prime[cnt++] = i; // 记录素数
        }
        ans += phi[i]; // 统计答案
        for(int j = 0; j < cnt; j++)
        {
            if(i * 1ll * prime[j] >= n) break;
            not_prime[i*prime[j]] = 1; // 筛掉
            if(i % prime[j] == 0)
            {
                phi[i*prime[j]] = phi[i] * prime[j]; // phi(pi) = phi(i) * p
                break; // 如果 prime[j] | i，说明这是 i 的最小质因子，更大的素数就不需要
了
            }
            else phi[i*prime[j]] = phi[i] * phi[prime[j]]; // phi(pi) = phi(i) *
(p-1)
        }
    }
    for(int j = 1; j < n; j++) // 和前面的代码一样，只不过不需要遍历 s
    {
        int i = n - j;
        int g = gcd(i, j);
        if(g == 1) ans++;
        if(a == i && b == j) break; // 没有外层循环也就不需要 flag 了
    }
    printf("%lld\n", ans); // 输出答案
    return 0;
}

int gcd(int x, int y) // 递归的 gcd
{
    if(!y) return x;
    return gcd(y, x % y);
}
```