E7 - Solution

A 解锁战利品

难度	考点
1	回文串判断，字符比较与计数

题目分析

根据题目要求，需要使用不定组输入语句读入多组字符串。对每组输入的字符串`str`，同时遍历它前一半与后一半的位置，比较对称位置上的字符是否相同，记录出现字符不同的情况的次数。若遍历完后记录的次数为`0`，说明字符串`str`是回文串，否则不是。按题目要求输出内容即可。

示例代码

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

//返回字符串s中对称位置上字符不相同的情况出现的次数，次数为0表示s是回文串
int checkString(const char s[]) {
    int len = (int) strlen(s);
    //strlen函数的返回值是size_t类型，建议将结果转换为int类型再赋值给int变量
    int cnt = 0;
    for (int i = 0, j = len - 1; i < j; i++, j--) { //用两个循环变量i,j实现遍历对称位
置
        if (s[i] != s[j]) {
            cnt++;
        }
    }
    return cnt;
}

int main() {
    char str[1001] = {};
    while (scanf("%s", str) != EOF) {
        int cnt = checkString(str);
        if (cnt == 0) {
            printf("yes\n");
        } else {
            printf("no %d\n", cnt);
        }
    }

    return 0;
}
```

B 小亮学矩阵乘法

难度	考点
2	二维数组 循环

题目分析

根据线性代数所学的知识，对于 m 行 n 列的矩阵 A 和 n 行 k 列的矩阵 B

$$A=\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \qquad B=\begin{pmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nk} \end{pmatrix}$$

设 $A\times B$ 的结果为矩阵 C，则 $C=\begin{pmatrix} c_{11} & \cdots & c_{1k} \\ \vdots & \ddots & \vdots \\ c_{m1} & \cdots & c_{mk} \end{pmatrix}$

其中，$c_{ij}=\sum_{l=1}^{n} a_{il}b_{lj},\ \ i=1,\cdots,m,\ \ j=1,\cdots,k$。

需要注意的是计算结果可能会超出 `int` 范围，需要保证计算过程和输出数组都需要在 `long long` 范围内。

示例代码

```
#include <stdio.h>
int main()
{
    int a[50][50]={0};
    int b[50][50]={0};
    long long c[50][50]={0};
    int m,n,k;
    scanf("%d%d%d",&m,&n,&k);
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            scanf("%d", &a[i][j]);
        }
    }
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < k; ++j) {
            scanf("%d", &b[i][j]);
        }
    }

    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < k; ++j) { //计算cij
            for (int l = 0; l < n; ++l) {
                c[i][j] += (1ll * a[i][l] * b[l][j]); //注意计算范围是long long范围
            }
        }
    }

    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < k; ++j) {
```

```
            printf("%lld ",c[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

C 画画字符画画

难度	考点
2	转义字符

问题分析

对于一个字符画，我们可以选择用字符数组存储每行，然后用 `puts()` 或 `printf()` 逐行输出。但要注意，由于转义字符的存在，我们需要对其进行特别处理。

由于可见字符含有空格，所以使用 `scanf("%s", ...)` 读入字符串可能并不合适，因为这样无法区分空格和换行符。

当使用 `gets()` 读入时，注意字符串末尾的换行符会被替换为 `\0`，也就是没有存储换行。

当使用 `puts()` 输出时，会自动在字符串末尾加上 `\n`，而 `printf()` 则不会，使用时要注意区分。

对于 `puts()` 函数，我们需要特判 `\`、`'` 和 `"`。

参考代码 #1

```c
#include <stdio.h>

int main()
{
    char s[500];
    printf("#include<stdio.h>\nint main()\n{\n");
    while (gets(s))
    {
        printf("    puts(\"");
        for (int i = 0; s[i]; i++)
        {
            if (s[i] == '\\' || s[i] == '\'' || s[i] == '\"')
                printf("\\");
            putchar(s[i]);
        }
        printf("\");\n");
    }
    printf("    return 0;\n}");
    return 0;
}
```

对于 `printf()` 函数，我们还需要特判 `%`。

参考代码 #2

```
#include <stdio.h>

int main()
{
    char s[505];
    printf("#include<stdio.h>\nint main()\n{\n");
    while (gets(s))
    {
        printf("    printf(\"");
        for (int i = 0; s[i]; i++)
        {
            if (s[i] == '\\' || s[i] == '\'' || s[i] == '\"')
                printf("\\");
            else if (s[i] == '%')
                printf("%%");
            putchar(s[i]);
        }
        printf("\\n\");\n");
    }
    printf("    return 0;\n}");
    return 0;
}
```

D 自定义格式化

难度	考点
2	格式化字符串，库函数 `sscanf`，`sprintf`

题目分析

思路一：按照Hint中给出的思路，善用 `sscanf` 和 `sprintf`，补全代码模板即可，具体见示例代码。

思路二：和思路一基本一致，但是可以利用 `printf` 的一个技巧更简单，具体见示例代码。

本题旨在让大家学会使用 `sscanf` 和 `sprintf` 。

示例代码 1

```
#include <stdio.h>
int main()
{
    char str[1005];
    char s[1005];
    char format_s[105];
    char format_p[105];
    int k, x;
    gets(str);
    gets(format_s);
    scanf("%d%d", &k, &x);
```

```
    sscanf(str, format_s, s);
    sprintf(format_p, "%%%d.%ds", k, x);
    printf(format_p, s);
    return 0;
}
```

示例代码 2

```
#include <stdio.h>
int main()
{
    char str[1005];
    char s[1005];
    char format[105];
    int k, x;
    gets(str);
    gets(format);
    scanf("%d%d", &k, &x);
    sscanf(str, format, s);
    printf("%*.*s", k, x, s);
    return 0;
}
```

E 摩卡与数独

难度	考点
4	二维数组

题目分析

本题就是判断一个数独是否是正确的数独局面，正确的做法是判断每一行、每一列、每一宫是不是出现且只出现一次 1 - 9，有如下几种常见的错误做法：

- 判断每一行、每一列、每一宫的和都是 45
- **判断每一行、每一列、每一宫的和都是 45 且积是 362880**（这个也是错的，因为符合这两个条件的九个数除了 `1 2 3 4 5 6 7 8 9` 外还有 `1 2 4 4 4 5 7 9 9` 如果你真的这么做了即使 AC 了也请再好好看看下面的代码）

  这两种情况的一组 hack 数据如下：

```
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```

```
1 2 4 4 4 5 7 9 9
4 4 5 7 9 9 1 2 4
7 9 9 1 2 4 4 4 5
2 4 4 4 5 7 9 9 1
4 5 7 9 9 1 2 4 4
9 9 1 2 4 4 4 5 7
4 4 4 5 7 9 9 1 2
5 7 9 9 1 2 4 4 4
9 1 2 4 4 4 5 7 9
```

- 没有跟原数组进行比对：

  这种情况的一种 hack 数据如下：

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

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

示例代码

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

int a[10][10];  // 表示读进来的初始数独局面
int b[10][10];  // 表示填完的数独局面
int vis[10];    // 表示某个数字是否出现过

void clear() {memset(vis,0,sizeof(vis));}
int main()
{
    int t;
    scanf("%d",&t);

    while(t--)
    {
        for(int i = 1;i <= 9;i++) {
            for(int j = 1;j <= 9;j++){
                scanf("%d",&a[i][j]);
            }
```

```
        }

        for(int i = 1;i <= 9;i++) {
            for(int j = 1;j <= 9;j++){
                scanf("%d",&b[i][j]);
            }
        }

        int flag = 1;  // 用 flag 标记是不是正确的数独

        // 检查是否符合原数独局面
        for(int i = 1;i <= 9;i++) {
            for(int j = 1;j <= 9;j++) {
                if(a[i][j] != 0 && a[i][j] != b[i][j])
                    flag = 0;
            }
        }

        // 检查每一行
        for(int i = 1;i <= 9;i++) {
            clear();
            for(int j = 1;j <= 9;j++) {
                vis[b[i][j]] = 1;
            }
            for(int j = 1;j <= 9;j++) {
                if(!vis[j]) {
                    flag = 0;
                }
            }
        }

        // 检查每一列
        for(int j = 1;j <= 9;j++) {
            clear();
            for(int i = 1;i <= 9;i++) {
                vis[b[i][j]] = 1;
            }
            for(int i = 1;i <= 9;i++) {
                if(!vis[i]) {
                    flag = 0;
                }
            }
        }

        // 检查每一宫
        for(int k = 1;k <= 9;k++) {
            clear();
            int x = ((k-1)/3) * 3 + 1;
            int y = ((k-1)%3) * 3 + 1;
            for(int i = 1;i <= 3;i++) {
                for(int j = 1;j <= 3;j++) {
                    vis[b[x+i-1][y+j-1]] = 1;
                }
            }
            for(int i = 1;i <= 9;i++) {
                if(!vis[i]) {
```

```
                    flag = 0;
                }
            }
        }

        if(flag != 0) {
            printf("Moca finish this sudoku perfectly!\n");
        }
        else {
            printf("Moca is so careless!\n");
        }
    }

    return 0;
}
```

因为把所有括号都写上了所以看上去可能比较繁琐，但是这道题的思路还是清晰明确的。

F 川川爱爬山

难度	考点
4	二分查找

题目分析

思路一：上/下界二分查找

普通的二分查找只能帮助我们判断 key 是否在数组中存在，可以给出一个位置。但是并不能告诉我们数组中 key 从哪个位置出现到哪个位置， key 不存在时也无法告诉我们如果要插入 key 应该插入到什么位置。为了解决该问题，我们引入上界二分查找和下界二分查找的概念：

- 下界二分查找：对于给定的 key，在递增序列 a 中查找第一个值不小于 key 的位置。
- 上界二分查找：对于给定的 key，在递增序列 a 中查找第一个值大于 key 的位置。

比如，对于样例中的长度为 15 序列 $1, 1, 3, 5, 5, 5, 5, 5, 5, 5, 5, 7, 8, 8, 8$，从下标为 0 开始标号，对不同的 key，其上下界为：

- $key = 5$，下界为 3，上界为 11；
- $key = 6$，下界为 11，上界也为 11；
- $key = 8$，下界为 12，上界记为 15；
- $key = 9$，上、下界均记为 15。

可以发现， key 的上界与下界的差就是 key 在序列中出现的次数，如果次数不为 0，其下界就是第一次出现的位置，上界减一就是最后一次出现的位置。

因此我们仅需利用函数实现上/下界二分查找即可，具体方法见示例代码。

推荐大家可以将示例代码中的上/下界二分查找函数保存下来作为函数模板。

思路二：合并+二分查找

由于输入序列递增，我们可以在读入时将相同的数进行合并，最后得到一个不存在相同数的递增序列，在合并时记录第一次出现的位置和出现的次数，最后查找时利用朴素的二分查找即可。

具体而言我们可以定义二维数组 $a[1000005][3]$ ，数组中的元素 $a[j][0]$ 表示第 j 个不同的数的值，$a[j][1]$ 表示第 j 个不同的数第一次出现的位置， $a[j][2]$ 表示第 j 个不同的数出现的次数。在读入时维护一个变量 cnt 表示存到了 $a[cnt]$ ，每遇到新的数将 cnt 自增 1 。

具体实现参照示例代码。

示例代码

思路一：上/下界二分查找

```
#include <stdio.h>
int a[1000005];
int lower_bound(int l, int r, int key) //下界二分查找
{
    while(l < r) //条件为小于而非小于等于
    {
        int mid = l + (r - l) / 2;
        if(a[mid] >= key) r = mid; //此处为大于等于
        else l = mid + 1; //此处必须要+1
    }
    return l;
}
int upper_bound(int l, int r, int key) //上界二分查找
{
    while(l < r)
    {
        int mid = l + (r - l) / 2;
        if(a[mid] > key) r = mid; //与下界二分查找几乎完全相同，仅有此处需修改为大于
        else l = mid + 1;
    }
    return l;
}
int main()
{
    int n, t, key;
    scanf("%d%d", &n, &t);
    for(int i = 0; i < n; ++i)
        scanf("%d", &a[i]);
    while(t--)
    {
        scanf("%d", &key);
        //注意此处第二个参数要传入可能的最大上界，即lower_bound(0,n,key)而非
lower_bound(0,n-1,key)
        int l = lower_bound(0, n, key); //计算下界
        int r = upper_bound(0, n, key); //计算上界
        if(l != r) //上下界不同，说明key在序列中出现
            printf("%d %d\n", l + 1, r - l); //由于下标从0开始计数，需要输出l+1才是题目
中的位置
        else //上下界相同，说明key在序列中没有出现
            printf("-1\n");
```

```
    }
    return 0;
}
```

思路二：合并+二分查找

```
#include <stdio.h>
int a[1000005][3]; //0-值，1-第一次出现的位置，2-出现次数
int binary_search(int l, int r, int key) //在区间[l,r]上查找key，返回出现的位置，若没有出现返回-1
{
    int mid;
    while(l <= r)
    {
        mid = l + (r - l) / 2;
        if(a[mid][0] > key) r = mid - 1;
        else if(a[mid][0] < key) l = mid + 1;
        else break;
    }
    return r < l ? -1 : mid;
}
int main()
{
    int n, t, tmp, cnt = 0, ans;
    scanf("%d%d%d", &n, &t, &a[0][0]); //第一个数先读入
    a[0][1] = 1; //第一个数第一次出现位置为1
    a[0][2] = 1; //第一个数当前出现次数为1
    for(int i = 2; i <= n; ++i)
    {
        scanf("%d", &tmp);
        if(tmp != a[cnt][0]) //如果读入的数与当前正在合并的数不相同，则更新正在合并的数
        {
            a[++cnt][0] = tmp; //cnt++; a[cnt][0]=tmp;
            a[cnt][1] = i; //记录第一次出现的位置
        }
        ++a[cnt][2]; //次数+1
    }
    while(t--)
    {
        scanf("%d", &tmp);
        ans = binary_search(0, cnt, tmp); //二分查找
        if(~ans) printf("%d %d\n", a[ans][1], a[ans][2]); //如果ans不为-1，则找到key，输出答案
        else printf("-1\n"); //ans=-1，没有找到key，输出-1
    }
}
```

Author: 哪吒

G 破壁计划

难度	考点
4~5	字符串函数

题目分析

相信大家在面壁计划中已经看出玄机，我们在进行字符串加密时，无论是取奇串还是偶串，其包含的信息本质是一致的，因为在加密中，对于原字符串和反转字符串的合成字符串而言，第 1 位和最后 1 位是有严格对照关系的。它们包含的信息本质上是一致的，因此我们是可以使用加密后的字符串还原出原字符串的。

但由于字符串的来源不同，同一个字符串可能来自于一个字符串的奇串，也可能来自于一个字符串的偶串，因此解密是有风险的，你只有 0.5 的几率能解密出真正的字符串，在本题中，我们暂且不考虑这一因素（或许大家可以期待一下？）。

事实上，由于上述的奇串和偶串本身就具有对应关系，我们就可以通过它们之间的对应关系由一个串求出另一个串，然后你就可以还原出原本的字符串啦。同时，由于本题要求得到的是较小的原本字符串，因此仅考虑待解密的字符串是由原字符串的奇串组成就可以了。

于是替换，于是反转，于是叠加，于是我们就得到了一个也许是原字符串的解密字符串，是不是很简单?

示例代码

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define mlen 1005

// 函数：将字符串进行逆序和字符逆序
void reverseStringAndAlphabets(char* str) {
    int start = 0;
    int end = strlen(str) - 1;

    // 首字符逆序
    while (start < end) {
        char temp = str[start];
        str[start] = str[end];
        str[end] = temp;
        start++;
        end--;
    }
    // 字符逆序 a->z, z->a,不考虑大写字母
    for (int i = 0; i < strlen(str); i++) {
        if (islower(str[i])) {
            str[i] = 'a' + 'z' - str[i];
        }
    }

}

//进行字符串解密
```

```
void decrystring(char* str)
{
    int len = strlen(str);
    //查找到所有的et0，转换为eto,用strstr实现
    char* p;
    p = strstr(str, "et0");
    while (p != NULL)
    {
        *p = 'e';
        p++;
        *p = 't';
        p++;
        *p = 'o';
        p++;
        p = strstr(p, "et0");
    }
    char newstr[mlen + 1];
    char ans[mlen + 1];
    strcpy(newstr, str);
    reverseStringAndAlphabets(newstr);

    for (int j = 0; j < len; j++)
    {
        if (j % 2 == 0)
        {
            ans[j] = str[j / 2];
        }
        else
        {
            ans[j] = newstr[(j - 1) / 2];
        }
    }
    ans[len] = '\0';
    strcpy(str, ans);
}

int main() {
    char s[1005];
    gets(s);

    decrystring(s);
    printf("%s\n", s);

    return 0;
}
```

H 编撰幻想乡缘起

难度	考点
5	字符串处理

题意解析

首先我们回忆一下字符串的本质：`C` 语言中并没有单独的字符串类型，所谓的字符串本质是是一个以空字符 `\0` 为结尾的一维 `char` 类型数组。注意这个空字符，当你将一维字符数组当做字符串来处理时，这个字符串会在第一个空字符处『结束』。例如你可以尝试运行以下代码：

```
char buf[10] = {'1', '2', '3', '\0', '4', '5', '6'};
printf("%s", buf);
```

输出为：

```
123
```

因为 `%s` 将 buf 以字符串形式处理，因此只会输出第一个 `\0` 之前的内容，使用 `%s` 读取字符串时则会自动加上一个 `\0` 。

类似的， `gets` 读取字符串时会将读取到的内容最后的 `\n` 替换为 `\0` 来保证字符串的完整性（若没有 `\n` 而是读到了文件末尾则是添加一个 `\0` ）。

同理，字符串操作函数，包括但不限于 `strlen` ， `strcpy` ， `strcat` ， `strstr` 都是同样的道理，只会处理第一个 `\0` 之前的内容。而 `strcpy` ， `strcat` 这些函数也会在操作结果的字符串的对应位置上补上一个 `\0` 。

那么由以上内容我们就能写出各个操作对应的代码。

1. 对于操作 1 拼接，直接调用 `strcat` 函数即可。

   ```
   strcat(str1, str2);
   ```

2. 对于操作 2 截断，我们在截断位置增加一个终止符即可。

   ```
   str1[i] = '\0';
   ```

3. 对于操作 3 插入，我们将原字符串拆成两段，将其与给出的字符串依次拼接即可。

   ```
   strcpy(tmp, str1 + i + 1); // 将后半段复制给tmp
   str1[i + 1] = '\0';        // 截断字符串，使其仅留下前半段
   strcat(str1, str2);        // 拼接插入字符串
   strcat(str1, tmp);         // 拼接后半段
   ```

   或者将原字符串后半段拼接到给出的字符串上，再复制到原字符串对应的位置。

   ```
   strcat(str2, str1 + i + 1); //将后半段拼接到str2后面
   strcpy(str1 + i + 1, str2); //将拼好的字符串复制到后半段的位置
   ```

4. 对于操作 4 删除，其本质和操作 3 一致，如下：

   ```
   strcpy(tmp, str1 + j + 1); // 将后半段复制给tmp
   str1[i] = '\0';            // 截断字符串，使其仅留下前半段
   strcat(str1, tmp);         // 拼接后半段
   ```

或者

```
strcpy(tmp, str1 + j + 1); // 将后半段复制给tmp
strcat(str1 + i, tmp);     // 将tmp复制给str1对应的位置
```

需要注意的是，`strcpy` 和 `strcat` 的 `dst` 和 `src` 不能重叠，否则是一个未定义行为，结果取决于编译器的实现方式。因此直接进行如下操作是错误的。

```
// !undefined behavior
strcpy(str1 + i, str1 + j + 1);
```

5. 对于操作 5， 同样本质与操作 3 一致，如下

```
strcpy(tmp, str1 + j + 1); // 将后半段复制给tmp
str1[i] = '\0';            // 截断字符串，使其仅留下前半段
strcat(str1, str2);        // 拼接插入字符串
strcat(str1, tmp);         // 拼接后半段
```

或者

```
strcat(str2, str1 + j + 1); //将后半段拼接到str2后面
strcpy(str1 + i, str2);     //将拼接好的字符串复制到正确的位置
```

6. 对于操作 6，我们已经在 C7-D 中做过了，此处不再赘述。

```
char *p1 = strstr(str1, str2);
if (p1 == NULL) {
    puts("NULL");
} else {
    char *p2 = p1;
    while (strstr(p2 + 1, str2) != NULL) {
        p2 = strstr(p2 + 1, str2);
    }
    printf("%d %d\n", (int)(p1 - str1), (int)(p2 - str1));
}
```

由此我们可以得到如下示例代码

示例代码 - 1

```
#include <stdio.h>
#include <string.h>
int main(void) {
    int op, i, j;
    char str1[10005], str2[10005], tmp[10005];
    scanf("%s", str1);
    while (scanf("%d", &op) != EOF) {
        switch (op) {
            case 1:
                scanf("%s", str2);
                strcat(str1, str2);
                break;
            case 2:
```

```
                scanf("%d", &i);
                str1[i] = '\0';
                break;
            case 3:
                scanf("%d%s", &i, str2);
                strcpy(tmp, str1 + i + 1);
                str1[i + 1] = '\0';
                strcat(str1, str2);
                strcat(str1, tmp);
                break;
            case 4:
                scanf("%d%d", &i, &j);
                strcpy(tmp, str1 + j + 1);
                str1[i] = '\0';
                strcat(str1, tmp);
                break;
            case 5:
                scanf("%d%d%s", &i, &j, str2);
                strcpy(tmp, str1 + j + 1);
                str1[i] = '\0';
                strcat(str1, str2);
                strcat(str1, tmp);
                break;
            case 6:
                scanf("%s", str2);
                char *p1 = strstr(str1, str2);
                if (p1 == NULL) {
                    puts("NULL");
                } else {
                    char *p2 = p1;
                    while (strstr(p2 + 1, str2) != NULL) {
                        p2 = strstr(p2 + 1, str2);
                    }
                    printf("%d %d\n", (int)(p1 - str1), (int)(p2 - str1));
                }
        }
        puts(str1);
    }
    puts(str1);
    return 0;
}
```

示例代码 - 2

```
#include <stdio.h>
#include <string.h>
int main()
{
    int op, i, j;
    char str1[10001], str2[10001];
    gets(str1);
    while (~scanf("%d", &op))
        switch (op)
        {
            case 1:
```

```
                scanf("%s", str2);
                strcat(str1, str2);
                break;
            case 2:
                scanf("%d", &i);
                str1[i] = '\0';
                break;
            case 3:
                scanf("%d%s", &i, str2);
                strcat(str2, str1 + i + 1);
                strcpy(str1 + i + 1, str2);
                break;
            case 4:
                scanf("%d%d", &i, &j);
                strcpy(str2, str1 + j + 1);
                strcpy(str1 + i, str2);
                break;
            case 5:
                scanf("%d%d%s", &i, &j, str2);
                strcat(str2, str1 + j + 1);
                strcpy(str1 + i, str2);
                break;
            default:
                scanf("%s", str2);
                char *p = strstr(str1, str2);
                if (!p)
                    puts("NULL");
                else
                {
                    printf("%d ", p - str1);
                    for(char *q = p; q = strstr(p + 1, str2); p = q);
                    printf("%d\n", p - str1);
                }
        }
    puts(str1);
    return 0;
}
```

I 项链密码

难度	考点
5~6	字符串

题目分析

首先，对于长度为 n 的环形字符串，起点的选取有 n 种可能，假如不考虑时间复杂度，很容易想到的做法就是遍历这 n 种情况，利用`strcmp`函数来比较不同起点字符串之间的大小，选出最大的作为起点，但由于`strcmp`的时间复杂度是 $O(n)$，导致整体的时间复杂度高达 $O(n^2)$，此题 n 的上限为 10^6，$O(n^2)$ 的算法一定会超时。
以下给出复杂度为 $O(n)$ 的解法。

以 77789 为例。起点的选取有五种可能，为了便于表述，将原串 77789 从左到右编号为 A, B, C, D, E，例如 "以 B 作为起点"，则密码为 77897，并且定义

$$
\begin{aligned}
&B[0] = 7\\
&B[1] = 7\\
&B[2] = 8\\
&B[3] = 9\\
&B[4] = 7
\end{aligned}
$$

求解过程如下：

1. **以 A 作为起点**，得到的密码为 77789，由于当前为首次遍历，A 就是当前最大密码的起点
2. **以 B 作为起点**，得到的密码为 77897，开始和当前最大密码进行比较，

   第一次比较：$B[0] = A[0]$ （$B[0] = 7, A[0] = 7$）

   第二次比较：$B[1] = A[1]$ （$B[1] = 7, A[1] = 7$）

   第三次比较：$B[2] > A[2]$ （$B[2] = 8, A[2] = 7$）

   所以，以 B 作为起点的密码 **大于** 以 A 作为起点的密码

   此时最大密码的起点**变为** B
3. 从**最后一次比较结束位置（位置 D）**开始遍历

   以 D 作为起点，得到的密码为 89777，开始和当前最大密码进行比较，

   第一次比较：$D[0] > B[0]$ （$D[0] = 8, B[0] = 7$）

   所以，以 D 作为起点的密码 **大于** 以 B 作为起点的密码

   此时最大密码的起点**变为** D
4. 由于上次的**比较次数为1**，所以从**最后一次比较结束位置的下个位置（位置 E）**开始遍历

   以 E 作为起点，得到的密码为 97778，开始和当前最大密码进行比较，

   第一次比较：$E[0] > D[0]$ （$E[0] = 9, D[0] = 8$）

   所以，以 E 作为起点的密码 **大于** 以 D 作为起点的密码

   此时最大密码的起点**变为** E
5. 此时已经遍历了一整圈，所以跳出循环，**最终最大密码的起点即为** E

综上，遍历位置的更新方式有两种，分别为：

1. 上次的比较次数**为1**：最后一次比较结束位置的下一位
2. 上次的比较次数**大于1**：最后一次比较结束位置

此种算法不论是起点坐标，还是当前遍历的下标，都永远是不减的，所以为线性复杂度 $O(n)$，同时，对于环形字符串，可以先将原串复制一份放在结尾，方法见代码。

示例代码

```
#include <stdio.h>
#include <string.h>
#define max(x, y) ((x) > (y) ? (x) : (y))
char s[2000010];
```

```c
int main()
{
    scanf("%s", s);
    int i, len = strlen(s);
    for (i = 0; i < len; i++) // 将原串复制一份放到原串的结尾
        s[len + i] = s[i];
    s[len + i] = '\0';

    int j, max = 0;        // max用来保存当前最大密码的起点
    for (i = 1; i < len;) // i为当前正在遍历的密码的起点
    {
        for (j = 0; j < len && s[i + j] == s[max + j]; j++)
            ;
        if (j == len) // 出现形如原串为321321的循环结构 可以直接跳出
            break;
        if (s[i + j] > s[max + j]) // 挑战成功
            max = i;               // 更新最大密码的起点
        i = max(i + j, i + 1);
    }
    s[max + len] = '\0';
    printf("%s", &s[max]);
    return 0;
}
```

J 有理数2023 题解

难度	考点
6~7	字符串处理，模块化编程

题目解析

本题主要是希望锻炼大家模块化编程的能力。大家刚看到这个题目，可能就吓晕了。这么多描述，好难啊，不会做。但是，这道题真有那么难吗？接下来，我将带你走进《有理数2023》的世界，开启奇妙的编程之旅。

在启程前，请各位朋友们记住一句话：

> 我们先完成程序的骨架，然后再一点一点填充它的血肉。

然后，我给同学们介绍一个C语言中相当有用的小知识，本门课不学，但是真的很有用，学完就可以开始了，毕竟「工欲善其事，必先利其器」嘛。而且作为看到 `J` 题的优秀同学，我相信你是很乐意多学一点东西的。它就是「结构体」：

结构体（struct）是由一批数据组合而成的结构型数据。组成结构型数据的每个数据称为结构型数据的“成员”。其中成员可以是基本类型（ `int` , `double` 等）的数据、数组，也可以是已经定义过的结构体的数据，数组。其基本定义方式如下：

```c
typedef struct{
    成员列表
} 类型的名字;
```

比如说我们要定义一个数据类型“日期”，那么我们可以用三个`int`型数据“年、月、日”表示如下：

```
typedef struct{
    int y,m,d;
} date;
```

这样就定义了一种新的数据类型为`date`，一个`date`型数据包含了三个`int`型数据，分别为`y,m,d`。

在使用时，我们可以直接写（就像使用`int`、`double`等基本类型时一样）

```
date a
```

声明一个`date`型的数据，其变量名为`a`。如果我们想要调用日期`a`的年份，那么我们可以写作：

```
a.y
```

同理，调用`a`的月份和日期时，也可以写作`a.m a.d`。

那么，如果我们想要输出a的年月日，只需要写作

```
printf("%d %d %d\n",a.y,a.m,a.d);
```

就行了。

好了，我们开始吧！

首先，我们定义一个结构体，名叫`fraction`，意思是分数。它由两个`long long int`类型变量组成，分别是分子`num`和分母`den`：

```
typedef struct {
    long long int num;//分子
    long long int den;//分母
} fraction;
```

本题的主函数是很好写的，直接按行读入输入的命令，然后将其按元素位置拆分即可。主函数是整个程序的起点，也是咱们今天旅程的起点：

```
int main() {
    char s[100];
    fraction a[200];
    int cnt = 0;
    while (fgets(s, 99, stdin) != NULL) {
        int l = strlen(s);
        ++cnt;
        switch (s[0]) {
            case 'I':
                a[(int)s[6]] = str2fra(s, 8, l - 1);//字符串转分数
                break;
            case 'A':
                a[(int)s[4]] = ADD(a[(int)s[4]], a[(int)s[6]]);//加法
                break;
            case 'S':
```

```
                a[(int)s[4]] = SUB(a[(int)s[4]], a[(int)s[6]]);//减法
                break;
            case 'M':
                a[(int)s[4]] = MUL(a[(int)s[4]], a[(int)s[6]]);//乘法
                break;
            case 'D':
                if (a[(int)s[6]].num == 0) {
                    printf("LINE %d DIV BY ZERO!\n", cnt);
                } else {
                    a[(int)s[4]] = DIV(a[(int)s[4]], a[(int)s[6]]);//除法
                }
                break;
            case 'E':
                if (EQU(a[(int)s[4]], a[(int)s[6]])) {//判等
                    printf("%c == %c\n", s[4], s[6]);
                } else {
                    printf("%c != %c\n", s[4], s[6]);
                }
                break;
            case 'O':
                if (isdigit(s[7])) {
                    int pos = 7;
                    for (pos = 7; pos < l; ++pos) {
                        if (!isdigit(s[pos])) {
                            break;
                        }
                    }
                    int n = str2LLint(s, 7, pos - 1);
                    OUTPUTDOT(n, a[(int)s[pos + 1]]);//输出小数
                } else {
                    OUTPUT(a[(int)s[7]]);//输出分数
                }

                break;
            default://都不是？那肯定出错了
                printf("Error in input!%s\n",s);
                break;
        }
    }
    return 0;
}
```

这时候你可能会感到疑惑，因为里面用到的这些函数都没有定义呀。是的，不过没有关系，请记住我们的宗旨：

> 我们先完成程序的骨架，然后再一点一点填充它的血肉。

首先，先来做最简单的部分，也就是四则运算：

```
fraction ADD(fraction x, fraction y) {
    fraction ans;
    ans.den = x.den * y.den;
    ans.num = x.num * y.den + x.den * y.num;
    redu(&ans);
    return ans;
}
```

```
fraction SUB(fraction x, fraction y) {
    fraction ans;
    ans.den = x.den * y.den;
    ans.num = x.num * y.den - x.den * y.num;
    redu(&ans);
    return ans;
}

fraction MUL(fraction x, fraction y) {
    fraction ans;
    ans.den = x.den * y.den;
    ans.num = x.num * y.num;
    redu(&ans);
    return ans;
}

fraction DIV(fraction x, fraction y) {
    fraction ans;
    ans.den = x.den * y.num;
    ans.num = x.num * y.den;
    redu(&ans);
    return ans;
}

int EQU(fraction x, fraction y) {
    redu(&x);
    redu(&y);
    if (x.den == y.den && x.num == y.num) {
        return 1;
    } else {
        return 0;
    }
}
```

一口气写完了5个函数，你是不是很有成就感？毕竟本题的大部分工作其实已经做完了嘛（笑）

但是聪明的你一定发现了，这里面用到了一个还没定义的函数：`redu(&x)`。它是化简分数的意思。它的出现是很自然的：用到分数之前，以及返回结果之前，当然需要化简呀。于是，我们可以继续写化简函数：

```
void redu(fraction *x) {
    if ((*x).num == 0 || (*x).den == 0) {//如果有0，直接退出
        return;
    }
    int flag = 1;
    if ((((*x).num < 0) + ((*x).den < 0)) == 1) {//判断符号，负号前置
        flag = -1;
    }
    (*x).den=llabs((*x).den);//取绝对值
    (*x).num=llabs((*x).num);
    long long int g = gcd((*x).den, (*x).num);
    (*x).den /= g;
    (*x).num /= g;
    (*x).num *= flag;//负号放分子上
```

```
    return;
}
```

聪明的你一定又发现了，这里面又用到了一个还没定义的函数：`gcd(a,b)`，因为化简分数时当然需要用到最大公约数。还没定义没有关系，请记住我们的宗旨：先完成程序的骨架，然后再一点一点填充它的血肉。

```
long long int gcd(long long int x, long long int y) {
    return x % y ? gcd(y, x % y) : y;
}
```

到现在，四则运算和判断相等的部分已经告一段落了，你可以泡杯茶庆祝一下了（撒花）。

接下来，我们来继续完成前面的 `str2fra` 函数，即把字符串转换为分数的函数。

```
fraction str2fra(char s[], int l, int r) {
    fraction ans;
    ans.den = 1LL; //先把分母定义为1，免得出现分母为0的问题
    int flag = 0; //flag里面存放这个常量的类型：整数？有限小数？循环小数？分数？

    //【这里判断类型】

    fraction x;//整数部分
    fraction y;//小数部分
    int cntzero = 0, cnt9 = 0;
    switch (flag) {
        case 0://整数
            //【整数转换为分数】
            break;
        case 1://有限小数
            //【有限小数转换为分数】
            break;
        case 2://循环小数
            //【循环小数转换为分数】
            break;
        case 3://分数
            //【直接处理分数】
            break;
        default:
            //都不是？那肯定是判断写错了。
            printf("str2fra Error:flag=%d\n",flag);
            break;
    }
    redu(&ans);
    return ans;
}
```

写完了？很轻松吧？

「轻松个鬼啊！」，你说，「关键部分不都还没写吗？」

不要着急嘛，先完成大骨架，再完成小骨架，分而治之。

判断类型：

```
for (int i = l; i <= r; ++i) {
        if (s[i] == '/') {
            flag = 3;//有分数线，必为分数
            pos1 = i;//分数线的位置
            break;
        }
        if (s[i] == '.') {//有小数点，是有限还是无限呢？
            pos1 = i;//小数点的位置
            ++flag;
        }
        if (s[i] == '(') {//有循环节了，看来是无限
            ++flag;
            pos2 = i;//循环节开始的位置
        }
}
//flag=0：整数；1：有限小数；2：循环小数；3：分数
```

整数转换为分数，太好写了吧：

```
ans.num = str2LLint(s, l, r);
ans.den = 1LL;
```

直接处理分数？也很好写嘛：

```
ans.num = str2LLint(s, l, pos1 - 1);
ans.den = str2LLint(s, pos1 + 1, r);
```

有限小数？So easy！

```
x.den = 1LL;
x.num = str2LLint(s, l, pos1 - 1);
for (int i = pos1; i <= r; ++i) {
    if (isdigit(s[i])) {
        ++cntzero;
    }
}
y.num = str2LLint(s, pos1 + 1, r);//字符串转整数，不用讲了吧？
y.den = mypow(10LL, cntzero);//这个函数也不用再讲了吧？
redu(&y);
if (s[l] == '-') {
    ans = SUB(x, y);
} else {
    ans = ADD(x, y);
}
break;
```

接下来就是无限小数转分数环节了。

之前大家都做过分数转换为循环小数的题。那么循环小数怎么转换为分数呢？

对于纯循环小数而言，首先抛开整数部分不看，即只看$0.\dot{A}\dot{B}$的形式，它的分子就是循环节，然后循环节有几位，那么分母就写几个 9。即：

$$0.\dot{A}\dot{B}=\frac{\overline{AB}}{99}$$

对于混循环小数而言，其分子是从小数点后开始到第一个循环节后的所有数字，减去小数点后的不循环数字；分母则是循环节有几位就写几个 9，不循环的部分有几位就再添几个 0。例如：

$$0.A\dot{B}\dot{C}=\frac{\overline{ABC}-A}{990}$$

相信大家已经发现了，纯循环小数就是混循环小数的特例，这也是我之前没有区分的原因。

```
x.den = 1LL;
x.num = str2LLint(s, l, pos1 - 1);
y.num = str2LLint(s, pos1 + 1, r) - str2LLint(s, pos1 + 1, pos2 - 1);

for (int i = pos1; i < pos2; ++i) {
    if (isdigit(s[i])) {
        ++cntzero;
    }
}
for (int i = pos2; i <= r; ++i) {
    if (isdigit(s[i])) {
        ++cnt9;
    }
}
y.den = mypow(10LL, cnt9) - 1;
for (int i = 1; i <= cntzero; ++i) {
    y.den *= 10;
}

redu(&y);
if (s[l] == '-') {
    ans = SUB(x, y);
} else {
    ans = ADD(x, y);
}
```

好了，把这些填进去，`str2fra` 函数就大功告成了，再休息一下吧！

接下来就是输出环节了。输出分数很简单，不多说了：

```
void OUTPUT(fraction x) {
    redu(&x);
    if (x.num == 0) {
        printf("0\n");
        return;
    }
    if (x.den == 1) {
        printf("%lld\n", x.num);
        return;
    }
    printf("%lld/%lld\n", x.num, x.den);
    return;
}
```

本题的最后一个难点就是输出小数了。

为了计算 a/b 的小数点后的 x 位，我们就给这个分子 a 乘以一个10^x。当然不是一下子就乘上去，而是算一位乘一次。实际上也就是列除法竖式的过程。这里可能讲得不太清楚，举例说明一下：

例如我们需要计算

$$\frac{17}{49}$$

小数点后的前4位。

首先将17乘以10变成170，然后计算$170/49=3\cdots 23$，那么小数点后第一位就是3，然后此时a变为23

再将23乘以10变成230，计算$230/49=4\cdots 34$，那么小数点后第二位就是4，然后此时a变为34

再将34乘以10变成340，计算$340/49=6\cdots 46$，那么小数点后第三位就是6，然后此时a变为46

再将46乘以10变成460，计算$460/49=9\cdots 19$，那么小数点后第四位就是9。

于是，我们算出$17/49$小数点后前四位是0.3469

然后，四舍五入，判零，不多说了，看代码吧：

```
void OUTPUTDOT(int n, fraction x) {
    redu(&x);

    int neg = 0;
    if (x.num < 0) {
        neg = 1;
        x.num *= -1;
    }
    long long int d = x.num / x.den;//整数部分
    fraction temp;
    temp.num = d;
    temp.den = 1LL;
    x = SUB(x, temp);
    long long int a = x.num, b = x.den;
    int ans[1002];//因为要四舍五入，所以要算到1000的后一位哦
    for (int i = 1; i <= n + 1; ++i) {
        a *= 10;
        int t = 0;
        while (a >= b) {
            a -= b;
            ++t;
        }
        ans[i] = t;
    }
    if (ans[n + 1] < 5) {
        ;//舍
    } else {
        int cur = n, flag = 0;
        while (cur >= 1) {
            if (ans[cur] < 9) {
                ++ans[cur];
                flag = 1;
                break;
```

```
            } else {
                ans[cur] = 0;
                --cur;
            }
        }
        if (!flag) {
            ++d;
        }
    }

    int k = (d != 0);//判零
    for (int i = 1; i <= n; ++i) {
        k += (ans[i] != 0);
    }
    if (k != 0 && neg == 1) {
        printf("-");
    }
    printf("%lld.", d);
    for (int i = 1; i <= n; ++i) {
        printf("%d", ans[i]);
    }
    printf("\n");
}
```

好了，恭喜你，这道题已经做完了，回头看看吧，本来只有一个骨架，我们逐渐给它填上血肉，终究会完成这样艰难的旅途的。

参考代码

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>
#define MAXN 5+
const double eps = 1e-7;

typedef struct {
    long long int num;//分子
    long long int den;//分母
} fraction;

long long int gcd(long long int x, long long int y);
void redu(fraction *x);
long long int str2LLint(char s[], int l, int r);
long long int mypow(long long int x, int a);
fraction str2fra(char s[], int l, int r);
fraction ADD(fraction x, fraction y);
fraction SUB(fraction x, fraction y);
fraction MUL(fraction x, fraction y);
fraction DIV(fraction x, fraction y);
int EQU(fraction x, fraction y);
void OUTPUT(fraction x);
void OUTPUTDOT(int n, fraction x);
```

```
int main() {
    char s[100];
    fraction a[200];
    int cnt = 0;
    while (fgets(s, 99, stdin) != NULL) {
        int l = strlen(s);
        ++cnt;
        switch (s[0]) {
            case 'I'://INPUT A 0.3
                a[(int)s[6]] = str2fra(s, 8, l - 1);
                break;
            case 'A'://ADD A B
                a[(int)s[4]] = ADD(a[(int)s[4]], a[(int)s[6]]);
                break;
            case 'S':
                a[(int)s[4]] = SUB(a[(int)s[4]], a[(int)s[6]]);
                break;
            case 'M':
                a[(int)s[4]] = MUL(a[(int)s[4]], a[(int)s[6]]);
                break;
            case 'D':
                if (a[(int)s[6]].num == 0) {
                    printf("LINE %d DIV BY ZERO!\n", cnt);
                } else {
                    a[(int)s[4]] = DIV(a[(int)s[4]], a[(int)s[6]]);
                }
                break;
            case 'E':
                if (EQU(a[(int)s[4]], a[(int)s[6]])) {
                    printf("%c == %c\n", s[4], s[6]);
                } else {
                    printf("%c != %c\n", s[4], s[6]);
                }
                break;
            case 'O':
                if (isdigit(s[7])) {//OUTPUT 20 A
                    int pos = 7;
                    for (pos = 7; pos < l; ++pos) {
                        if (!isdigit(s[pos])) {
                            break;
                        }
                    }
                    int n = str2LLint(s, 7, pos - 1);
                    OUTPUTDOT(n, a[(int)s[pos + 1]]);
                } else {
                    OUTPUT(a[(int)s[7]]);
                }

                break;
            default:
                printf("Error in input!%s\n",s);
                break;
        }
    }
    return 0;
}
```

```
long long int gcd(long long int x, long long int y) {
    return x % y ? gcd(y, x % y) : y;
}

void redu(fraction *x) {
    if ((*x).num == 0 || (*x).den == 0) {
        return;
    }
    int flag = 1;
    if ((((*x).num < 0) + ((*x).den < 0)) == 1) {
        flag = -1;
    }
    (*x).den = llabs((*x).den);
    (*x).num = llabs((*x).num);
    long long int g = gcd((*x).den, (*x).num);
    (*x).den /= g;
    (*x).num /= g;
    (*x).num *= flag;
    return;
}

long long int mypow(long long int x, int a) {
    long long int ans = 1LL;
    for (int i = 1; i <= a; ++i) {
        ans *= x;
    }
    return ans;
}

long long int str2LLint(char s[], int l, int r) {
    //功能：将s[l,r]字符串转换为long long整数，除了开头的负号以外，忽略所有符号
    long long int ans = 0LL;
    for (int i = l; i <= r; ++i) {
        if (isdigit(s[i])) {
            ans = ans * 10LL + (s[i] - '0');
        }
    }
    return s[l] == '-' ? -ans : ans;
}

fraction str2fra(char s[], int l, int r) {
    fraction ans;
    ans.den = 1LL;
    int flag = 0;
    int pos1 = -1, pos2 = -1;
    for (int i = l; i <= r; ++i) {
        if (s[i] == '/') {
            flag = 3;
            pos1 = i;
            break;
        }
        if (s[i] == '.') {
            pos1 = i;
            ++flag;
        }
```

```
        if (s[i] == '(') {
            ++flag;
            pos2 = i;
        }
    }
    fraction x;//整数部分
    fraction y;//小数部分
    int cntzero = 0, cnt9 = 0;
    switch (flag) {
        case 0://整数
            ans.num = str2LLint(s, l, r);
            ans.den = 1LL;
            break;
        case 1://有限小数 0.325 pos1=1 l=0 r=4
            x.den = 1LL;
            x.num = str2LLint(s, l, pos1 - 1);
            for (int i = pos1; i <= r; ++i) {
                if (isdigit(s[i])) {
                    ++cntzero;
                }
            }
            y.num = str2LLint(s, pos1 + 1, r);
            y.den = mypow(10LL, cntzero);
            redu(&y);
            if (s[l] == '-') {
                ans = SUB(x, y);
            } else {
                ans = ADD(x, y);
            }
            break;
        case 2://循环小数
            x.den = 1LL;
            x.num = str2LLint(s, l, pos1 - 1);
            //混循环 0.14(43) pos1=1 l=0 r=7 pos2=4
            y.num = str2LLint(s, pos1 + 1, r) - str2LLint(s, pos1 + 1, pos2 - 1);

            for (int i = pos1; i < pos2; ++i) {
                if (isdigit(s[i])) {
                    ++cntzero;
                }
            }
            for (int i = pos2; i <= r; ++i) {
                if (isdigit(s[i])) {
                    ++cnt9;
                }
            }
            y.den = mypow(10LL, cnt9) - 1;
            for (int i = 1; i <= cntzero; ++i) {
                y.den *= 10;
            }

            redu(&y);
            if (s[l] == '-') {
                ans = SUB(x, y);
            } else {
                ans = ADD(x, y);
```

```
            }
            break;
        case 3://分数
            ans.num = str2LLint(s, l, pos1 - 1);
            ans.den = str2LLint(s, pos1 + 1, r);
            break;
        default:
            printf("str2fra Error:flag=%d\n", flag);
            //TODO
            break;
    }
    redu(&ans);
    return ans;
}

void OUTPUT(fraction x) {
    redu(&x);
    if (x.num == 0) {
        printf("0\n");
        return;
    }
    if (x.den == 1) {
        printf("%lld\n", x.num);
        return;
    }
    printf("%lld/%lld\n", x.num, x.den);
    return;
}

void OUTPUTDOT(int n, fraction x) {
    redu(&x);

    int neg = 0;
    if (x.num < 0) {
        neg = 1;
        x.num *= -1;
    }
    long long int d = x.num / x.den;
    fraction temp;
    temp.num = d;
    temp.den = 1LL;
    x = SUB(x, temp);
    long long int a = x.num, b = x.den;
    int ans[1001];
    for (int i = 1; i <= n + 1; ++i) {
        a *= 10;
        int t = 0;
        while (a >= b) {
            a -= b;
            ++t;
        }
        ans[i] = t;
        //printf("%d", t);
    }
    if (ans[n + 1] < 5) {
        ;//
```

```
    } else {
        int cur = n, flag = 0;
        while (cur >= 1) {
            if (ans[cur] < 9) {
                ++ans[cur];
                flag = 1;
                break;
            } else {
                ans[cur] = 0;
                --cur;
            }
        }
        if (!flag) {
            ++d;
        }
    }

    int k = (d != 0);
    for (int i = 1; i <= n; ++i) {
        k += (ans[i] != 0);
    }
    if (k != 0 && neg == 1) {
        printf("-");
    }
    printf("%lld.", d);
    for (int i = 1; i <= n; ++i) {
        printf("%d", ans[i]);
    }
    printf("\n");
}

fraction ADD(fraction x, fraction y) {
    fraction ans;
    ans.den = x.den * y.den;
    ans.num = x.num * y.den + x.den * y.num;
    redu(&ans);
    return ans;
}

fraction SUB(fraction x, fraction y) {
    fraction ans;
    ans.den = x.den * y.den;
    ans.num = x.num * y.den - x.den * y.num;
    redu(&ans);
    return ans;
}

fraction MUL(fraction x, fraction y) {
    fraction ans;
    ans.den = x.den * y.den;
    ans.num = x.num * y.num;
    redu(&ans);
    return ans;
}

fraction DIV(fraction x, fraction y) {
```

```
    fraction ans;
    ans.den = x.den * y.num;
    ans.num = x.num * y.den;
    redu(&ans);
    return ans;
}

int EQU(fraction x, fraction y) {
    redu(&x);
    redu(&y);
    if (x.den == y.den && x.num == y.num) {
        return 1;
    } else {
        return 0;
    }
}
```

- End -