A - 立个 Flag

难度	考点
1	输入输出，转义字符

题目分析

将样例中所给的字符画输出即可。在输出时注意处理转义字符和换行符的问题。

需要注意转义或者特殊处理的字符：单引号，双引号，百分号，反斜杠。

示例程序

```c
#include <stdio.h>

int main()
{
    printf("          \'\' @@                    \n");
    printf("          \'\'      ##               \n");
    printf("          \'\'           ++          \n");
    printf("          \'\'                %%%%     \n");
    printf("          \'\'                    \\\\   \n");
    printf("          \'\'                      >\n");
    printf("          \'\'                  //   \n");
    printf("          \'\'                %%%%     \n");
    printf("          \'\'           ++          \n");
    printf("          \'\'      ##               \n");
    printf("          \'\' @@                    \n");
    printf("          \"\"                       \n");
    printf("          \"\"                       \n");
    printf("          \"\"                       \n");
    printf("          \"\"                       \n");
    printf("          \"\"                       \n");
    printf("          \"\"                       \n");
    printf("==========================================\n");

    return 0;
}
```

B - 圆滚滚大满贯！

题目分析

根据题目所给的相关公式，计算并对应输出即可。

推荐使用宏定义 `#define PI 3.1415926` 来定义 π 并使用，方便调试和后续更改。

请使用 `double` 型变量存储以保证范围和精度，使用 `float` 型会在后续的计算中由于范围不足导致溢出或者精度不够导致误差较大。

示例程序

```c
#include <stdio.h>
#define pi 3.1415926

int main()
{
    double r, a, b;

    scanf("%lf%lf%lf", &r, &a, &b);
    printf("%.2f %.2f %.2f %.2f", 2 * pi * r, pi * r * r, pi * r * r * r * 4 / 3, pi * a * b);

    return 0;
}
```

C - 难度预测

难度	考点
2	逻辑运算

题目分析

本题为课件 C2-10 的改编。利用了逻辑表达式的真值等于 1 这个特点，我们可以将 6 个对 $a_i = b_i$ 的判断结果直接相加，最终得到正确预测题目的个数。

示例程序

```c
#include <stdio.h>

int main()
{
    int a1, a2, a3, a4, a5, a6;
    int b1, b2, b3, b4, b5, b6;
    int ans;

    scanf("%d%d%d%d%d%d", &a1, &a2, &a3, &a4, &a5, &a6);
    scanf("%d%d%d%d%d%d", &b1, &b2, &b3, &b4, &b5, &b6);
    ans = (a1 == b1) + (a2 == b2) + (a3 == b3) + (a4 == b4) + (a5 == b5) + (a6 == b6);
    printf("%d\n", ans);

    return 0;
}
```

D - 打分

难度	考点
3	数组，合理选择数据类型

本题可以有两种解法，分别是使用数组的解法和不使用数组的解法。

在使用数组的解法中，直接用数组记录所有输入的值，然后先遍历数组寻找最值，再将非最值求平均数即可。

在不使用数组的解法中，用 `maxNum` 和 `minNum` 时时记录下最大最小值，用 `cntMax` 和 `cntMin` 两个变量记录最大值和最小值的数量，用 `sum` 记录下所有数字的和，最后做差求平均数即可。注意，判断 `"#DIV/0"` 时要考虑所有数字相同的情况，此时 `cntMax == cntMin == n`。

HINT: 在寻找最大最小值时，我们一般会给存储最值的变量赋一个无穷大的值。由于计算机无法存储无穷大，所以我们一般用一个非常大的值或者非常小的值替代，一般是 `0x7fffffff` 或者 `0x3f3f3f3f`。前者是 32 位整数 (一般是 `int`) 最大值的十六进制表示，后者是前者的一半。一般后者比较常用，其优点是两个这样的"无穷大"相加时不会溢出成负数，在一些情况下非常实用。

注意计算时使用 `long long`，在这里同样可以有个小技巧，采用 `#define LL long long` 可为 `long long` 类型起一个较为简短的别名 `LL`，在后续的编程过程中可采用 `LL` 来代替 `long long`。

解法一

```c
#include <stdio.h>
#define INF 0x3f3f3f3f
#define N 100000

int a[N + 5];

int main()
{
    int n, i, max = -INF, min = INF, cnt = 0;
    long long sum = 0;

    scanf("%d", &n);

    // 第一次循环，找到数组中最大最小值
    for (i = 1; i <= n; i++)
    {
        scanf("%d", &a[i]);
        if (a[i] > max) max = a[i];
        if (a[i] < min) min = a[i];
    }

    // 第二次循环，统计数组中合法数字之和与个数
    for (i = 1; i <= n; i++)
    {
        if (a[i] != max && a[i] != min)
        {
            cnt++;
            sum += a[i];
        }
```

```
    }

    if (cnt == 0)
        printf("#DIV/0!");
    else
        printf("%.2f", (double)sum / cnt);

    return 0;
}
```

解法二

```
#include <stdio.h>
#define INF 0x3f3f3f3f

int main()
{
    int n, i;
    long long cntMax = 0, cntMin = 0,
              maxNum = -INF, minNum = INF,
              x, sum = 0;

    scanf("%d", &n);

    // 在循环中直接维护最大最小值及其个数
    for (i = 1; i <= n; ++i)
    {
        scanf("%lld", &x);
        cntMax += (x == maxNum);
        cntMin += (x == minNum);
        if (x > maxNum) maxNum = x, cntMax = 1;
        if (x < minNum) minNum = x, cntMin = 1;
        sum += x;
    }

    if (cntMax + cntMin == n || cntMax == n) // 只有两种值或一种值的情况是无解
    {
        printf("#DIV/0!");
    }
    else
    {
        printf("%.2f", 1.0 * (sum - maxNum * cntMax - minNum * cntMin) / (n - cntMax - cntMin));
    }
    return 0;
}
```

E - 小翔哥学古典密码

难度	考点
3	字符读入，ASCII

题目分析

这道题主要考察对于如何利用循环从文本中读取不定长度的字符文本，逻辑表达式的短路求值，加深对输入流和字符计算的理解。

在课上的 PPT 中，介绍了利用 `while` 循环和 `getchar` 语句输入不定长度字符文本的方法。然而在本题中需要注意的是，用 `scanf` 读入密钥 K 后，输入流的读指针（可以理解为光标）停留在 K 之后，第一行行尾换行符之前，这时如果直接循环输入，会导致最终结果多输出了一个换行（可能是大部分同学PE的原因）。解决办法是可以在 `scanf` 后跟一个空的 `getchar` 语句，吃掉多余的那个换行符，将读指针移动到下一行文本真正开始的地方。

在循环内部我们可以利用逻辑表达式的短路求值来判断当前字符是否是小写字母，如果是，依据题面的恺撒密码的解密算法进行变换；否则，直接输出。具体算法请参考下方代码。

示例程序

```c
#include <stdio.h>

int main()
{
    int k;
    char c, p;

    scanf("%d", &k);
    getchar(); // 这里getchar() 的作用是吃掉第一行的换行符，使得下次 getchar() 读入的内容从下一行开始
    while ((c = getchar()) != EOF)
    {
        if ('a' <= c && c <= 'z')
        {
            c = c - 'a';           // 用0-25的数字指代
            p = (c - k + 26) % 26; // P = (C-K) MOD 26
            p = p + 'a';           // 转为正常的ASCII码
            printf("%c", p);
        }
        else
            printf("%c", c);
    }

    return 0;
}
```

F - 二次函数

难度	考点
3	浮点数计算，输出控制

题目分析

根据题目所给的相关知识，对应输出即可

1. 抛物线开口方向由 a 决定，$a > 0$ 时，开口向上
2. 抛物线对称轴是 $x = -\frac{b}{2a}$
3. 抛物线顶点坐标是 $(-\frac{b}{2a}, \frac{4ac - b^2}{4a})$
4. 二次函数求根公式为 $x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$

示例程序

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

int main()
{
    double a, b, c;
    double x, y, r1, r2;

    scanf("%lf%lf%lf", &a, &b, &c);

    // 抛物线开口方向由 a 决定，a > 0 时，开口向上
    if (a > 0)
    {
        printf("UP\n");
    }
    else
    {
        printf("DOWN\n");
    }

    // 抛物线对称轴是 x = -b / (2 * a)
    if (a * b > 0)
    {
        printf("LEFT\n");
    }
    else if (a * b == 0)
    {
        printf("MID\n");
    }
    else
    {
        printf("RIGHT\n");
    }

```

```c
    // 顶点坐标
    x = -b / (2 * a);
    y = (4 * a * c - b * b) / (4 * a);
    printf("%.2lf %.2lf\n", x, y);

    // 求根
    if (b * b < 4 * a * c)
    {
        printf("NULL\n");
    }
    else if (b * b == 4 * a * c)
    {
        printf("%.2lf\n", x);
    }
    else
    {
        r1 = (-b + sqrt(b * b - 4 * a * c)) / (2 * a);
        r2 = (-b - sqrt(b * b - 4 * a * c)) / (2 * a);
        printf("%.2lf %.2lf\n", r1, r2);
    }

    return 0;
}
```

G - 金仙花数（简单版）

难度	考点
3	循环，按位提取

题目分析

按照题意，遍历区间 $[l, r]$ 内每个数，判断其是否符合条件即可。可以观察到在题目范围内金仙花数只能是三位数（一、二位数不满足要求，而唯一的四位数 1000 不是金仙花数），因此实际可以一律将要处理的数当三位数处理。

将三位数中的每一位数提取出来的方式，请参考 PPT 例 2-9 中的方法。根据该方法，可以写出如下的示例程序 1。

此外，也可以用一种按位处理数字的方法进行处理，尤其是在数位很大或者不确定的时候，该方法比上述方法处理起来更轻松。具体的实现见示例程序 2。

```
int a = 0, b = 1; // a 为和，b 为乘积
while (num > 0)
{
    a = a + (num % 10); // 取末尾最后一位数，比如 23456 就取 6
    b = b * (num % 10);
    num = num / 10; // 整形除法，不会产生进位，会直接抹除最后一位，如 23456 变成 2345
}
```

示例程序 1

```
#include <stdio.h>

int main()
{
    int n, l, r, cnt;
    int i;
    int a, b, c, sum, prod;

    scanf("%d", &n);
    while (n--)
    {
        scanf("%d%d", &l, &r);

        cnt = 0;
        for (i = l; i <= r; i++)
        {
            // a, b, c 分别为百位、十位、个位
            a = i / 100 % 10;
            b = i / 10 % 10;
            c = i % 10;
            // 按要求计算和 sum 和乘积 prod
            sum = a + b + c;
            prod = a * b * c;
            if (sum >= 20 && prod >= 162)
                cnt++;
```

```
        }
        if (cnt > 0)
            printf("%d\n", cnt);
        else
            printf("404 Not Found\n");
    }

    return 0;
}
```

示例程序 2

```
#include <stdio.h>

int main()
{
    int t, l, r;
    int i;
    int ans;
    int tmp, a, b;

    scanf("%d", &t);
    while (t > 0)
    {
        ans = 0;
        scanf("%d%d", &l, &r);
        for (i = l; i <= r; i++)
        {
            // 按要求计算数位和 a 和数位乘积 b
            tmp = i, a = 0, b = 1;
            while (tmp > 0)
            {
                a = a + (tmp % 10);
                b = b * (tmp % 10);
                tmp = tmp / 10;
            }
            if (a >= 20 && b >= 162) ans = ans + 1;
        }
        if (ans == 0)
            printf("404 Not Found\n");
        else
            printf("%d\n", ans);
        t = t - 1;
    }

    return 0;
}
```

H - 我们无法一起分组

难度	考点
4	数组，基本数据类型，循环

题目分析

根据题干解释，我们可维护一个数组 a，用来存储班级同学学号名单。

首先，我们需要完成题目的读入。正常读入小组人数 n 后，我们需要读入每个同学的学号 $a[i]$，由于名单人数不定，我们采用 `EOF` 来判定读入是否结束。在读入学号的同时，我们可以计算出全班同学的学号之和 Sum 和全班同学人数 $Count_{all}$，这样在读入完成后，我们就可以通过 $Sum/Count_{all}$ 计算出全班同学学号平均值 AVE_{all}。

在接下来的分组过程中，根据题目要求每个组的同学在首尾翻转后的名单上是相邻的，我们需要倒序遍历数组 a。为了保证接下来分得的每个小组人数都为 n，我们可以设置倒序循环遍历的步长为 n。那么在倒序的每次循环中，我们需要判断剩余人数 $Count_{remain}$ 和小组人数 n 的关系，有如下两种情况：

- $Count_{remain} \geq n$，说明还可以继续分组，对于当下分出来的这个组，我们需要判断它的学号平均值 AVE_i 和 AVE_{all} 之间的关系，同样有两种情况：
  - $AVE_i \geq AVE_{all}$，我们输出当前组的序号和组内每个人的学号。
  - $AVE_i \leq AVE_{all}$，无需进行其他操作。
- $Count_{remain} < n$，说明无法继续分组，输出 `out of range` 和剩余学生的学号。

注意：

- 题目中说到学号的范围是 `int`，但在计算全班学号总和的过程中可能会超出 `int` 的范围，所以需要使用 `long long` 类型存储学号和。

示例程序

```c
#include <stdio.h>

int main()
{
    int n;
    int a[300];
    int m = 0;
    long long sumAll = 0, sumGroup = 0;
    int i, j;
    int num = 1;

    scanf("%d", &n);
    while (scanf("%d", &a[m]) != EOF)
    {
        sumAll += a[m];
        m++;
    }

    // 将 i 从 m - 1 到 0 处理，实现倒序的要求
    for (i = m - 1; i >= 0; i -= n)
    {
```

```
        if (i + 1 < n) // 剩余人数小于 n 人
        {
            printf("out of range\n");
            for (j = i; j >= 0; j--) printf("%d\n", a[j]);
        }
        else           // 可以正常分组
        {
            sumGroup = 0;
            for (j = i; j > i - n; j--) sumGroup += a[j];
            if (sumGroup * m >= sumAll * n) // 通过将 sumGroup / n >= sumAll / m 移项为乘法比较，避免浮点数之间运算产生误差
            {
                printf("Group:%d\n", num);
                for (j = i; j > i - n; j--) printf("%d\n", a[j]);
            }
            num++;
        }
    }

    return 0;
}
```

I - 平行时空的日期计算

难度	考点
5	时间日期处理

题目分析

本题主要考察了日期计算、数组、多组数据输入输出，此外，也考察了同学们用标记变量标记某种情况（如样例程序中的 `flag` 标记日期是否合法）。为了简化同学们的计算程度，助教设定了一个平行时空，在这个时空下计算日期只需要计算距离 0000 年 1 月 1 日有多少天，再用取模运算就可以得到日期。此题题目不难理解，但是有几个小坑，并且由于是多组数据输入，因此只要有一个点没有写对就只能得 0 分。

几个需要注意的点

1、日期的存储方式：用数组，当 i 表示月份的时候，用 `month[i]` 就表示某月的天数。（当然，这里你可以不用数组保存日期，但相应的需要在判断该月有几天时用很多 `if-else` 以及 `||` 来判断）

2、非法日期的判断：只需要判断 `day` 是否大于该月的天数。

3、星期日的输出：星期一到星期六的输出直接输出 `days % 7` 即可,而星期日需要特判一下，详见标准代码。（也有其他的方法，这里不赘述）

4、非法日期的输出：因为发现同学们上一次上节课时，输出一个语句经常不用复制粘贴而自己把那句话打出来，导致漏空格等错误出现的 WA，本次助教特意将 `lT lS A WRONG DATE` 中的字母 `I` 换成了字母 `l`。

5、数组的定义方式：（个人）建议全部定义为全局数组，不要用变量定义数组的大小！

示例程序

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

int main()
{
    // 用数组保存每个月的天数能够简化代码编写 ，如果不用数组的话就需要在程序中加一些条件语句判断该月的天数
    int year[2] = {365, 366};                                                    // year[0] 为非闰年对应的一年的天数，year[1] 为闰年对应的一年的天数
    int month1[13] = {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}; // month1[i] 用来保存闰年 i 月的天数
    int month2[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}; // month2[i] 同来保存非闰年 i 月的天数
    int y, m, d, flag, i, isleap; // 用 flag 判断日期是否合法。每次循环开始另flag = 1；如果日期不合法则另 flag = 0。

    while (scanf("%d%d%d", &y, &m, &d) != EOF)
    {
        flag = 1;
        // 以下条件语句判断日期是否合法
        if (y % 4 == 0 && y % 100 != 0 || y % 400 == 0)
```

```
        {
            isleap = 1;
            if (d > month1[m])
            {
                flag = 0;
            }
        }
        else
        {
            isleap = 0;
            if (d > month2[m])
            {
                flag = 0;
            }
        }
        if (flag == 0) printf("lT lS A WRONG DATE\n");
        // 以下语句计算合法日期距离 0 年 1 月 1 日的天数
        else
        {
            for (i = 0; i < y; i++)
            {
                if ((i % 4 == 0 && i % 100 != 0) || (i % 400 == 0))
                    d += year[1];
                else
                    d += year[0];
            }
            for (i = 0; i < m; i++)
            {
                if (isleap)
                    d += month1[i];
                else
                    d += month2[i];
            }
            printf("%d\n", d % 7 == 0 ? 7 : d % 7);
        }
    }

    return 0;
}
```

J - 基础物理实验（简单版）

难度	考点
5	数学思维，一维数组的运用

题目分析

我们可以将序列 $[p_1,p_2,\cdots,p_n]$ 按照下标的奇偶分为两个序列 $[p_1,p_3,\cdots,p_{2\lfloor\frac{n}{2}\rfloor+1}]$ 和 $[p_2,p_4,\cdots,p_{2\lfloor\frac{n}{2}\rfloor}]$，那么对于题目中的一次操作，等价于将上述任意一个序列中任意两个相邻的数字交换。

注意到将 $[p_1,p_2,\cdots,p_n]$ 变为 $[1,2,\cdots,n]$ 等价于分别将 $[p_1,p_3,\cdots,p_{2\lfloor\frac{n}{2}\rfloor+1}]$ 和 $[p_2,p_4,\cdots,p_{2\lfloor\frac{n}{2}\rfloor}]$ 变为 $[1,3,\cdots,2\lfloor\frac{n}{2}\rfloor+1]$ 和 $[2,4,\cdots,2\lfloor\frac{n}{2}\rfloor]$，因此题目有解当且仅当所有的奇数都出现在 $[p_1,p_3,\cdots,p_{2\lfloor\frac{n}{2}\rfloor+1}]$且所有的偶数都出现在 $[p_2,p_4,\cdots,p_{2\lfloor\frac{n}{2}\rfloor}]$。

我们定义一个序列的逆序对为

$$\{(i,j)\mid i<j \text{ and } p_i>p_j\}$$

可以发现，对于相邻的两个数 p_i 和 p_{i+1}，如果$p_i>p_{i+1}$，那么它们两个在交换后该序列的逆序对会减少 1；如果$p_i<p_{i+1}$，那么它们两个在交换后该序列的逆序对会增加 1，而一个序列在单调递增前，总能找到相邻的两个数 p_i 和 p_{i+1} 满足 $p_i>p_{i+1}$，因此每次操作我们都可以让上述两个序列之一的逆序对数量减少 1。而 $[1,3,\cdots,2\lfloor\frac{n}{2}\rfloor+1]$ 和 $[2,4,\cdots,2\lfloor\frac{n}{2}\rfloor]$ 的逆序对数量均为 0，因此最终答案即为 $[p_1,p_3,\cdots,p_{2\lfloor\frac{n}{2}\rfloor+1}]$ 的逆序对数量加上 $[p_2,p_4,\cdots,p_{2\lfloor\frac{n}{2}\rfloor}]$ 的逆序对数量。

我们可以使用一维数组和二重 `for` 循环来实现求解一个序列的逆序对数量，时间复杂度为 $O(n^2)$。

示例程序

```c
#include <stdio.h>

int main()
{
    int a[1086];
    int t, n, i, j, k = 1, tag, ans;

    scanf("%d", &t);
    while (k <= t)
    {
        k++;
        scanf("%d", &n);
        for (i = 1; i <= n; i++)
            scanf("%d", &a[i]);

        // tag 表示当前序列是否非法
        tag = 0;
        for (i = 1; i <= n; i++)
            if (i % 2 != a[i] % 2) // i 的奇偶性与 a[i] 不同，非法
                tag = 1;

        if (tag) // 无解
```

```
        {
            printf("-1\n");
        }
        else // 有解
        {
            ans = 0;
            // 统计奇数位置上的逆序对个数
            for (i = 1; i <= n; i += 2)
            {
                for (j = 1; j < i; j += 2)
                {
                    ans += a[j] > a[i];
                }
            }
            // 统计偶数位置上的逆序对个数
            for (i = 2; i <= n; i += 2)
            {
                for (j = 2; j < i; j += 2)
                {
                    ans += a[j] > a[i];
                }
            }
            printf("%d\n", ans);
        }
    }
    return 0;
}
```