A - 素因数（签到）

考点	难度
循环，判断	1

题目说明

本题考查了循环语句、判断语句，难度很小。

样例程序

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

int main()
{
    int n, i, cnt, flag, j;

    while ((scanf("%d", &n)) != EOF)
    {
        cnt = 0;
        for (i = 2; i <= n; i++)
        {
            if (n % i == 0)
            {
                flag = 1;
                for (j = 2; j <= sqrt(i); j++)
                {
                    if (i % j == 0)
                    {
                        flag = 0;
                        break;
                    }
                }
                if (flag == 1)
                {
                    while (n % i == 0)
                    {
                        n /= i;
                    }
                    cnt++;
                }
            }
        }
        printf("%d\n", cnt);
    }

    return 0;
}
```

B - 恰面包

考点	难度
递推，数学运算	1

题目说明

根据"前一天面包数量是后一天数量加一的二倍"倒推。注意需要估算一下答案的大小，储存答案的变量要选取合适的数据类型。

样例程序

```c
#include <stdio.h>

int main()
{
    int n;
    long long ans;

    while (scanf("%d", &n) != EOF)
    {
        ans = 1;
        for (int i = 1; i < n; i++)
        {
            ans += 1;
            ans *= 2;
        }
        printf("%lld\n", ans);
    }

    return 0;
}
```

C - 函数的作用?

考点	难度
使用函数来优化代码结构	2

题目说明

本题要求同学们对五组数据进行不同的运算，得到对应结果。这当然可以在主函数里写循环来解决，但是在日后的编程学习中，像本题这样明确需求的操作，如果全部按顺序放在主函数中，势必会造成代码冗长，且逻辑关系模糊。为此，我们可以将这种操作逻辑抽象为一个函数，来简化整体代码，并且使代码的逻辑结构更加清晰。

样例程序

```c
#include <stdio.h>

int calculate(int x, int op)
{
    return (op == 0) ? x * 2 + 1 : (op == 1) ? x / 2 - 1 : (op == 2) ? x << 2 : (op == 3) ? x >> 2 : (op == 4) ? x & 2 : 0;
}

int main()
{
    int n, t;
    int i, j;

    scanf("%d", &n);
    for (i = 0; i <= 4; i++)
    {
        for (j = 1; j <= n; j++)
        {
            scanf("%d", &t);
            printf("%d ", calculate(t, i));
        }
        printf("\n");
    }

    return 0;
}
```

D - 蛇形折线

考点	难度
数学推导	3

题目分析

容易证明，最优解 (a,b) 一定在某个下行斜坡上，如果在上行斜坡必定能通过缩小 x 而使得该点位于下一个下行斜坡中。

于是 x 轴上的点 $(a+b,0)$ 位于该蛇形折线上，即有 $a+b=2kx, k\in N_+$。要使 x 最小，需要使 k 最大。由 $x\geq b$ 得 $x=\frac{a+b}{2k}\geq b$，故 $k\leq\frac{a+b}{2b}$。显有右式 ≥ 0。由于 k 为整数，取 $k=\left\lfloor\frac{a+b}{2b}\right\rfloor$。当 $k=0$ 时无解，此时 $a<b$。代入可得 x。

```c
#include <stdio.h>

int main()
{
    int i, a, b;

    while (~scanf("%d%d", &a, &b))
    {
        if (a < b)
            printf("-1\n");
        else
            printf("%.10f\n", (a + b) / (2.0 * ((a + b) / (2 * b))));
    }

    return 0;
}
```

E - 三角形切割术

考点	难度
递归、条件判断	5

题目说明

本题考察了递归和简单的条件语句。

具体实现，首先判断个位是否为零，如果为零，直接输出个位变为 1、2、3 的三角形编号即可，当尾号不为 0 时，只需从最低位（个位）向最高位进行分析，当某一位出现之前没有出现过的 1、2、3 其中的一个数字时，输出将此位换位 0 的三角形编号即可（需要注意大小顺序）。

一些例子：

- 1230 个位为 0，直接输出 1231、1232、1233。
- 112233 个位不为零，从个位开始分析，个位出现了之前没有出现过的3，所以 112230 应该为一个结果，百位出现了之前没有出现过的 2，所以 1120 应该为一个结果，同理 10 也为所要求的结果，所以最终应该输出 10、1120、112230。

样例程序

```c
#include <stdio.h>

int mark[4] = {0}, count = -1;
long long output[3];

void get_t(long long x)
{
    int n1 = x % 10;

    if (n1 == 0)
    {
        return;
    }
    else
    {
        if (mark[n1] == 0)
        {
            mark[n1] = 1;
            output[++count] = x / 10 * 10;
            if (count == 2) return;
        }
        get_t(x / 10);
    }
}

int main()
{
    long long x;
    int i;

    scanf("%lld", &x);
```

```
32
33      if (x % 10 == 0)
34          printf("%lld\n%lld\n%lld", x + 1, x + 2, x + 3);
35      else
36      {
37          get_t(x);
38          for (i = count; i >= 0; i--) printf("%lld\n", output[i]);
39      }
40
41      return 0;
42  }
```

F - qsort

难度	考点
3	`qsort` 函数

题目分析

本题按照题目要求使用 `qsort` 函数进行排序即可，题目给出的简单版本的 `compar` 函数使用 `*(int*)a - *(int*)b` 的方式判断大小关系，可能出现溢出的情况导致判断错误。希望同学们通过本题了解C语言中判断是否会溢出是程序员的任务，编写代码时需要万分小心是否会溢出。 另外，由于 `qsort` 函数声明了 `compar` 函数返回值为 `int` 类型，修改 `compar` 函数类型的方法不适用本题。

示例代码

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

int compar(const void *a, const void *b)
{
    if (*(int *)a > *(int *)b) return 1;
    if (*(int *)a < *(int *)b) return -1;
    return 0;
}

int main()
{
    int n, i;
    int a[1005] = {0};

    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }
    qsort(a, n, sizeof(int), compar);
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }

    return 0;
}
```

G - 修复石碑

难度	考点
3	数组

题目分析

题目中一共有两个角色爱斯博士和神秘组织，为了分别存储他们的修复情况，可以使用 a, b 两个数组。

此题是让我们判断谁能够首先修复数组，也就是谁的数组里 -1 首先都变成了正整数。故而可以使用两个计数器分别表示两个数组内 -1 的个数。谁的计数器首先变成了 0 ，谁就首先修复完了石碑。

特殊需要注意的是，当事件是 3 时候，需要把表示爱斯博士的 a 数组赋值给表示神秘组织的 b 数组。

下面请看代码：

示例程序

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

#define maxn 1010
int a[maxn], b[maxn], cnta = 0, cntb = 0, tmp;

int main()
{
    int n, q, i, j, fl = -1; // 使用 fl 来标记答案

    scanf("%d%d", &n, &q);
    for (i = 1; i <= n; i++)
    {
        scanf("%d", &a[i]);
        if (a[i] == -1)
            cnta++; // 如果读到了-1，就将计数器加一
    }
    while (q--)
    {
        int op;
        scanf("%d", &op); // 读入第一个表示事件的标识符
        if (op == 1)
        {
            scanf("%d%d", &i, &tmp);
            a[i] = tmp; // 将修复后的数值赋给 a 数组
            cnta--;     // 计数器减一
            if (cnta == 0 && fl == -1)
            {
                fl = 1; // 如果计数器减为 0 且之前无人修复完成，则表示首先修复
            }
        }
        else if (op == 2)
        {
            scanf("%d%d", &i, &tmp);
            b[i] = tmp; // 将修复后的数值赋给 a 数组
```

```
            cntb--;     // 计数器减一
            if (cntb == 0 && fl == -1)
            {
                fl = 2; // 如果计数器减为 0 且之前无人修复完成，则表示首先修复
            }
        }
        else
        {
            memcpy(b, a, sizeof(a)); // 数组的复制 ，也可以使用 for循环
            cntb = cnta;             // 计数器的赋值
        }
    }
    if (fl == 1)
    {
        printf("1\n"); // 爱斯博士首先恢复输出 1
        for (i = 1; i <= n; i++)
        {
            printf("%d ", a[i]); // 输出恢复后的数组
        }
    }
    else if (fl == 2)
    {
        printf("2\n"); // 神秘组织首先恢复输出 2
        for (i = 1; i <= n; i++)
        {
            printf("%d ", b[i]); // 输出恢复后的数组
        }
    }
    else
    {
        printf("-1\n"); // 无人恢复，输出 -1
    }

    return 0;
}
```

H - 金仙花数（困难版）

考点	难度
预处理，前缀和	4

题目分析

简单版の分析：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。

困难版の分析：由于数据加强，每次询问都遍历 $[l,r]$ 区间是会 `TLE` 的，所以需要进行复杂度为 $O(n)$ 的**预处理**。

给定数组 `a[3000005]`，用来表示区间 $[1,i]$ 内金仙花数的数量。其满足：

- 若 i 为金仙花数，则 `a[i] = a[i-1] + 1`。
- 否则 `a[i] = a[i - 1]`。

在开始处理 t 次询问之前，先将 $[1,i]$（$1 \le i \le 3 \times 10^6$）区间内所有金仙花数统计出来，便可以在 $O(1)$（很短的）时间内处理每次询问。

易得 $[l,r]$ 区间内的金仙花数数量 = $[1,r]$ 区间内数量 $-[1,l-1]$ 区间内数量，即 `ans = a[r] - a[l-1]`。

示例程序

```cpp
#include <stdio.h>

#define maxn 3000005

int ans[maxn];

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

    for (i = 1; i < maxn; i++)
    {
        ans[i] = ans[i - 1];
        tmp = i, a = 0, b = 1;
        while (tmp)
        {
            a += tmp % 10;
            b *= tmp % 10;
            tmp /= 10;
        }
        if (a >= 20 && b >= 162) ans[i]++;
    }

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

```
        if (ans[r] - ans[l - 1] > 0)
            printf("%d\n", ans[r] - ans[l - 1]);
        else
            puts("404 Not Found");
    }

    return 0;
}
```

I - Encode the Number!!!

难度	考点
5	数据格式、二进制、阅读

题目解析

本题背景部分翻译自 IEEE Std 754™-2019 原文。

(1)

本格式下最大的实数的编码为

```
0 11110 1111111111
S EEEEE TTTTTTTTTT
```

对应实数为

$$\mathbf{Ans1} = 1.1111111111_{(2)} \times 2^{30-15} = 65504.$$

(2)

本格式下最小正实数的编码为

```
0 00000 0000000001
S EEEEE TTTTTTTTTT
```

对应实数为

$$\mathbf{Ans2} = 0.0000000001_{(2)} \times 2^{-14} = 2^{-24} = 0.000000059604644775390625.$$

(3)

$$2020 = 11111100100_{(2)} = 1.1111100100_{(2)} \times 2^{10}.$$

对应本格式下编码为

```
0 11001 1111100100
S EEEEE TTTTTTTTTT
```

故

$$\mathbf{Ans3} = \texttt{0110011111100100}.$$

(4)

该编码为

```
1 01101 0110101010
S EEEEE TTTTTTTTTT
```

对应的值为

$$\mathbf{Ans4} = (-1)^1 \times 1.0110101010_{(2)} \times 2^{-2} = -0.35400390625.$$

(5)

$$19.52 = 10011.\dot{1}0000101000111101010\dot{1}_{(2)}.$$

取前 11 位有效数字，后续部分第一位为 0，故舍去，得到：

$$19.52 \approx 1.0011100001 \times 2^4.$$

对应本格式下编码为

```
0 10011 0011100001
S EEEEE TTTTTTTTTT
```

故

$\mathbf{Ans5} = \texttt{0100110011100001}.$

(6)

$$0.20372333 = 0.00110100001001110011011 0\ldots_{(2)}$$

这个数在二进制下的循环节长达 312500 位，这里省略。我们取前 11 位有效数字即可，第 12 位有效数字为 1，故向前进 1。我们得到：

$$0.20372333 \approx 1.1010000101 \times 2^{-3}.$$

10000 可在本格式下精确表示：

$$10000 = 10011100010000_{(2)} = 1.0011100010 \times 2^{13}.$$

两数的编码分别为：

```
0 01100 1010000101
S EEEEE TTTTTTTTTT
```

```
0 11100 0011100010
S EEEEE TTTTTTTTTT
```

将两数的 T 部分补上前置 1 和小数点后相乘：

$$1.1010000101_{(2)} \times 1.0011100010_{(2)} = 1.11111101010101101010_{(2)}.$$

取小数点后 10 位数字。第 11 位为 0，故后续部分舍去，得到尾数部分的结果：

```
x xxxxx 1111110101
S EEEEE TTTTTTTTTT
```

接下来确定其余两部分。

两正数相乘结果应为正数，符号位应为 0；

两正常值相乘，且结果没有发生上溢、下溢，也没有超出指数范围，因此指数为原两数的指数部分相加，扣除一个 $bias$：

$E = 01100_{(2)} + 11100_{(2)} - 1111_{(2)} = 11001_{(2)}$

因此结果编码为

```
0 11001 1111110101
S EEEEE TTTTTTTTTT
```

其对应的值为：

$$\text{Value} = 2^{11001_{(2)}-15} \times 1.1111110101_{(2)} = 11111110101_{(2)} = 2037.$$

故

$$\mathbf{Ans6} = 2037 - 0.20372333 \times 10000 = -0.2333.$$

示例代码

```c
#include <stdio.h>

char Ans[6][233] = {
    "65504",
    "0.00000005960464477539062 5",
    "0110011111100100",
    "-0.35400390625",
    "0100110011100001",
    "-0.2333"};

int main()
{
    int n;

    scanf("%d", &n);
    puts(Ans[n - 1]);

    return 0;
}
```

J - 高等代数基础训练

难度	考点
6	矩阵的秩，二维数组的应用

题目分析

题目等价于列向量组（行向量组） $A_{n\times m}$ 最少是多少个列向量（行向量）的线性组合，即求 $\mathrm{rank}(A_{n\times m})$。

我们需要模拟矩阵消元的过程，求出矩阵的秩，这里使用高斯-若尔当消元法：

1. 如果 $n < m$，将矩阵转置。
2. 从 1 到 m 枚举第 i 列：
   - 如果 $A_{i,i}, A_{i+1,i}, \cdots, A_{n,i}$ 均为 0，直接跳过该列。
   - 否则，将 $A_{i,i}, A_{i+1,i}, \cdots, A_{n,i}$ 中绝对值最大的那个元素所在的行交换到第 i 行以减少浮点误差。将 $A_{i,i}$ 化为 1，通过 $A_{i,i}$ 将 $A_{i+1,i}, A_{i+2,i}, \cdots, A_{n,i}$ 全部消为 0。
3. 设第 2 步中跳过了 x 列，易得 $\mathrm{rank}(A_{n\times m}) = m - x$。

示例程序

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

#define maxn 205

int main()
{
    int t, n, m, i, j, k, tmp, rank, x;
    double a[maxn][maxn], b[maxn][maxn], Tmp;

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

        if (n < m)
        {
            for (i = 1; i <= n; i++)
                for (j = 1; j <= m; j++)
                    b[j][i] = a[i][j];
            tmp = n, n = m, m = tmp;
            for (i = 1; i <= n; i++)
                for (j = 1; j <= m; j++)
                    a[i][j] = b[i][j];
        }

        rank = 0;
        for (i = 1; i <= m; i++)
```

```
        {
            x = i;
            for (j = i + 1; j <= n; j++)
                if (fabs(a[j][i]) > fabs(a[x][i]))
                    x = j;
            if (fabs(a[x][i]) < 1e-2) continue;
            rank++;

            if (x != i)
                for (j = 1; j <= m; j++)
                    Tmp = a[i][j], a[i][j] = a[x][j], a[x][j] = Tmp;
            for (j = 1; j <= m; j++)
                if (i ^ j) a[i][j] /= a[i][i];
            a[i][i] = 1;

            for (j = 1; j <= n; j++)
            {
                if (j == i) continue;
                for (k = 1; k <= m; k++)
                {
                    if (k == i) continue;
                    a[j][k] -= a[j][i] * a[i][k];
                }
                a[j][i] = 0;
            }
        }
        printf("%d\n", rank);
    }

    return 0;
}
```