A - 三线共点

难度	考点
1	if 判断

题目分析

根据数学知识，可以先求出前两条线的交点，接着验证第三条线经过该点即可。需要注意的是，最后的比较式可以将分式乘上公分母，使得比较式两侧都是整数，以避免浮点误差。

示例代码

```c
#include <stdio.h>

int main()
{
    int a, b, c, d, e, f;

    while (scanf("%d%d%d%d%d%d", &a, &b, &c, &d, &e, &f) != EOF)
    {
        printf((a - e) * (d - b) == (a - c) * (f - b) ? "Yes\n" : "No\n");
    }

    return 0;
}
```

B - 大 水 题

难度	考点
2	数组、字符串

题目分析

本题按照题目要求，将输入的字符串分别与题目提供的 5 个字符串进行比较，并按要求输出结果即可。需注意保存输入字符串的数组应该比题目要求的范围至少大1，以保证有空间存储表示字符串结束的 `\0` ，同时需注意不同字符串要求的输出间隔不同，分别为空格、换行符 `\n` 、制表符 `\t` 。

本题比较字符串是否相同，可以使用 `strcmp(字符串1, 字符串2)` 函数，它位于头文件 `string.h` 中，如果返回值为0则代表两个字符串完全相同。

示例程序

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

int main()
{
    char c[51];

    scanf("%s", c);
    if (strcmp(c, "I_Love_BUAA") == 0)
        printf("10\t25");
    else if (strcmp(c, "1952_2020") == 0)
        printf("68 1025");
    else if (strcmp(c, "SHIE_COLLEGE") == 0)
        printf("37\n73");
    else if (strcmp(c, "buaa-2020-fall-programming") == 0)
        printf("2020\t10");
    else if (strcmp(c, "Beihang-University") == 0)
        printf("1952 2020");
    else
        printf("2333\n6666");

    return 0;
}
```

C - string.h

考点	难度
字符串处理	2

题目分析

本题考察了对 `strlen`，`strcmp`，`strncat` 三个字符串处理函数的应用，同学们只要理解这三个函数的用法，即可完成本题。

示例程序

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

int main()
{
    int T, n;
    char op[5], a[105], b[105];
    int cmp;

    scanf("%d", &T);
    while (T--)
    {
        scanf("%s", op);
        if (strcmp(op, "len") == 0)
        {
            scanf("%s", a);
            printf("%d\n", (int)strlen(a));
        }
        else if (strcmp(op, "cmp") == 0)
        {
            scanf("%s%s", a, b);
            cmp = strcmp(a, b);
            if (cmp == 0)
                puts("=");
            else if (cmp < 0)
                puts("<");
            else
                puts(">");
        }
        else if (strcmp(op, "cat") == 0)
        {
            scanf("%s%s%d", a, b, &n);
            printf("%s\n", strncat(a, b, n));
        }
    }

    return 0;
}
```

D - dushuzi（简单版）

考点	难度
字符数组2	2

题目分析

由于仅有个位数，且无无意义数字，所以只需读到什么就输出什么，可以用 `if-else` 或者 `switch` 分支语句进行判断输出，也可以将所有数字的读法预先写好放入字符数组，直接用下标 `数字-'0'` 调用即可。

示例程序

```c
#include <stdio.h>

int main()
{
    char s[233];
    char num[15][15] = {"ling", "yi", "er", "san", "si", "wu", "liu", "qi", "ba", "jiu"};
    scanf("%s", s);
    int i = 0;

    for (; s[i] != '\0'; i++)
    {
        if (s[i] == '-')
            printf("fu ");
        else if (s[i] == '.')
            printf("dian ");
        else
            printf("%s ", num[s[i] - '0']);
    }

    return 0;
}
```

E - TLE 的平均值

考点	难度
字符数组2	2

题目分析

由于仅有个位数，且无无意义数字，所以只需读到什么就输出什么，可以用 `if-else` 或者 `switch` 分支语句进行判断输出，也可以将所有数字的读法预先写好放入字符数组，直接用下标 `数字-'0'` 调用即可。

本题题意是要我们设计一种查询极快的查询算法，因为总数据量很大，查询次数也很多，如果每次查询都遍历区间里所有数字的话肯定会 TLE。

我们采用“缓存”的思想，即在读入 n 个数字的时候就计算好部分数据并存起来，在查询的时候直接取出来用就好，这种思想实际上在用空间换时间，存的数据越多，时间花费越少。即俗话说的“好记性不如烂笔头”。

在本题中，我们存储的数据是前 n 个数的和。查询第 L 个数字到第 R 个数字的平均值的时候，使用前 R 个数字的和减去前 $(L-1)$ 个数字的和再除以 $(R-L+1)$ 即可得答案，不用遍历所有 $(R-L+1)$ 个数字，查询效率很高。

示例程序

```c
#include <stdio.h>
int a[1000010];
long long sum[1000010];

int main()
{
    int m, n;
    int i, l, r;

    scanf("%d%d", &m, &n);

    for (i = 1; i <= n; i++)
    {
        scanf("%d", &a[i]);
        sum[i] = a[i];
        sum[i] += sum[i - 1];
    }
    for (i = 1; i <= m; i++)
    {
        scanf("%d%d", &l, &r);
        printf("%lld\n", (sum[r] - sum[l - 1]) / (r - l + 1));
    }

    return 0;
}
```

F - 现在我也是 scanf 了

难度	考点
2	数组、字符串

题目分析

本题按照题目要求，将输入的大数看做是一个字符串，之后新建一个 `int` 数组来存储这个大数的每一位，我们就完成了一个高精度整数的读入。

之后给这个高精度整数的最低位 $+1$，然后处理下进位输出即可。

示例程序

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

char a[105];
int b[105];

void process()
{
    int i;
    for (i = 0; i < strlen(a); i++)
        b[i + 1] = a[i] - '0';
    b[strlen(a)]++;
    for (i = strlen(a); i >= 0; i--)
    {
        if (b[i] > 9)
        {
            b[i] -= 10;
            b[i - 1]++;
        }
    }
}

int main()
{
    int i;

    scanf("%s", a);
    process();
    if (b[0])
    {
        for (i = 0; i <= strlen(a); i++)
            printf("%d", b[i]);
    }
    else
    {
        for (i = 1; i <= strlen(a); i++)
            printf("%d", b[i]);
    }

```

```
    return 0;
}
```

G - 方形「Square Creature」

难度	考点
3	二维数组、字符数组

题目分析

本题的核心是二维数组，画图过程即是对二维数组指定位置赋值的过程。

本题需要注意的点如下：

- 注意题中对大小、位置描述的横竖、行列，以及对行列描述是从 0 还是从 1 开始；
- 注意画布外的图形的处理。大致有以下两种处理方式：
  - 不绘制画布外的图形；
  - 绘制画布外的图形，但不输出。

  如果采用第一种处理方式，则可以用一个函数代替常规的二维数组赋值表示作画，在函数里特判范围，以免越界（函数示例如下）；如果采用第二种处理方式，则要注意输出范围，不要输出画布外的内容，同时还要注意二位数组需要开大一些，以免下标越界。

```
// 第一种处理方式函数示例

char canvas[100][101];  // 定义画布
int m, n;   // 画布大小

void draw(int x, int y, char c) {   // 该函数代替 canvas[x][y] = c
    if (x >= m || y >= n) {
        return;
    }
    canvas[x][y] = c;
}
```

- 注意初始化二维数组为空格（注意不是将数组初始化为 0，空格是 ASCII 码为 32 的一个可见字符）。此外，初始化最好老老实实地用二重循环，若要用 `memset` 函数，则要注意以下几点（**这些注意点不仅限于这道题**）：
  - 谨慎判断要初始化的内存空间大小。如果小了，则无法起到初始化的效果；如果大了，则会破坏其他内存空间，可能会破坏其他变量的值；
  - `memset` 初始化的内存空间大小以字节为单位，而 `sizeof` 函数返回目标的大小，同样以字节为单位。如 `sizeof(char) * n` 表示一个大小为 `n` 的字符数组大小；
  - `memset` 初始化的内存空间大小以字节为单位，故初始化 `int` 数组为 `0` 之外的值时**谨慎**使用该函数，因为 `int` 变量的大小是 4 字节。
- 输出方法大致可分为以下两种：
  - 按字符挨个输出，每一行手动换行；
  - 用 `puts` 函数或 `printf("%s")` 格式化输出按行输出。

  如果采用第二种方式输出，记得字符串应以 `\0` 结束。

题解中将绘制三种方形的方式封装成了函数。其中第 1 种方形采用了递归函数的思路，从最外层开始向内绘制。

示例程序

```c
#include <stdio.h>

char canvas[300][300];

void d1(int y, int x, int a, int b, char c) // 递归过程，其中 x, y 代表位置，a, b 代表大小，c 代表正在绘制的这一层的字母
{
    if (a <= 0 || b <= 0)
    { // 递归终止条件
        return;
    }
    int i = 0;
    for (i = 0; i < a; i++)
    {
        canvas[y - 1][x - 1 + i] = c;
        canvas[y - 2 + b][x - 1 + i] = c;
    }
    for (i = 0; i < b; i++)
    {
        canvas[y - 1 + i][x - 1] = c;
        canvas[y - 1 + i][x - 2 + a] = c;
    }
    d1(y + 1, x + 1, a - 2, b - 2, c + 1); // 递归绘制内层
}

void d2(int y, int x, int a, int b, int c)
{
    int i, j;
    for (i = 0; i < b; i++)
    {
        for (j = 0; j < a; j++)
        {
            canvas[y - 1 + i][x - 1 + j] = c;
        }
    }
}

void d3(int y, int x)
{
    canvas[y - 1][x - 1] = '/', canvas[y - 1][x] = '-', canvas[y - 1][x + 1] = '\\';
    canvas[y][x - 1] = '|', canvas[y][x] = 'o', canvas[y][x + 1] = '|';
    canvas[y + 1][x - 1] = '\\', canvas[y + 1][x] = '-', canvas[y + 1][x + 1] = '/';
}

int main()
{
    int t, m, n, a, b, c, y, x;
    int i, j;

    scanf("%d%d", &m, &n);
    for (i = 0; i < m; i++)
    {
```

```
        for (j = 0; j < n; j++)
        {
            canvas[i][j] = ' ';
        }
    }
    while (~scanf("%d%d%d", &t, &y, &x))
    {
        if (t == 1)
        {
            scanf("%d%d", &a, &b);
            d1(y, x, a, b, 'A');
        }
        else if (t == 2)
        {
            scanf("%d%d%d", &a, &b, &c);
            d2(y, x, a, b, c);
        }
        else if (t == 3)
        {
            d3(y, x);
        }
    }

    for (i = 0; i < m; i++)
    { // 将每行字符串结尾置为 '\0'
        canvas[i][n] = 0;
    }

    for (i = 0; i < m; i++)
    {
        puts(canvas[i]);
    }

    return 0;
}
```

H - 食堂排队

难度	考点
3	二分查找

题目分析

通过题目描述，我们可以提取出这样一个模型：即给定一个单调递增序列，查询某一数字是否在该序列中。

观察数据范围，序列长度最大为 10^6，查询次数最大也为 10^6，如果我们每次采用遍历序列的方法查询，那么循环次数最大为 10^{12} 次，一定会超时，因此我们这里采用二分法来查找。我们知道二分法时间复杂度为 $O(\log_2 n)$，那么对于本题来说循环次数大约是 $10^6 \cdot \log_2(10^6)$ 次。

示例程序

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

int a[1000000];

int main()
{
    int n, t;
    int i;
    int x, l, r, mid;

    scanf("%d%d", &n, &t);
    for (i = 0; i < n; i++) // 读入序列
    {
        scanf("%d", &a[i]);
    }
    for (i = 0; i < t; i++) // t 次查询
    {
        scanf("%d", &x);
        l = 0, r = n - 1; // 每次二分的初始区间
        while (l <= r)    // 二分查找失败的边界条件：l > r
        {
            mid = (l + r) / 2;
            if (a[mid] == x)
            {
                printf("true\n");
                break;
            }
            else if (a[mid] < x)
                l = mid + 1; // 小了，往右半区间查
            else
                r = mid - 1; // 大了，往左半区间查
        }
        if (l > r) printf("false\n");
    }

```

```
    return 0;
}
```

I - 网名真难取

难度	考点
4	排序、字符串比较

题目分析

本题主要考察排序。只需在比较时注意按题目要求进行比较即可。比较时，这种有多个可能性的比较判断结构，建议先构思清楚再动手敲代码，并善用 `break` 和 `continue` 使程序简洁清晰。无论用何种排序方法，建议将排序主体作为一部分，比较作为另一部分进行模块化编程，这样程序结构清晰易于查错。

示例程序

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

int n;
char s[101][101], tmp[101];

int compare(char s1[], char s2[])
{
    int l1 = strlen(s1);
    int l2 = strlen(s2);

    if (l1 > l2)
        return 1;
    if (l1 < l2)
        return 0;
    for (int i = 0; i < l1; i++)
    {
        if (s1[i] == s2[i])
            continue;
        if (s1[i] == '6')
            return 0;
        if (s2[i] == '6')
            return 1;
        if (s1[i] > s2[i])
            return 0;
        if (s1[i] < s2[i])
            return 1;
    }

    return 0;
}

int main()
{
    int i, j;

    scanf("%d", &n);
    for (i = 0; i < n; i++)
```

```c
        scanf("%s", s[i]);

    // bubble sort
    for (i = 0; i < n - 1; i++)
        for (j = 0; j < n - i - 1; j++)
            if (compare(s[j], s[j + 1]) == 1)
            {
                strcpy(tmp, s[j]);
                strcpy(s[j], s[j + 1]);
                strcpy(s[j + 1], tmp);
            }

    for (i = 0; i < n; i++)
        printf("%s\n", s[i]);

    return 0;
}
```

J - Fourier Slow Transform

难度	考点
6	数组的应用，向量组的秩

题目分析

orz 也是一种位运算，可以发现它和按位异或运算等价，因此所有能得到的数均在 $[0, 2047]$ 的范围。

设 $a_{i,j} \in \{0, 1\}$ 表示从 S 中 i 个数（可以重复）进行异或操作能否得到 j 个数，如果可以为 1 否则为 0 。

我们现在已经有了 $a_{1,0}, a_{1,1}, \cdots, a_{1,2047}$：

$$a_{1,j} = \begin{cases} 1, & j \in S \\ 0, & j \notin S \end{cases}$$

考虑如何从 $a_{i,0}, a_{i,1}, \cdots, a_{i,2047}$ 推出 $a_{i+1,0}, a_{i+1,1}, \cdots, a_{i+1,2047}$，有：

$$a_{i+1,j} = \exists (x, y \in [0, 2047])[(a_{i,x} = 1) \text{ and } (x \operatorname{xor} y = j) \text{ and } (y \in S)]$$

我们可以二重循环枚举 x 和 y 来完成这一步。

这样我们一共要做 $m - 1$ 次上述操作，从 S 中 i 个数（可以重复）进行异或操作得到的数所组成的集合的元素之和即为 $\sum_{j=0}^{2047} j a_{i,j}$ 。这样时间复杂度为 $O(n^2 m)$，不足以通过本题。

按照每一位来看，异或操作相当于模 2 且不进位的加法，如果我们把两个长度为 n 的二进制串异或起来，可以看作将两个 n 维向量在模 2 意义下相加。

S 中的所有数都可以写成长度为 11 的二进制串（更高位必然都是 0），我们将 S 中的数都写成二进制，则形成了一个 n 个 11 维向量的向量组，这个向量组的秩最多为 11，因此任意一个数至多需要 11 个数异或就能被表示出来。

需要注意的是，奇数个向量线性组合出来的向量组不一定和偶数个向量线性组合出来的向量组完全一致，因此我们需要做 12 次上述操作，当 $i > 12$ 时输出 $i - 2$ 的答案即可，时间复杂度为 $O(n^2 \log n)$ ，可以通过本题。

示例程序

```
#include <stdio.h>

#define maxn 2505

const int M = 2048;

int main()
{
    int t, n, m, x, i, j, k, a[maxn], b[maxn], c[maxn], ans[maxn];

    scanf("%d", &t);
    while (t--)
    {
        scanf("%d%d", &n, &m);
```

```
        memset(a, 0, sizeof(a));
        memset(b, 0, sizeof(b));
        for (i = 1; i <= n; i++)
            scanf("%d", &x), a[x] = b[x] = 1;
        for (i = 2; i <= m; i++)
        {
            if (i > 12)
            {
                ans[i] = ans[i - 2];
            }
            else
            {
                memset(c, 0, sizeof(c));
                for (j = 0; j < M; j++)
                {
                    for (k = 0; k < M; k++)
                    {
                        if (!a[j] || !b[k]) continue;
                        c[j ^ k] = 1;
                    }
                }
                ans[i] = 0;
                for (j = 0; j < M; j++)
                    ans[i] += (a[j] = c[j]) ? j : 0;
            }
            printf("%d\n", ans[i]);
        }
    }

    return 0;
}
```