E7

D 自定义格式化

sprintf用处举例：

输入：两个浮点数 a，b，一个整数 n

输出：输出 a/b ，保留n位小数

样例输入

```
2.0 3.0 5
```

样例输出

```
0.66667
```

提示

实现类似于 printf("%.nf", c); 的程序（n为变量）

代码示例

```
#include <stdio.h>
int main() {
    double a, b, c;
    int n;
    scanf("%lf%lf%d", &a, &b, &n);
    c = a / b;

    char format[20];
    sprintf(format, "%%.%df", n);
    printf(format, c);
}
```

sscanf用处举例:

多组输入：n行学生信息，每行包含姓名(字符串)和期末成绩(浮点数)，空格分开，**输入顺序不固定**

输出：按期末成绩的**字典序**排名，每行为 姓名 期末成绩(保留两位小数)

0 < n <= 1000, 字符串长度 <= 100

样例输入

```
4
violet 100.00
91.27 blue
94.36 yellow
white 95.88
```

样例输出

```
violet 100.00
white 95.88
yellow 94.36
blue 91.27
```

提示

- 字符串读取整行，根据首字符判断两种输入顺序
- sscan读到**字符串数组/字符指针数组**和浮点数数组，排序后输出
- 姓名当作字符串处理，期末成绩当作浮点数处理。冒泡排序，**同时交换**

选择指针数组还是二维字符数组?

char *name[1005];

char name[1005][105];

代码示例1（冒泡）

```c
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
char *name[1005];// 存储n个学生的姓名
double score[1005]; // 存储n个学生的成绩
int main() {
    int n;
    scanf("%d", &n);
    char input[200];

    // 解析输入
    getchar(); // 用gets前读掉整数n后面的换行符
    for (int i = 0; i < n; i++) {
        gets(input);
        name[i] = (char *)malloc(105);
        if (isalpha(input[0])) {
            // 姓名在前
            sscanf(input, "%s%lf", name[i], &score[i]);
        } else {
            // 成绩在前
            sscanf(input, "%lf%s", &score[i], name[i]);
        }
    }

    // 冒泡排序
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < n - i; j++) {
            if (score[j] < score[j + 1]) {
                double temp = score[j];
                score[j] = score[j + 1];
                score[j + 1] = temp;

                char *ptemp = name[j];
                name[j] = name[j + 1];
                name[j + 1] = ptemp;
            }
```

```
        }
    }

    // 按序输出
    for (int i = 0; i < n; i++) {
        printf("%s %.2f\n", name[i], score[i]);
    }
}
```

代码示例2（qsort）

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
char *name[1005];// 存储n个学生的姓名
double score[1005]; // 存储n个学生的成绩
int order[1005];
// order[0] = 3代表排名第一的同学的名字和成绩分别为 name[3], score[3]
// 初值为从0-n的连续整数序列，即order[i] = i;

int cmpfunc(const void *a, const void *b) {
    int x = *(int *)a;
    int y = *(int *)b;
    if (score[y] > score[x]) {
        return 1;
    } else if (score[y] < score[x]) {
        return -1;
    } else {
        return 0;
    }
}
int main() {
    int n;
    scanf("%d", &n);
    char input[200];

    // 解析输入
    getchar(); // 用gets前读掉整数n后面的换行符
    for (int i = 0; i < n; i++) {
        gets(input);
        name[i] = (char *)malloc(105);
        if (isalpha(input[0])) {
            // 姓名在前
            sscanf(input, "%s%lf", name[i], &score[i]);
        } else {
            // 成绩在前
            sscanf(input, "%lf%s", &score[i], name[i]);
        }
        order[i] = i; // 设定初始值
    }

    // qsort对order[i]排序
    qsort(order, n, sizeof(order[0]), cmpfunc);

    // 按序输出
```

```
    for (int i = 0; i < n; i++) {
        printf("%s %.2f\n", name[order[i]], score[order[i]]);
    }
}
```

对于代码示例2，name用二维字符字符数组和指针数组的效率相同

二分 Killer!

问题引入

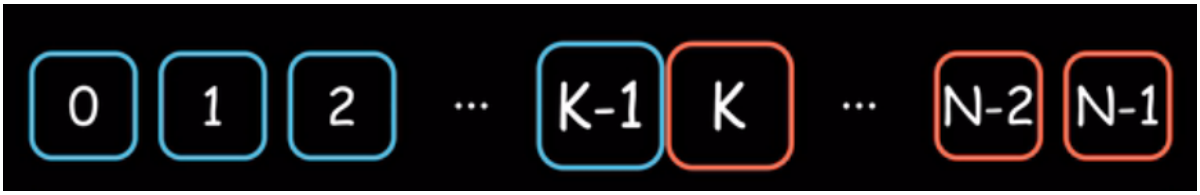
对于序列 $1，2，3，5，5，5，8，9$

数值	1	2	3	5	5	5	8	9
下标	0	1	2	3	4	5	6	7

先用线性遍历求出正确答案。

问题列表	答案（下标）
1. 找到最后一个 < 5 的元素	2
2. 找到第一个 >= 5 的元素	3
3. 找到最后一个 <= 5 的元素	5
4. 找到第一个 > 5 的元素	6

- 转换思路，将**“元素位置的查找”**转变为**"红蓝边界的划分"**



1. 第一步，**划分红蓝区域**

- 问题**1，2** ： $1，2，3，5，5，5，8，9$
- 问题**3，4**： $1，2，3，5，5，5，8，9$

2. 第二步，确定**"isblue()"** 函数

- 问题**1，2：x < 5**
- 问题**3，4**： **x <= 5**

3. 第三步，根据要求返回 l 或 r

- 问题**1，3**： **返回** l
- 问题**2，4**： **返回** r

寻找过程

1. 变量 l 始终指向 蓝色区域 **的最右端**， 变量 r 始终指向红色区域 **的最左端**

   所以 l 的初值为 -1， r 的初值为 N

2. $m = (l + r)/2$ （向下取整）

3. ```
   if (isblue(a[m])) {
       l = m;
   } else {
       r = m;
   }
   ```

4. 之所以 l, r 不赋值为 $m + 1$或$m - 1$， 是为了保证**第一条**

5. 回到第二条，继续循环。

6. 循环结束条件？**while (l + 1 != r)**

   即：当 $l + 1 == r$ 时，l 就是蓝色区域右边界，r 就是红色区域左边界

7. 返回值： 根据题目要求返回 l 或 r

代码模板

```
int blue_bsearch(int a[], int aSize, int key) {
    int l =  -1, r = aSize, m;
    while (l + 1 != r) {
       m = l + (r - l) / 2; // 这种写法保证中间变量不会超 int 范围
       if (isblue(a[m])) { // isblue函数按上述分析自行确定
           l = m;
       } else {
           r = m;
       }
        return l 或者 r;
    }
}
```

题目1代码样例（找到最后一个 < 5 的元素）

```
int blue1_bsearch(int a[], int aSize, int key) {
    int l = -1, r = aSize, m;
    while (l + 1 != r) {
        m = l + (r - l) / 2;
        if (a[m] < key) {
            l = m;
        } else {
            r = m;
        }
    }
    return l;
}
```
```

题目2代码样例（找到第一个 >= 5 的元素）

```
int blue2_bsearch(int a[], int aSize, int key) {
    int l = -1, r = aSize, m;
    while (l + 1 != r) {
        m = l + (r - l) / 2;
        if (a[m] < key) {
            l = m;
        } else {
            r = m;
        }
    }
    return r;
}
```

题目3代码样例（找到最后一个 <= 5 的元素）

```
int blue3_bsearch(int a[], int aSize, int key) {
    int l = -1, r = aSize, m;
    while (l + 1 != r) {
        m = l + (r - l) / 2;
        if (a[m] <= key) {
            l = m;
        } else {
            r = m;
        }
    }
    return l;
}
```

题目4代码样例（找到第一个 > 5 的元素）

```
int blue4_bsearch(int a[], int aSize, int key) {
    int l =  -1, r = aSize, m;
    while (l + 1 != r) {
        m = l + (r - l) / 2;
        if (a[m] <= key) {
            l = m;
        } else {
            r = m;
        }
    }
    return r;
}
```

F 川川爬山

题目解析

假设序列中存在 key

1. 求 **第一个** $>=$ key **的元素**的位置 $loc1$
2. 求 **最后一个** $<=$ key **的元素**的位置 $loc2$

3. 出现的次数为 $loc2 - loc1 + 1$

若不存在则必有 $loc1 \neq loc2$

代码示例

```
#include <stdio.h>
// 查找第一个 >= key的元素 （首位）
int blue2_bsearch(int a[], int aSize, int key) {
    int l = -1, r = aSize, m;
    while (l + 1 != r) {
        m = l + (r - l) / 2;
        if (a[m] < key) {
            l = m;
        } else {
            r = m;
        }
    }
    return r;
}
// 查找最后一个 <= key的元素
int blue3_bsearch(int a[], int aSize, int key) {
    int l = -1, r = aSize, m;
    while (l + 1 != r) {
        m = l + (r - l) / 2;
        if (a[m] <= key) {
            l = m;
        } else {
            r = m;
        }
    }
    return l;
}

int a[1000005];
int main()
{
    int n, t;
    scanf("%d%d", &n, &t);
    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }

    for (int i = 0; i < t; i++) {
        int key;
        scanf("%d", &key);
        int loc1 = blue2_bsearch(a, n, key);
        int loc2 = blue3_bsearch(a, n, key);
        if (loc1 <= loc2) {
            printf("%d %d\n", loc1 + 1, loc2 - loc1 + 1);
        } else {
            printf("-1\n");
        }
    }
    return 0;
}
```

递归

二要素

1. 递推公式
2. 递归出口

规模大的、较难解决的问题**变成规模较小**的、易解决的**同一问题（调用自身）**。规模较小的问题又变成规模更小的问题，并且小到一定程度可以**直接得出它的解（递归出口）**，从而得到原来问题的解。

何时使用递归?

1. 求 $f(n)$，而 $f(n)$ 的值和 $f(m)(m \leq n)$ 有关。

- **C5 E Catalan 数**

$Catalan$ 数是组合数学中常见的一个数列，可以用于表示 n 个节点组成不同构二叉树的方案数，$n+2$ 边形被分割成三角形的方案数等。其值计算方式为：

$$C_n = \begin{cases} 1 & n = 0 \\ \sum_{i=0}^{n-1} C_i \cdot C_{n-1-i} & n \geq 1 \end{cases}$$

现在给出若干个 n，请你计算相应的 C_n

- **C5 J 哪吒的递归函数**

函数 $f(x)$ 定义如下：

$$f(x) = \begin{cases} 1 & \text{, if } x = 0 \\ f\left(\left\lfloor \frac{x}{2} \right\rfloor\right) + f\left(x - 2^{\lfloor \log_2 x \rfloor}\right) & \text{, else} \end{cases}$$

其中 $\lfloor \ \rfloor$ 表示向下取整。

注意到 $\left\lfloor \frac{x}{2} \right\rfloor$ 是将 x 的二进制表示的最低位去掉，$x - 2^{\lfloor \log_2 x \rfloor}$ 是将 x 的二进制表示的最高位去掉。

给出若干个 x，请你计算 $f(x)$。

2. 可以将问题抽象为一个函数 $f(n)$，而 $f(n)$ 的值和 $f(m)(m \leq n)$ 有关。

- **C5 G 戎樱花的汉诺塔**

  $f(n)$：传入一个整数n，输出将n层汉诺塔从一侧移动到另一侧的步骤

  $f(n)$ 可以通过 $f(n-1) + f(1) + f(n-1)$ 实现

题目描述

戎璎花今天也在赛之河原开心的举行着堆石子比赛，今天的主题是**汉诺塔**！

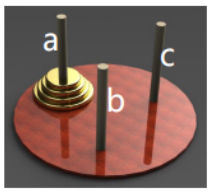
戎璎花热情的邀请你来当解说员，解说她玩汉诺塔的过程！

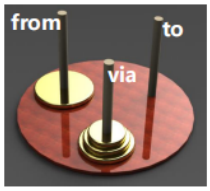
在比赛过程中有：

1. 比赛使用的是三柱汉诺塔，这三个柱子各有标记，分别为一个大写字母。
2. 在左边的柱上有 n 个圆盘，从上到下，每个圆盘依次被编号为：$1, 2, 3, \ldots, n$
3. 戎璎花一次只能搬动一片圆盘
4. 每个圆盘只能放在编号更大的圆盘之上
5. 戎璎花是垒石头的高手！她会以最少的步数将汉诺塔移到最右边的柱子上！

Hanoi塔的递归实现

例5-15：汉诺塔 Hanoi Tower





函数重在接口，递归重在调用

这个 n 的意思是有 n 个盘子的hanoi塔。
本函数把 n 个盘子从柱子 a 通过 b 挪到 c 上。

这个 i 的意思是第 i 号盘。本函数把第 i 号盘子
从柱子 from 直接挪到 to 上。

```c
void hanoi(int n, char a, char b, char c); // 接口清晰
void move(int i, char from, char to);

int main()
{
    int num;
    char a = 'A', b = 'B', c ='C';
    scanf("%d", & num);
    hanoi(num, a, b, c);
    return 0;
}

void hanoi(int n, char from, char via, char to)
{
    if (n == 1)
    {
        move(n, from, to);
        return;
    }
    hanoi(n - 1, from, to, via); // 调用正确
    move(n, from, to);
    hanoi(n - 1, via, from, to); // 调用正确
}

void move(int i, char from, char to)
{
    printf("disk %d, %c --> %c\n", i, from, to);
}
```

- **C8 哪吒玩汉诺塔**

  $f(m,n)$: 传入参数 m,n ，返回移动一个 m 层汉诺塔的第 n 步是第几层。

题目描述

哪吒喜欢玩汉诺塔，但是他总记不住第几步应该移动哪个圆盘，你能帮帮他吗？

给出汉诺塔的层数 m 和移动步数 n ，求在移动一个 m 层汉诺塔的过程中，第 n 步是在移动第几小的圆盘（即最初汉诺塔从上到下的第几层圆盘）。

如果忘记了汉诺塔的规则可以参考 **Hint** 。

自顶而下分析。移动汉诺塔的方法为：先将 $m-1$ 层汉诺塔从 A 柱移动到 B 柱，再将第 m 层圆盘从 A 柱移动到 C 柱，再将 $m-1$ 层汉诺塔从 B 柱移动到 C 柱，共需要 2^m-1 次移动。

因此移动一个 m 层汉诺塔的第 2^{m-1} 步是在移动第 m 层的圆盘，前 $2^{m-1}-1$ 步是在移动 $m-1$ 层汉诺塔，后 $2^{m-1}-1$ 步也是在移动 $m-1$ 层汉诺塔。

据此我们可以使用递归求出答案。设 $f(m,n)$ 表示移动一个 m 层汉诺塔的第 n 步移动的是第几层，则

- 若 $n<2^{m-1}$，则 $f(m,n)=f(m-1,n)$;
- 若 $n>2^{m-1}$，则 $f(m,n)=f(m-1,n-2^{m-1})$;
- 若 $n=2^{m-1}$，则 $f(m,n)=m$（基本情况）。

- **E5 G 格雷码**

  $f(n)$: 传入整数n，输出n位格雷码

  $f(n)$ 可以由 $f(n-1)$ 经过一些简单操作后求出

格雷码是一种二进制编码方式，它用 n 位的二进制来表示数。 与普通的二进制表示不同的是，它要求相邻两个数字只能有1个数位不同。首尾两个数字也要求只有1位之差。

下面是格雷码的递归构造方法：

1. 1 位格雷码是 $0,1$
2. $(n+1)$ 位格雷码中的前 2^n 个码字等于 n 位格雷码的码字，按顺序书写，加前缀 0
3. $(n+1)$ 位格雷码中的后 2^n 个码字等于 n 位格雷码的码字，按逆序书写，加前缀 1
4. $(n+1)$ 位格雷码的集合 $=n$ 位格雷码集合(顺序)加前缀 0 + n 位格雷码集合(逆序)加前缀 1

请按顺序输出 n 位格雷码。

3. dfs（深度优先搜索）类型

- **C7 | 拔刀**

通过率：/ (%)　正解率：/ (%)

题目描述

将军的战刀由于用力过猛被卡在石头缝里了，八重宫司希望你能计算出战刀剩余的长度。

给定一个高为 h，宽为 w 的矩阵，该矩阵只包含 $0, 1$ 两种元素，元素间以空格隔开，其中 0 代表石头，1 代表刀身， 请你计算从**第一行**开始，刀剩余的最长长度时多少？即，计算最多有多少行存在元素 1 能够与第一行的元素 1 **连通。**

本题中连通指，两个元素 1 间存在一条通路，该通路上的每一个元素均为 1（包括端点）。

输入

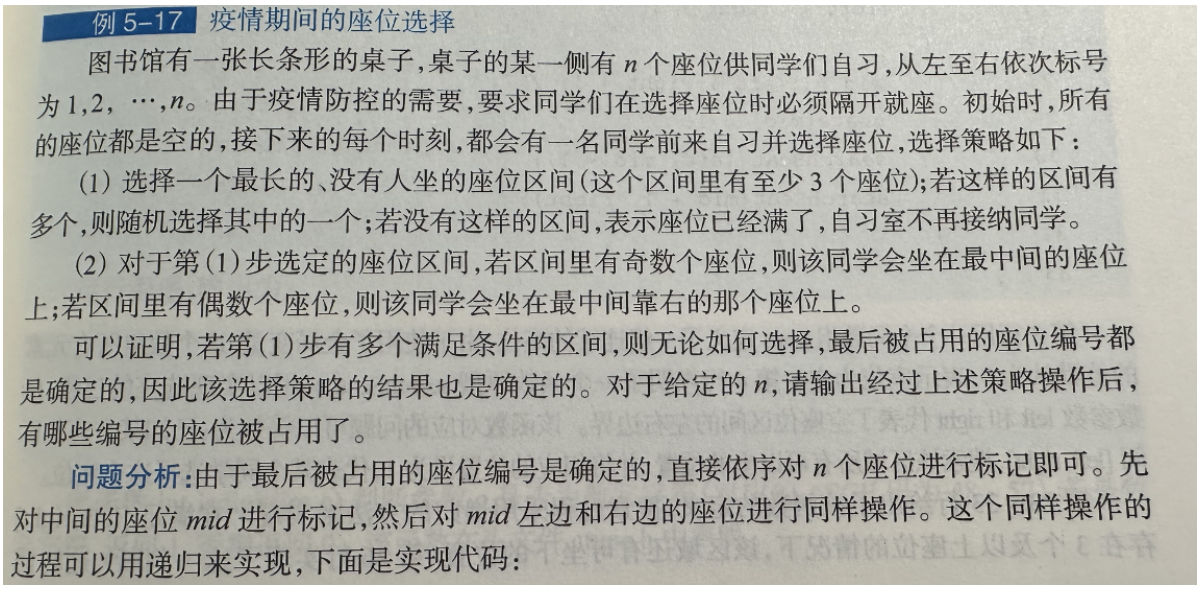
- **E6 | Permutation?**

题目描述

数列 $(p_1, p_2, \cdots, p_n)$ 是集合 $\{1, 2, \cdots, n\}$ 的排列当且仅当集合的每个元素都在数列中恰好出现一次。例如，$(3, 1, 4, 2, 5)$ 是 $\{1, 2, 3, 4, 5\}$ 的一个排列。

对于给定的 n ， 我们希望知道存在多少个集合 $\{1, 2, \cdots, n\}$ 的排列 $(p_1, p_2, \cdots, p_n)$ 使得 $2 \leq |p_i - p_{i+1}| \leq 3$ 对任意的 $1 \leq i < n$ 都成立。

请同学们课后自学课本上的递归函数章节，例5-17



例 5-17　疫情期间的座位选择

图书馆有一张长条形的桌子，桌子的某一侧有 n 个座位供同学们自习，从左至右依次标号为 $1, 2, \cdots, n$。由于疫情防控的需要，要求同学们在选择座位时必须隔开就座。初始时，所有的座位都是空的，接下来的每个时刻，都会有一名同学前来自习并选择座位，选择策略如下：

(1) 选择一个最长的、没有人坐的座位区间（这个区间里有至少 3 个座位）；若这样的区间有多个，则随机选择其中的一个；若没有这样的区间，表示座位已经满了，自习室不再接纳同学。

(2) 对于第 (1) 步选定的座位区间，若区间里有奇数个座位，则该同学会坐在最中间的座位上；若区间里有偶数个座位，则该同学会坐在最中间靠右的那个座位上。

可以证明，若第 (1) 步有多个满足条件的区间，则无论如何选择，最后被占用的座位编号都是确定的，因此该选择策略的结果也是确定的。对于给定的 n，请输出经过上述策略操作后，有哪些编号的座位被占用了。

问题分析：由于最后被占用的座位编号是确定的，直接依序对 n 个座位进行标记即可。先对中间的座位 *mid* 进行标记，然后对 *mid* 左边和右边的座位进行同样操作。这个同样操作的过程可以用递归来实现，下面是实现代码：