C8 - Solution

A 诚信守则

难度	考点
1	输入输出、字符串

题目分析

读完题目后，我们发现只需要判断读入的是哪个串，进行相应的输出就可以。判断是哪个串我们的第一反应是使用 `strstr`，但是实际上不用那么麻烦，因为题目保证了输入只可能是这两个串其中一个，所以我们只看第二个字符就能看出是哪一个串。

示例代码

```c
#include <stdio.h>

char str[105];
int main() {
    gets(str);
    if (str[1] == ' ')
        printf("If one loses honesty, he will lose everything.");
    else
        printf("I promise I do not cheat in the exam.");
    return 0;
}
```

B Softplus

难度	考点
1	浮点数、函数

题目分析

根据题目中给出的表达式调用对应的库函数进行输入输出即可。

示例代码

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

int main() {
    double f;
    while(~scanf("%lf", &f)) {
        // 也可以把 log(1 + exp(f)) 封装成一个函数
        printf("%.4lf\n", log(1 + exp(f)));
    }
    return 0;
}
```

C Hello World, and Beyond

难度	考点
1	输入输出、字符转义

题目分析

输出样例中给出的图案即可，当然别忘了处理一些需要转义的字符。`'`、`"` 和 `\` 都是输出时需要转义的字符，可以在前面加上 `\` 进行输出；还有一个较为常用的是 `%`，输出时要写 `%%`。

对每一行写一个 `printf("xx\n")` 即可，其中 xx 是经过转义处理的字符串。下面题解中给出了一个使用了续行符机制的技巧，续行符（实际上就是 `\`）用来表示续行符的下一行与续行符所在的代码是同一行（具体参考下面的代码）。

示例代码

```c
#include <stdio.h>

int main() {
    // 注意下面每一行最后的 \，代表续行
    printf("  ___ ___         .__  .__           __      __           .__
._\n\
 /   |   \\   ____ |  | |  |   ____   /  \\    /  \\___________|  |    __| |\n\
/    \'    \\_/ __ \\|  | |  |  /  _ \\  \\   \\/\\/   /  _ \\_  __ \\  |   / __
|\n\
\\    Y    /\\  ___/|  |_|  |_(  \"_\" )  \\        (  \"_\" )  | \\/  |__/ /_/
|\n\
 \\___|___/  \\___  \"____/____/\\____/    \\__/\\__/ \\____/|__|
|____/\\_____|\n");
    return 0;
}
```

扩展补充

可以尝试写一个程序，用来自动生成输出对应字符画的程序。

D 位运算计算器

难度	考点
2	位运算，输入输出

题目分析

简单来看，该题分为两部分，第一部分就根据读入的操作式计算相应的结果，第二就是求结果中有多少位是 1。

第一个任务中需要注意的就是操作符前后都会有空格，如果直接用 `scanf("%d%c%d", &a, &op, &b);` 那么就会把空格读进来，而没有真正读入操作符，一种简单的方法是在 %c 前面加一个空格（要注意在 scanf 中写空格将匹配所有对应位置的空格和换行符，而并非单纯表示匹配一个空格，只建议在读入字符时且不想读入换行符和空格的情况下，在 %c 前面加空格）。

第二个任务也做过很多次了，可以参考 hint 中的循环。

示例代码

```
#include <stdio.h>

int popcount(unsigned x) {
    int ret = 0;
    for(int i = 0; i <= 31 ;i++)
        ret += (x >> i) & 1;
    return ret;
}

int main() {
    unsigned a, b;
    char op;
    while(~scanf("%d %c%d", &a, &op, &b)) {
        unsigned c;
        // 下面的结构也可以用 switch
        if(op == '&')
            c = a & b;
        else if(op == '|')
            c = a | b;
        else
            c = a ^ b;
        printf("%u %d\n", c, popcount(c));
    }

    return 0;
}
```

E 小懒獭与半素数

难度	考点
3	流程控制、判断素数

题目分析

关于如何判断一个数是不是质数，这个我们已经练习讲解过许多次了，算是前置知识。

这道题中，判断一个数字是不是半素数，相当于看一个数是不是由且仅由两个素数乘积得到。显然，如果一个数字是半素数，那么它必有一个素因子不大于 $\sqrt{n}$ 。所以最简单的一个方法我们只需要从 2 到 $\sqrt{n}$ 进行枚举，找到能整除 n 的数字 k ，看 k 和 $\frac{n}{k}$ 是不是同为素数。

该题在数据上只要能想到只枚举 $\sqrt{n}$ 即可通过，但是显然有更好的优化方法，比如如果找到了一个能整除 n 的数字 k ，如果 k 和 $\frac{n}{k}$ 是不是同为素数，显然 n 已经不可能是一个半素数了。

示例代码

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

int isPrime(int x) {
    if (x < 2) return 0;
    for (int i = 2; i <= sqrt(x); i++) {
        if (x % i == 0) return 0;
    }
    return 1;
}

// 如果是半素数，函数本身返回 1，同时利用指针传递两个质数因子
// 这里利用了一个指针向函数外返回多个值的技巧
int isSemiPrime(int n, int *p, int *q) {
    for (int i = 2; i <= sqrt(n); i++) {
        if (n % i == 0) {
            if(isPrime(i) && isPrime(n / i)) {
                *p = i;
                *q = n / i;
                return 1;
            } else
                return 0;
        }
    }
    // 证明这个数本身是一个素数
    return 0;
}

int main() {
    int n;
    while(~scanf("%d", &n)) {
        int p, q;
        if (isSemiPrime(n, &p, &q)) {
            printf("%d %d\n", p, q);
        } else {
```

```
            printf("Not otter's type!\n");
        }
    }
    return 0;
}
```

F 扫雷 2024

难度	考点
4	二维数组、流程控制

题目分析

看完这道题，我们可以想到的一个方法是可以用一个二维数组 vis 标记哪些位置有雷，如果读入了一个位置 (i, j) 有雷，那我们就把 `vis[i][j]` 标记为 1。最后再遍历这个二维数组，如果某一位是 1 输出 `*`，如果某一位不是 1 统计周围八个方向有多少个 1 后输出，对周围八个方向进行遍历时我们可以利用两个数组来表示方向进行遍历；也可以用一个二重循环枚举横纵坐标进行遍历。

在统计周围八个格子有多少个 1 时我们要处理边界情况，实际上这也有一种常用的小技巧，就是不特殊判定周围的格子是否会越界，而是想办法让它即使越界了也不会影响最后的结果。比如我们可以假想地图周围有一圈宽度为 1 的围墙，这圈围墙是没有雷的，所以即使我们不特判位于边界处的格子的周围八个方向是否越界，也不会对结果造成影响。

示例代码

```
#include <stdio.h>

// 用来存储哪些格子有雷
int vis[55][55];
// 8个方向
int dx[8] = {-1, -1, -1, 0, 1, 1, 1, 0};
int dy[8] = {-1, 0, 1, 1, 1, 0, -1, -1};

int main() {
    int n, m, k;
    scanf("%d%d%d", &n, &m, &k);

    // 标记雷的位置
    for (int i = 0; i < k; i++) {
        int x, y;
        scanf("%d %d", &x, &y);
        vis[x][y] = 1;  // 1 表示该位置有雷
    }

    // 计算每个格子周围雷的数量并输出
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (vis[i][j] == 1)
                printf("*");
            else {
```

```
                int count = 0;
                // 遍历周围的8个方向
                for (int d = 0; d < 8; d++) {
                    int ni = i + dx[d], nj = j + dy[d];
                    count += vis[ni][nj];
                }
                printf("%d", count);
            }
        }
        printf("\n");
    }

    return 0;
}
```

G 排行榜整理

难度	考点
4	快速排序

题目分析

首先使用qsort进行快速排序，以过题数为第一关键字，罚时为第二关键字进行排序。

这里我们要对排名进行处理，如果过题数和罚时都完全相同，则两个人的排名相同，那这样下个人的排名和上一个保持一致，就不需要改变，如果不同，那么排名就是位于数列中的位置

示例代码

```
#include <stdio.h>
#include <stdlib.h>
int score[500005][2];
int cmp(const void *p1,const void *p2){
    int *a1=(int*) p1;
    int *a2=(int*) p2;
    if(a1[0]==a2[0]){
        return a1[1]>a2[1]?1:-1;
    }
    return a1[0]<a2[0]?1:-1;
}
int main(){
    int n,i,rank;
    scanf("%d",&n);
    for(i=1;i<=n;i++){
       scanf("%d%d",&score[i][0],&score[i][1]);
    }
    qsort(score+1,n,2*sizeof(int),cmp);
    for(i=1;i<=n;i++){
        if(!(score[i][0]==score[i-1][0]&&score[i][1]==score[i-1][1])){
            rank=i;
        }
```

```
            printf("%d: %d %d\n",rank,score[i][0],score[i][1]);
    }
    return 0;
}
```

H Orch1d 的全排列

难度	考点
5	字符串、字典序、递归、全排列

题目分析

由题意可知，我们要输出所有排列方式中字典序最大的结果，因此不难想到解法分为**全排列**以及**字符串比较**两部分。首先我们使用递归找出所有可能的单词排列顺序，再将每种情况拼接出的字符串按字典序比较大小，最后输出字典序最大的字符串即可。

示例代码

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<string.h>
#define ll long long
int sum;//单词个数
char word[10][10];//存单词
char ans[100];//字典序最大的字符串
int jud[10],order[10];//全排列
void getAnser(int now)
{
    if(now==sum)
    {
        char mont[100]="";
        for(int i=0;i<sum;i++)
        {
            strcat(mont,word[order[i]]);//将按当前排列顺序的单词拼接起来
        }
        if(strcmp(mont,ans)>0) strcpy(ans,mont);//如果字典序比当前答案大则更新
        return;
    }
        //生成全排列
    for(int i=0;i<sum;i++)
    {
        if(!jud[i])
        {
            jud[i]=1;
            order[now]=i;
            getAnser(now+1);
            jud[i]=0;
        }
    }
}
```

```
int main()
{
    while(scanf("%s",word[sum])!=EOF) sum++;//输入
    getAnser(0);
    printf("%s",ans);
    return 0;
}
```

本题的数据范围较小，当输入单词的数目较多时，用全排列找到所有排列顺序的方法显然会时间超限，因此可以考虑一种贪心的做法：对于任意两个字符串 S_1 和 S_2，如果 S_1S_2 的字典序比 S_2S_1 大，说明 S_1 比 S_2 更适合排在序列的前面，因此只需要对输入的单词按上述条件进行 `qsort` 排序，最终按顺序输出排序后的单词即为答案。

示例代码

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
char str[10][10], c;
int cnt;
int cmp(const void *a, const void *b) {
    char str1[20] = "", str2[20] = "";
    strcat(str1, (char *) a);
    strcat(str1, (char *) b);
    strcat(str2, (char *) b);
    strcat(str2, (char *) a);
    return strcmp(str2, str1);
}
int main() {
    while (scanf("%s", str[cnt++]) != EOF);
    qsort(str, cnt + 1, sizeof(str[0]), cmp);
    for (int i = 0; i < cnt + 1; i++)
        printf("%s", str[i]);
}
```

I 小水獭与大作业

难度	考点
6	malloc、字符串

题目分析

我们可以想到，我们可以开两个数组，第一个数组用来存所有的房间名称，第二个数组用来存每个房间的内容，两个数组之间用下标来对应。

对于存内容的数组，如果直接开二维数组的话，第二个维度太大许多空间就被浪费了（对于这道题会 MLE）；第二维开的太小的话又存不下最长的字符串，所以我们想到用 malloc 为每个房间的内容动态分配空间。我们要开一个数组存房间的内容，换句话说，数组中的每个元素是一个长度不定的字符数组，所以想到了开一个指针数组。每次追加内容时，显然我们要新开空间存储新的内容，但是由于 malloc 开

辟的空间两次之间不一定连续，所以我们再新开空间的时候，要用新开的空间存储原来的内容和追加的内容，并在把原来的内容存到新空间之后，把老的空间释放掉。

关于一些细节问题比如申请的空间要比 len 多一位，用来存字符串最后的空字符；还比如空格（用 gets 读入）和换行（第一行后面的）的处理。

示例代码

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

char names[1005][15];
char newName[15];
char *context[1005];
char newContent[10005];

int main() {
    int t;
    scanf("%d", &t);
    // 处理换行符
    getchar();
    int cnt = 0;
    while(t--) {
        gets(newName);
        int pos = 0;
        // 看看房间是否存在
        for(int i = 1; i <= cnt ;i++) {
            if(strcmp(names[i], newName) == 0)  {
                pos = i;
                break;
            }
        }

        // 房间不存在新开房间
        if(pos == 0) {
            pos = ++cnt;
            strcpy(names[pos], newName);
        }

        gets(newContent);
        // 注意新开的房间对应的内容数组里现在是一个空指针
        if(context[pos] == NULL) {
            // 开合适的空间
            context[pos] = (char *)malloc(strlen(newContent) + 1);
            memset(context[pos], 0, strlen(newContent) + 1);
            strcpy(context[pos], newContent);
        }
        else {
            char *tem = context[pos];
            // 为老内容和新内容开空间
            context[pos] = (char *)malloc(strlen(tem) + strlen(newContent) + 1);
            // 复制过去
            strcpy(context[pos], tem);
            strcat(context[pos], newContent);
```

```
            // 归还旧的空间
            free(tem);
        }
    }

    for(int i = 1; i <= cnt ;i++) {
        printf("%s:\n", names[i]);
        printf("%s\n", context[i]);
    }

    return 0;
}
```

J 小僵尸吃脑子 2 （简单版）

难度	考点
6	dfs 、剪枝

题目分析

很容易想到深度优先搜索算法：从起点开始尝试向四周游走，直到搜过全图（注意第一次到达终点时不一定是最优解），但是显然这个数据范围是会超时的。于是我们可以想到，只有这条路径通向某个坐标时，到达该坐标的时间比以前到达的时间更短，才选择走这条路径（即向这个点搜索）。

设全图中最慢的能抵达的点的耗时为 T_{max} ，从而剪枝后的时间复杂度不会超过 $O\left(nmT_{max}\right)$ 。另外，由于访问所有坐标的总次数还会随着地雷的增加而显著下降，因此即使地雷的排列有序导致 T_{max} 很大，程序也不会运行超时。因此这种方法是可以接受的。

下学期学过数据结构课程后，同学们可以掌握更快解决本题的方法，例如：

1. 堆优化的广度优先搜索算法（类似于 $Dijkstra$ 算法）；
2. $0-1BFS$ 算法（维护两个队列进行广度优先搜索）

这两个方法都不是本课程要求的算法，因此暂不要求同学们掌握。

示例代码

```
#include <stdio.h>
char r[110][110];
int T[110][110];
int n, m, t;
void dfs(int x, int y, int tm)
{
    T[x][y] = tm;
    if(x > 1 && r[x - 1][y] != '^' && T[x - 1][y] > (tm + 1 + t * (r[x - 1][y] ==
'#')))
        dfs(x - 1, y, (tm + 1 + t * (r[x - 1][y] == '#')));
    if(y > 1 && r[x][y - 1] != '^' && T[x][y - 1] > (tm + 1 + t * (r[x][y - 1] ==
'#')))
        dfs(x, y - 1, (tm + 1 + t * (r[x][y - 1] == '#')));
```

```
    if(x < n && r[x + 1][y] != '^' && T[x + 1][y] > (tm + 1 + t * (r[x + 1][y] ==
'#')))
        dfs(x + 1, y, (tm + 1 + t * (r[x + 1][y] == '#')));
    if(y < m && r[x][y + 1] != '^' && T[x][y + 1] > (tm + 1 + t * (r[x][y + 1] ==
'#')))
        dfs(x, y + 1, (tm + 1 + t * (r[x][y + 1] == '#')));
}
int main()
{
    scanf("%d%d%d", &n, &m, &t);
    for(int i = 1;i <= n;i++)
    {
        getchar();
        for(int j = 1;j <= m;j++)
        {
            T[i][j] = 110101;
            r[i][j] = getchar();
        }
    }
    dfs(1, 1, 0);
    for(int i = 1;i <= n;i++)
        for(int j = 1;j <= m;j++)
            if(r[i][j] == '*')
                printf("%d", T[i][j]);
    return 0;
}
```

K ddz 种树

难度	考点
7	二分答案

题目分析

我们尝试二分答案 s，考虑如何检查一个 s 是否可行。

对于一个答案 s，记第 i 棵树需要浇 c_i 次水高度才能不低于 s，显然 $c_i = \lceil \frac{s}{a_i} \rceil$，我们可以在优先满足最西边的树浇水次数的情况下逐步将机器人往东移动，最后判断总移动次数是否小于 x 即可。详见示例代码注释。

示例代码一

```
#include <stdio.h>
#define ll long long

ll a[100005], n, x;
ll c[100005];

int check(ll s) {
    for (ll i = 0; i < n; i++) {
        c[i] = (s + a[i] - 1) / a[i];// 计算 s / a[i] 向上取整
```

```
    }
    // 为满足这个答案 s 需要操作的次数
    ll cnt = 0;
    for (ll i = 0; i < n; i++) {
        // 每次循环前机器人在 i - 1 的位置
        if (c[i] > 0) {// 第 i 棵树还需要浇水
            // 先往东移动一次到 i ，然后东西交替 c[i] - 1 次
            cnt += 1 + 2 * (c[i] - 1);
            // 同时第 i + 1 棵树会被浇 c[i] - 1 次
            c[i + 1] -= (c[i] - 1);
        } else if (i < n - 1) {// 第 i 棵树不需要浇水
            cnt++;// 直接往东移动到 i ，如果是最后一棵树就不需要移动了
        }
    }
    // 只有需要操作的次数小于等于 x 才行
    return cnt <= x;
}

void work() {
    scanf("%lld %lld", &n, &x);
    for (ll i = 0; i < n; ++i) {
        scanf("%lld", &a[i]);
    }
    // 二分答案
    ll l = 0, r = 1000000000000000000ll;
    while (l < r) {
        ll mid = (l + r + 1) >> 1;
        if (check(mid)) l = mid;
        else r = mid - 1;
    }
    printf("%lld\n", l);
}

int main() {
    ll t; scanf("%lld", &t); while(t--) work();
    return 0;
}
```