C7 - Solution

A Otter 256

难度	考点
1	字符串

题目分析

本题为输入一个长度八位的字符串，然后判断字符串的格式是否满足要求。

输入完一个字符串后，最简单的办法我们可以先检查前五位是否依次为 O、T、T、E、R（也可以利用 strstr 函数或者 strncmp 函数直接看前五位是不是 `OTTER`），然后把后三位提取为一个整数看看是否在合法范围内。

需要注意的是在判断后三位是否合法的时候有同学使用了这样的代码 `strcmp(str + 5, "000") >= 0 && strcmp(str + 5, "256") <= 0`，要注意的是在字典序意义下比较是不一定正确的，还要判断后三位是不是都是数字（一个不正确的例子如 `OTTER00a`）。

示例代码

```
#include <stdio.h>
#include <ctype.h>

char str[10];

int main() {
    while(scanf("%s", str) != EOF) {
        // 判断前五位也可以逐位判断
        // 也可以 strstr(str, "OTTER") == str
        if(strncmp(str, "OTTER", 5) == 0 && isdigit(str[5]) && isdigit(str[6]) &&
isdigit(str[7])) {
            int num = (str[5] - '0') * 100 + (str[6] - '0') * 10 + str[7] - '0';
            if(num >= 0 && num <= 256)
                printf("Valid\n");
            else
                printf("Invalid\n");
        } else
            printf("Invalid\n");
    }
}
```

B 爱吃糖葫芦

难度	考点
2	字符串拼接 `strcat`

题目分析

本题输入为三个字符串，输出为字符串拼接后的后半部分。需要注意的有两点：

1. 用于存储拼接三个字符串的`char`类型数组要足够大，在本题应不小于$3 \times 1000 = 3000$
2. `ans`字符串的后半部分的范围应当是$[\frac{|ans|}{2}, |ans|)$

示例代码

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

char a[1009], b[1009], c[1009];
char ans[3009];

int main() {
    scanf("%s%s%s", a, b, c);
    strcat(ans, c);
    strcat(ans, b);
    strcat(ans, a);
    for(int i=strlen(ans)/2;i<strlen(ans);i++){
        printf("%c",ans[i]);
    }
    //printf("%s", ans + strlen(ans) / 2);
    return 0;
}
```

C 林士谔算法 信类2024Ver

难度	考点
2	函数传参，指针类型参数

题目分析

本题相当于代码填空题，利用给定的函数实现功能，主要考察数组类型参数和指针类型参数的传参。

数组传参时，在函数中的形参如这种形式：`int f(int a[])`，调用函数传参时如这种形式：`int a[100];f(a);`

指针传参时，在函数中的形参如这种形式：`int f(int *a)`，调用函数传参时如这种形式：`int a;f(&a);`（一定要注意`*`在指针相关部分有两种不同的含义，一个是声明一个指针类型，一个是对指针类型使用解引用）

接下来每一行逐行翻译即可，最关键的地方就是调用函数那一行。

示例代码

```
#include<stdio.h>
#include<string.h>
double a[20];
```

```c
void Shie(int n, double a[], double *p, double *q);

int main()
{
    /*------下面根据指引写你自己的代码------*/
    //定义 int 型变量 n，double 型变量 p 和 q
    int n;
    double p, q;

    //读取n
    scanf("%d", &n);

    //从第n项开始，一直到第0项，读取全局数组a
    for(int i = n; i >= 0 ;i--)
        scanf("%lf", &a[i]);

    //调用Shie函数
    Shie(n, a, &p, &q);

    //输出p q
    printf("%.6lf %.6lf", p, q);

    return 0;
}

/*--下面是Shie函数，只要调用即可，无需理解原理--*/
void Shie(int n, double a[], double *p, double *q)
{
    double eps = 1e-12;
    double b[20];
    double c[20];
    // 数组 b 是多项式 a 除以当前迭代二次三项式的商
    memset(b, 0, sizeof(b));
    // 数组 c 是多项式 b 乘以 x 平方再除以当前迭代二次三项式的商
    memset(c, 0, sizeof(c));
    *p = 0;
    *q = 0;
    double dp = 1;
    double dq = 1;
    while (dp > eps || dp < -eps || dq > eps || dq < -eps)  // eps 自行设定
    {
    double p0 = *p;
    double q0 = *q;
    b[n - 2] = a[n];
    c[n - 2] = b[n - 2];
    b[n - 3] = a[n - 1] - p0 * b[n - 2];
    c[n - 3] = b[n - 3] - p0 * b[n - 2];
    int j;
    for (j = n - 4; j >= 0; j--) {
            b[j] = a[j + 2] - p0 * b[j + 1] - q0 * b[j + 2];
        c[j] = b[j] - p0 * c[j + 1] - q0 * c[j + 2];
    }
    double r = a[1] - p0 * b[0] - q0 * b[1];
    double s = a[0] - q0 * b[0];
    double rp = c[1];
    double sp = b[0] - q0 * c[2];
```

```
    double rq = c[0];
    double sq = -q0 * c[1];
    dp = (rp * s - r * sp) / (rp * sq - rq * sp);
    dq = (r * sq - rq * s) / (rp * sq - rq * sp);
    *p += dp;
    *q += dq;
    }
    return;
}
```

D 魔法师水獭与标准咒语指南

难度	考点
3	strstr、字符串、数组与指针

题目分析

本题主要考察库函数 `strstr` 的使用，并借由该函数引导同学们掌握 `<string.h>` 中一类库函数的使用。

在 `strstr(text, keyword)` 中，我们要传两个参数，一个是原字符串 $text$，一个是要查找的子串 $keyword$，根据 PPT 上的例子，传进去两个数组名（数组名相当于数组的起始地址，利用 `char` 数组的起始地址我们就能确定一个字符串）。`strstr(text, keyword)` 返回值是 $keyword$ 第一次在 $text$ 中出现的地址（没找到返回空指针 `NULL`），返回值类型是 `char *`。因为一个数组在内存中是连续存储的，而数组首地址指向的是数组中下标为 `0` 的元素的位置，指向同一数组内的元素的指针作差后，得到的结果就是两个位置的下标差。所以将前面查到的目标子串第一次出现位置的首地址减去 $text$ 数组首地址，就得到了目标子串首字母在 $text$ 数组中的位置下标。

最后，如果不处理整数后面的换行符的话，根据 `gets` 会一直读入，直到遇到换行符停止，把这个换行符一起读进来后再把它变成空字符，在读入完 n 之后如果直接使用 `gets` 读检索关键词，那么其实只会把这个换行符读走，导致字符串其实是空串。

示例代码

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

char str[1005];
char tem[1005];

int main() {
    gets(str);
    int n;
    scanf("%d", &n);
    getchar();
    for(int i = 1; i <= n; i++) {
        gets(tem);
        char *r = strstr(str, tem);
        if(r == NULL)
            printf("Spell Not Found!\n");
```

```
        else
            printf("%d\n", r - str);
    }
    return 0;
}
```

E 摩卡与指针

难度	考点
3	输入输出、指针与数组

题目分析

数组名相当于数组的起始地址，假设数组中每个元素的数据类型为 `type`，根据 `type` 我们就能知道每个类型元素的大小 `size`（建议使用诸如 `sizeof(type)` 的方式来获取该类型元素的大小，因为在不同平台上同样数据类型的大小可能并不一致，使得写出来的程序在本地和在 OJ 上结果不同）。那么第 i 个元素的起始地址就能利用公式 $base + (i \times size)$ 来计算，其中 $base$ 是数组的起始地址。

在这道题中，读入 `type` 后，可以用 `strcmp` 来确定 `type` 具体是什么，还可以只通过第一个字符来判断 `type`。

第二就是假如我声明了一个数组 `int a[100];`，实际上合法的下标范围只有 $[0, 99]$，并没有 100，尤其是在利用字符数组读入字符串的地方，一定要注意这一点。

示例代码

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

char str[15];

int main() {
    gets(str);
    int sz = 0;
    switch (str[0])
    {
    case 'i':
        // 这道题规定了每种类型的大小，所以我们也可以不使用 sizeof(int) 这样的方式
        sz = 4;
        break;

    case 'c':
        sz = 1;
        break;

    case 'f':
        sz = 4;
        break;

    case 'd':
        sz = 8;
```

```
        break;

    case 'l':
        sz = 8;
        break;
    }
    int n, q, pos;
    scanf("%d%d", &n, &q);
    for(int i = 1; i <= q ;i++) {
        scanf("%d", &pos);
        if(pos >= n)
            printf("Careless Otter!\n");
        else
            printf("0x%08x\n", pos * sz);
    }
    return 0;
}
```

F ddz 点亮矿洞

难度	考点
4	循环、模拟

题目分析

根据题意简单模拟，开一个数组存放每个格子的亮度，对于每个铜灯更新它能照到的范围的亮度，最后遍历数组计算亮度小于 7 的格子个数即可。（详见示例代码一）

当然，你可以忽略亮度到底是多少，只记录是否大于 7 。详见（示例代码二）

示例代码一

```
#include <stdio.h>
#define max(a, b) ((a) > (b) ? (a) : (b))
#define ll long long

ll light[505005], L[4] = {15, 12, 8, 4};

void work() {
    ll n, k, a, b;
    scanf("%lld %lld", &n, &k);
    for (ll i = 1000; i <= n + 999; ++i) {
        light[i] = 0;
    }
    while (k--) {
        scanf("%lld %lld", &a, &b);
        ll l = L[b], i = 0;
        a += 999;
        do {
            light[a + i] = max(light[a + i], l);
            //此处可能越界，所以将所有下标加999
```

```
            light[a - i] = max(light[a - i], l);
            i++;
        } while (--l);
    }
    ll cnt = 0;
    for (ll i = 1000; i <= n + 999; ++i) {
        if (light[i] <= 7) {
            cnt++;
        }
    }
    printf("%lld\n", cnt);
}

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

示例代码二

```
#include <stdio.h>
#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) < (b) ? (a) : (b))
#define ll long long

ll light[505005], L[4] = {15, 12, 8, 4};

void work() {
    ll n, k, a, b;
    scanf("%lld %lld", &n, &k);
    for (ll i = 1; i <= n; ++i) {
        light[i] = 0;
    }
    ll cnt = n;
    while (k--) {
        scanf("%lld %lld", &a, &b);
        //遍历这个铜灯能让哪些格子不刷怪
        for (ll i = max(1, a - L[b] + 8); i <= min(n, a + L[b] - 8); ++i) {
            if (!light[i]) {
                light[i] = 1;
                cnt--;
            }
        }
    }
    printf("%lld\n", cnt);
}

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

G 盟主的挑战--逆序真言

难度	考点
5	字符串

题目分析

在各个情况下的处理方式，在题目描述中表述的较为明确。

需要注意：

1. 如果口诀包含秘籍名称，那么相较于口诀，秘籍名称位置和顺序不能发生变化
2. 若逆置结果字典序 **不大于** 原始顺序字典序，那么 **每个字母** 都要颠倒

> **尽信书不如无书**，了解一个函数的实现是为了更好的为程序服务，而不应该局限于函数本身

示例代码

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

char g[1005], s[10005], rev[10005];
int gLen, sLen, k;
int low, hi, gst, ged;

char *str_rev(char *str, int offset) {
    low = offset, hi = offset + k - 1;//当前口诀的起止点
    while (s[hi] == '\0') hi--;
    if (low == gst) low += gLen;
    if (hi == ged) hi -= gLen;
    for (; low < hi;) {
        char tmp = str[low - offset];
        str[low - offset] = str[hi - offset];
        str[hi - offset] = tmp;
        low++, hi--;
        if (low == gst) low += gLen;
        if (hi == ged) hi -= gLen;
    }
}

void solve(int offset) {
    char *p = strstr(s + offset, g);
    gst = -1, ged = -1;
    if (p != NULL) {//如果有名称字串
        gst = p - s;//名称字串的起点
        ged = gst + gLen - 1;//名称字串的终点
    }
    strcpy(rev, s + offset);//复制到rev字符数组
    str_rev(rev, offset);//根据规则逆置口诀
    if (strcmp(rev, s + offset) > 0) {
        strcpy(s + offset, rev);
    } else {
        for (int i = offset; s[i]; i++) {
```

```
            s[i] = 'a' + 'z' - s[i];//颠倒原数组
        }
    }
}

int main(void) {
    scanf("%d", &k);
    scanf("%s%s", g, s);
    gLen = strlen(g);
    sLen = strlen(s);
    for (int i = 0; i < sLen; i += k) {
        char tmp = s[i + k];
        s[i + k] = '\0';
        solve(i);
        s[i + k] = tmp;
    }
    printf("%s", s);
    return 0;
}
```

H 盟主的挑战--不战而胜

难度	考点
5	排序、贪心

题目分析

在武林争霸赛中，每个擂主都有两条可能的认输路径：

1. 直接被盟主挑战而认输；
2. 受到盟主威慑力的影响而认输。

深入分析这一情况，我们可以发现擂主之间的威慑传递具有对称性：如果擂主b先认输，擂主a能够感受到来自b的威慑力；反之，如果擂主a先认输，擂主b同样能感受来自a的威慑力。

这意味着我们的分析可以有序地从左至右或从右至左进行，关键在于保持顺序的一致性。

1. 对于任何一个擂主，如果它是序列中的第一个，或者与前一个擂主的距离超过了威慑力m的范围，那么盟主必须亲自发起挑战。
2. 在其他情况下，擂主会因为前一个擂主认输而感受到威慑力，从而无需盟主挑战。

示例代码

```
#include <stdio.h>

int T, n, d, ans;
int a[10009];

void swap(int tar[], int i, int j) {
    int tmp = tar[i];
    tar[i] = tar[j];
```

```
    tar[j] = tmp;
}

void bubble(int tar[], int num) {
    for (int i = 1; i <= num; i++) {
        for (int j = i; j <= num; j++) {
            if (tar[i] > tar[j]) {
                swap(tar, i, j);
            }
        }
    }
}

int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &d);
        ans = 0, a[0] = -d - 1;
        for (int i = 1; i <= n; i++)
            scanf("%d", &a[i]);
        bubble(a, n);
        for (int i = 1; i <= n; i++) {
            if (a[i - 1] + d < a[i]) {
                ans++;
            }
        }
        printf("%d\n", ans);
    }
}
```

I 摩卡与水獭排队 3

难度	考点
5	贪心，二分

题目分析

在 `摩卡与水獭排队 2` 中，由于输入有序，所以我们直接简单地使用一个二分即可。在这道题中，唯一的区别就是输入的水獭身高不再有序了。

但显然，对于 $i < j$，如果 $a_i \geq a_j$，那么答案必然不是 j，此时，无论 a_j 是多少，只要在 $[1, a_i]$ 范围内，都不会影响答案，都与原问题等价。所以当 $i < j$ 且 $a_i \geq a_j$ 时，都可以把 a_j 修改为与 a_i 相同的值而不会影响答案。 从而将原数组转化为一个查询等价的非递减数组，转化为 `摩卡与水獭排队 2`，继续用二分法解决这道题。

在 std 上，该题代码仅仅比 `摩卡与水獭排队 2` 多了两行。

示例代码

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

```c
int binary_search(int arr[], int n, int h) {
    int left = 0, right = n - 1, result = -1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (arr[mid] >= h) {
            result = mid;
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }

    // 如果 result != -1，说明找到了，因为编号从 1 开始，我们要给 result + 1
    return result == -1 ? -1 : result + 1;
}

int heights[500005];

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

    // 读入身高
    for (int i = 0; i < n; i++)
        scanf("%d", &heights[i]);

    for(int i = 1; i < n ;i++) {
        if(heights[i] < heights[i - 1])
            heights[i] = heights[i - 1];
    }

    while (m--) {
        int h;
        scanf("%d", &h);
        // 进行查找
        printf("%d\n", binary_search(heights, n, h));
    }

    return 0;
}
```

J Baymax的困难字符串

难度	考点
6	dfs

题目分析

本题的基本框架不难确定：从左到右依次考虑每个位置上的字符。问题的关键在于如何判断当前字符串是否已经存在连续的重复子串。一种方法是检查所有长度为偶数的子串，分别判断每个子串的前一半是否等于后一半。尽管这种方法是正确的，但是做了很多无用功，因为大量重复子串在前一次递归时已被检查。因此，我们只需要判断每一次递归时新增在末尾的字符是否产生重复子串即可，示例代码如下。

示例代码

```
#include <stdio.h>
int n, L, A[10005], cnt = 0;
int dfs(int cur)
{
    if (cnt++ == n) //dfs进行cnt次，相当于找字典序第cnt小的字符串
    {
        for (int i = 0; i < cur; i++)
            printf("%c", A[i] + 'A');
        return 0;
    }
    else
    {
        for (int i = 0; i < L; i++)
        {
            A[cur] = i;
            int ok = 1;
            for (int j = 1; 2 * j <= cur + 1; j++) //检查长度为 j*2 的后缀
            {
                int equal = 1;
                for (int k = 0; k < j; k++) //检查前一半是否等于后一半
                {
                    if (A[cur - k] != A[cur - k - j])
                    {
                        equal = 0;
                        break;
                    }
                }
                if (equal) //后一半等于前一半，不合法
                {
                    ok = 0;
                    break;
                }
            }
            if (ok) //方案合法，递归寻找下一个字符
                if (!dfs(cur + 1))
                    return 0;
        }
    }
    return 1;
}
int main()
{
    scanf("%d%d", &n, &L);
    dfs(0);
}
```

实际上，当 $L \geq 3$ 时就很少回溯了，此时可以构造出无限长的字符串，不存在相邻的重复子串。