上机5解析

A ZX的生日

难度	考点
2	字符串 数组

题目分析

本题来源于PPT中的例题，比较简单，可以练习使用二维数组来保存星期的字符串，如果用m来表示下月k日的星期，那么$m=(n-x+y+k)\%7$。

示例代码

```
#include <stdio.h>
char day_name[][12] ={
        "Sunday",
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday",
        "Saturday"};
int main(void)
{
    int n, x, y, k, m;
    while (scanf("%d%d%d%d", &n, &x, &y, &k) > 0)
    {
        m = (n - x + y + k) % 7;
        printf("%s\n", day_name[m]);
    }
    return 0;
}
```

B学生会选举

难度	考点
1	排序

题目分析

运用课上学的的冒泡排序，选择排序等即可。

示例代码

```c
#include <stdio.h>
#define N 1000
void bubbleSort(int[], int); //冒泡排序
int num[N + 10];             //存放选票编号的数组
int main()
{
    int n, i = 0; //n:选票的个数
    while (~scanf("%d", &num[i]))
        i++;
    n = i; //不定个数输入 需要求出选票的个数n
    bubbleSort(num, n);
    for (i = 0; i < n; i++)
        printf("%d ", num[i]);
    return 0;
}
void bubbleSort(int a[], int n)
{
    int i, j, hold, flag;
    for (i = 0; i < n - 1; i++)
    {
        flag = 0;
        for (j = 0; j < n - 1 - i; j++)
        {
            if (a[j] > a[j + 1])
            {
                hold = a[j];
                a[j] = a[j + 1];
                a[j + 1] = hold;
                flag = 1;
            }
        }
        if (0 == flag)
            break;
    }
}
```

C统计上机成绩

难度	考点
2	二维数组

题目分析

本题考察二维数组的应用，需要注意双层循环中遍历的顺序，理清行下标和列下标的含义。

示例代码

```
#include <stdio.h>
int main()
{
    int a[100][6];
    int m, n;
    int i, j;
    int sum, count;
    scanf("%d%d", &m, &n);
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &a[i][j]);
    //计算每次上机的平均分
    for (j = 0; j < n; j++)
    {
        sum = 0;
        for (i = 0; i < m; i++)
        {
            sum += a[i][j];
        }
        printf("%.2f ", (double)sum / m);
    }
    printf("\n");
    //计算每次上机的优秀率
    for (j = 0; j < n; j++)
    {
        count = 0;
        for (i = 0; i < m; i++)
        {
            if (a[i][j] >= 80)
                count++;
        }
        printf("%.2f%% ", count * 100.0 / m);
    }
    printf("\n");
    //计算每个学生的平均分
    for (i = 0; i < m; i++)
    {
        sum = 0;
```

```
        for (j = 0; j < n; j++)
        {
            sum += a[i][j];
        }
        printf("%.2f\n", (double)sum / n);
    }
    return 0;
}
```

D最长行

难度	考点
3	字符串

题目分析

`scanf()` 函数遇到空白符就结束该字符串的读取，因此如果遇到中间有空格的句子，比如样例，就会造成只读进一个词的情况。`gets()` 和 `fgets()` 函数都是一次读入一行，但是问题在于会把行尾的空白符也全部读入，因此需要手动去除空白符。但是要注意的是 `gets()` 对读入数据长度不作限制，如果不知道读入数据最长有多长，就容易导致危险的错误。而 `fgets()` 本身就对读入数据的长度做了限制，应用时也较为安全。

示例代码

```
#include <stdio.h>
#include <string.h>
char str[20001];
char in[20001];
int isSpace(char c); //isSpace()函数用于判断是否为空白符
int main()
{
    int len, max = 0;
    while (fgets(in, 20001, stdin) != NULL) //本题为什么要用fgets()而不用scanf()?
    {
        len = strlen(in);
        for (; len > 0 && isSpace(in[len - 1]); len--) //小心一行全是空白符的情况
        {
            in[len - 1] = 0;
        }
        if (len >= max)
        {
            strcpy(str, in);
            max = len;
        }
    }
```

```
    printf("%d %s#END", max, str);
    return 0;
}
int isSpace(char c)
{
    return (c == ' ') || (c == '\n') || (c == '\r') || (c == '\t');
}
```

E 求阶乘·改

难度	考点
4	数组，性能分析

题目分析

本题需要考虑性能。如果 10^5 下的询问能够事先全部处理好，则可以每次都很快地回答询问。
注意到每次询问的数 n 不会超过 10^6，因此可以先把 n 在 10^6 以内的答案先预处理出来，在之后的询问中无需运算就能得到结果。我们可以预处理一个数组 $sum = \sum_{i=0}^{n} i!$，并且有关系式 $sum_i = i \times sum_{i-1}$。
另外，本题中会出现大 `int` 相乘的情况，需要采用 `long long` 完成程序编写。

示例代码

```
#include <stdio.h>
#include <stdlib.h>
// 合理使用宏定义能够方便代码编写
#define ll long long
#define N ((int)1e6 + 5)
#define MOD (1000003)
ll fac[N], sum[N];
int main()
{
    int q, n, i;
    fac[0] = sum[0] = 1;
    // 递推计算 i!
    for (i = 1; i < N; i++)
        fac[i] = fac[i - 1] * i % MOD;
    // 递推计算 sum_i
    for (i = 1; i < N; i++)
        sum[i] = (sum[i - 1] + fac[i]) % MOD;
    scanf("%d", &q);
    while (q--)
    {
        scanf("%d", &n);
        printf("%lld\n", sum[n]);
```

```
    }
    return 0;
}
```

F人工执杖

难度	考点
3	二维数组

题目分析

只需要按照题目要求读入两个二维数组并进行处理即可。

在本题中，我坐标系的定义与工程上的惯例相同，即图形的向右为x正方向，向上为y正方向，以图片左上角为原点。

示例代码

```
#include <stdio.h>

int img[725][965] = {0};
int out[725][965] = {0};
int core[20][20] = {0};
int w, h, m;

void conv()
{
    int i, j, x, y;
    for (i = 0; i <= h - m; ++i)
    {
        for (j = 0; j <= w - m; ++j)
        {
            for (x = 0; x < m; ++x)
            {
                for (y = 0; y < m; ++y)
                {
                    out[i][j] += img[i + y][j + x] * core[y][x];
                }
            }
        }
    }
}

int main()
{
    int i, j;
```

```
    // 输入
    scanf("%d %d %d", &h, &w, &m);
    for (i = 0; i < h; i++)
    {
        for (j = 0; j < w; j++)
        {
            scanf("%d", &img[i][j]);
        }
    }
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < m; j++)
        {
            scanf("%d", &core[i][j]);
        }
    }

    conv(); // 计算

    // 输出
    for (i = 0; i <= h - m; i++)
    {
        for (j = 0; j <= w - m; j++)
        {
            printf("%d ", out[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

G爱的魔力转圈圈

难度	考点
3	二维数组、模拟

题目分析

这道题其实有两种做法：一种可以寻找每次前进格数的规律，不过比较复杂，我没做出来，非常不好意思。
第二种就是上面的做法，这种做法用到了一种概念，就是记录访问，在遍历二维数组的过程中，有两种坐标不能访问，一种是边界之外的，比如坐标小于0的，和大于等于n的；另一种是已经访问过的，这种我们在每个访问过的点上打一个标记，就是vis数组。我们把这两种情况的格子理解成一堵墙，顺时针访问的时候，我们从起点向右走，撞到墙就右转，直到无路可走。反之亦然。

示例代码

```
#include <stdio.h>
#include <string.h>
char a[150][150], laji[10];
int n, vis[150][150];
int islegal(int x, int y) //判断x,y点是否合法，或者是否被访问过
{
    if (x < 0 || x >= n)
        return 0;
    if (y < 0 || y >= n)
        return 0;
    if (vis[x][y])
        return 0;
    return 1;
}
int dx[4] = {0, 1, 0, -1}; //右下左上四种情况下，走一步x的变化
int dy[4] = {1, 0, -1, 0}; //右下左上四种情况下，走一步y的变化
int main()
{
    int i, j, k, x, y, now, nx, ny;
    scanf("%d", &n);
    gets(laji);
    for (i = 0; i < n; ++i)
    {
        for (j = 0; j < n; ++j)
            scanf("%c", &a[i][j]);
        gets(laji);
    }
    for (k = 1; k >= -1; k -= 2)
    {
        x = y = 0;
        now = (k == 1 ? 0 : 1);       //顺时针先向右，逆时针先向下
        memset(vis, 0, sizeof(vis)); //清空访问情况
        for (i = 0; i < n * n; ++i)
        {
            vis[x][y] = 1; //记录访问情况
            if (k == 1 && a[x][y] <= 'z' && a[x][y] >= 'a')
                printf("%c", a[x][y]); //顺时针输出小写
            if (k == -1 && a[x][y] <= 'Z' && a[x][y] >= 'A')
                printf("%c", a[x][y]); //逆时针输出大写
            nx = x + dx[now];
            ny = y + dy[now];
            if (!islegal(nx, ny))
                now = (now + k + 4) % 4; //预判下一步是否合法，不合法通过变now转向
            x = x + dx[now];
            y = y + dy[now]; //更新x,y
        }
        printf("\n");
```

```
    }
    return 0;
}
```

H魔镜

难度	考点
4	字符串

题目分析

根据题意，要将字符串中的每个单词进行反转，即翻转字符串的区间。翻转区间的基本操作和期中考试的F题(斯波利特平衡术)类似，重要的是在这道题中如何划分区间。
首先对于反转字符串中的一段区间，可以写一个函数reverse，给出待反转区间的左边界l和右边界r，利用循环将该区间翻转过来。在函数中，l和r充当了循环变量，用来控制交换的位置，其中循环头的作用是使得l和r始终位于区间的对称位置，而循环体则是标准的用来交换两个变量的代码，交换了位于l和r处的字符。
在程序的主体部分，输入字符串后即可循环遍历，找到字符串中所有连续的单词。对于一个单词我们需要标记出单词的头和尾，由于遍历一行字符串是顺序进行的，先经过头再经过尾，因此只需用一个额外的变量来记录单词头，随着遍历进行到一个单词尾，立即完成区间翻转即可。注意判断单词头和单词尾时要考虑位于整个字符串首或尾的情况。
另外要将所有的元音字母变为大写，辅音字母变为小写，可以写一个函数来判断一个字符是否为元音字母。转换大小写可以调用ctype.h中的toupper和tolower函数，也可以自己用ascii码来计算。

示例代码

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
char txt[120005];
int isvowel(char c)
{
    c = tolower(c);
    return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';
}
void reverse(int l, int r)
{
    char tmp;
    for (; l < r; l++, r--)
    {
        tmp = txt[l];
        txt[l] = txt[r];
        txt[r] = tmp;
    }
```

```
}
int main()
{
    int n, T, i, j, l, p;
    scanf("%d", &n);
    while (getchar() != '\n')
        ; // 过滤掉第一行输入的行尾换行符
    for (T = 1; T <= n; T++)
    {
        gets(txt);
        l = strlen(txt); // 获取字符串的长度
        p = 0;
        for (i = 0; i < l; i++)
        {
            if (isalpha(txt[i]))
            {
                txt[i] = isvowel(txt[i]) ? toupper(txt[i]) : tolower(txt[i]);
                // 转换大小写，这里用一个三目运算符来代替if判断
                if (i == 0 || !isalpha(txt[i - 1])) // 单词的开头
                    p = i;
                if (i == l - 1 || !isalpha(txt[i + 1])) // 单词的末尾
                    reverse(p, i);                      // 翻转单词
            }
        }
        printf("Case #%d: %s\n", T, txt);
    }
    return 0;
}
```

I 关于名字的愤怒

难度	考点
3	字符串输入输出，字符串处理

题目分析

整体的思路是这样的，从前到后遍历字符串s， 对于$s[i]$，如果它及后面的字母跟a匹配，则输出b，并直接跳到$s[i+strlen(a)]$重复操作；如果$s[i]$不匹配，则直接输出$s[i]$。
一些细节方面的处理是：对于有转义字符的，直接用"%s"是无法正常输出的，一种解决方案是可以逐个字符输出，即用"%c"。

示例代码

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

```c
#include <string.h>
#define N 205
char s[N], a[25], b[25];
int sl, al, bl;
int comp(int p) //看s[p]及之后的字符是否与a匹配
{
    if (p + al > sl)
        return 0;
    int i;
    for (i = 0; i < al; i++)
        if (s[p + i] != a[i])
            return 0; //只要有一个字符不匹配，即可判断为不匹配
    return 1;
}
void Init() //读入数据
{
    fgets(s, N - 1, stdin);
    fgets(a, 25, stdin);
    fgets(b, 25, stdin);
    sl = strlen(s);
    while (s[sl] != ';')
        sl--; //字符串的末尾可能会有多余的换行符，要清除掉；';'也要处理
    al = strlen(a);
    while (a[al] != ';')
        al--;
    bl = strlen(b);
    while (b[bl] != ';')
        bl--;
}
int main()
{
    Init();
    int i, j, flag = 1;
    for (i = 0; i < sl - al + 1; i++)
        if (comp(i)) //判断s是否包含子串a
        {
            flag = 0;
            break;
        }
    if (flag)
    {
        printf("No Replacement!\n");
        return 0;
    }
    for (i = 0; i < sl; i++)
        if (comp(i)) //判断能否成功匹配
        {
            for (j = 0; j < bl; j++)
                printf("%c", b[j]); //输出替换的字符串b
```

```
            i += al - 1;            //越过后面al个字符（因为已经跟a匹配了），注意for
循环自带i++，所以这里i+=(al-1)
        }
        else
        {
            printf("%c", s[i]); //不匹配则输出当前字符
        }
    printf(";\n");
    return 0;
}
```

J 红黑歌合战

难度	知识点
4	排序

说在前面

首先简单过一下时间和空间复杂度的基本概念，大家可能还不是很熟悉，在此作粗略了解。
可以大致认为，对于一般的C程序，计算机每1000ms，可以处理10^8左右的运算量，以此我们可以大致估计自己的代码能否通过题目的时限要求：假设题目中输入数据的规模为 n ，在你的程序中运用到了一个二重循环如下：

```
for(i = 0; i < n; i++)
for(j = 0; j < n; j++)
//do something...
```

那么你的这段程序的运算量便与n^2呈近似正比的关系。可能其中包含一些常数级别的运算，但一般情况下我们可以忽略这些常数运算，只关心与输入规模相关的复杂度。假设你的程序中其他的代码段时间复杂度相比上述的循环可以忽略，那么当n的输入规模在10^4以上时，你的程序将很难通过1000ms的时间限制。
再简单介绍一下空间复杂度，这里就只说说声明数组的问题，假设我们的程序使用了一个`int`型的数组`int a[MAXN];`，那么根据`int`型数据的字节长度，可以算出这个数组占据的内存空间为$\frac{4\cdot MAXN}{1024}$ kb。在做题时请留意自己的数组规模是否过大以至于超过题目的内存限制，另外较大的数组建议声明为全局数组。

题目分析

一句话题面：给一列数，求出重复次数最多的数中最大的那个数。
我们可以对读入的数字进行从小到大的排序，再依次遍历排好的数组，当前一个元素与当前元素相同时，计数器+1，若不小于最大值，便更新最大值和答案。注意，由于已经按照从小到大的次序排序，为了满足答案是更大的数，这里运用的是`>=`而非`>`，具体细节请大家认真思考。
当然，直接使用二重循环对每个数字记录最大值的暴力方法也可以，但是会比使用高效的排序算法的速度慢一些，不过对于红组的数据，都是可以通过的。

但对于黑组来说，情况有所不同，如果需要将黑组的数据（$n = 10^6$）全部用数组存储并操作，将会超过本题的内存限制（事实上在排序的速度上也会不足，但是鉴于评测机性能不太稳定这次就没认真卡时间复杂度），故不得不考虑其他算法。这里我们发现虽然黑组的数据量大，但是其数据均为正整数且数值很小（$b_i \leq 10^3$），于是一种最简单的散列算法（也可称之为桶排序），便呼之欲出了：将数值当做数组下标建立对应关系，数组的值代表了该数值出现的次数，那么我们只需要`10^3`级别的数组大小，读入数据后再遍历一遍数组即可得到答案，在时空上都有了很大的改进。

示例代码

```
#include <stdio.h>
#include <string.h>
int a[1005], b[1005];
int main()
{
    int m, n, i, j, x, tmp;
    while (~scanf("%d%d", &m, &n))
    {
        int max = 0;
        int ansa, ansb, maxa, maxb, cnt;
        memset(b, 0, sizeof(b)); //注意每组数据处理前，要将数组b置零
        for (i = 0; i < m; i++)
        {
            scanf("%d", &a[i]);
        }
        for (i = 0; i < n; i++)
        {
            scanf("%d", &x);
            max = max > x ? max : x; //这个max是为了记录黑组中最大的数，确定之后的遍历上限
            b[x]++;
        }
        //朴素的冒泡排序即可，当然同学们可以使用效率更高的排序方法
        for (i = 0; i < m - 1; i++)
        {
            for (j = 0; j < m - i - 1; j++)
            {
                if (a[j + 1] < a[j])
                {
                    tmp = a[j];
                    a[j] = a[j + 1];
                    a[j + 1] = tmp;
                }
            }
        }
        ansa = a[m - 1], maxa = 1, cnt = 1; //注意细节
        for (i = 1; i < m; i++)
        {
            if (a[i] == a[i - 1]) //出现a[i-1]、a[i+1]等访问时一定注意是否越界
```

```
            {
                cnt++;
                if (cnt >= maxa)
                {
                    maxa = cnt;
                    ansa = a[i];
                }
            }
            else
                cnt = 1;
        }
        ansb = 0, maxb = 0; //注意细节
        for (i = 1; i <= max; i++)
        {
            if (b[i] >= maxb)
            {
                maxb = b[i];
                ansb = i;
            }
        }
        printf("%d %d\n", ansa, ansb);
    }
    return 0;
}
```

K传快一点吧

难度	考点
2	循环结构

题目分析

这是一个简单模拟的数学题：
矩阵乘法具有结合律，所以向量a×矩阵A×矩阵B==向量a×(矩阵A×B)，其中A×B中的运算即矩阵乘法。
题目所求即n个矩阵的矩阵乘法乘积。
简单计算一下结果的理论最大值10000*10000*10*10000*10=1e14超出了int，但不超过long long，所以可以用long long存储和运算。
示例代码中，S12表示前两个矩阵的乘积，K表示要输出的答案。

示例代码

```
#include <stdio.h>
int main()
{
    int a1, b1, a2, b2, a3, b3, i, j, k;
```

```
    long long S1[10][10], S2[10][10], S3[10][10];
    long long S12[10][10], K[10][10], sum;
    scanf("%d%d%d%d%d%d", &a1, &b1, &a2, &b2, &a3, &b3);
    for (i = 0; i < a1; i++)
        for (j = 0; j < b1; j++)
            scanf("%lld", &S1[i][j]);
    for (i = 0; i < a2; i++)
        for (j = 0; j < b2; j++)
            scanf("%lld", &S2[i][j]);
    for (i = 0; i < a3; i++)
        for (j = 0; j < b3; j++)
            scanf("%lld", &S3[i][j]);
    for (i = 0; i < a1; i++)
    {
        for (j = 0; j < b2; j++)
        {
            sum = 0;
            for (k = 0; k < b1; k++)
            {
                sum += S1[i][k] * S2[k][j];
            }
            S12[i][j] = sum;
        }
    }
    for (i = 0; i < a1; i++)
    {
        for (j = 0; j < b3; j++)
        {
            sum = 0;
            for (k = 0; k < b2; k++)
            {
                sum += S12[i][k] * S3[k][j];
            }
            K[i][j] = sum;
        }
    }
    for (i = 0; i < a1; i++)
    {
        for (j = 0; j < b3; j++)
        {
            printf("%lld ", K[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

L三进制加法

难度	考点
5	高精度计算

题目解析

这道题目比较综合，需要对字符串和数组操作的熟练掌握，使用指针可以简化操作。

需要我们考虑的各种情况在**HINT**中已经有了一定程度的说明，这些情况都需要我们处理好。

主要的难点可能在于，虽然题目是“加法”，但是根据数字的符号情况不同，可能是加法或是减法。当然可以分别设计一个减法函数和一个加法函数，但其实这两种情况也可以统一在一起实现，示例代码中给出了一种解法。

还有一个难点可能在于最终结果的正负判断，这个需要我们仔细想清楚，示例代码也给出了一种解法，肯定也还有更好的解法。

p.s. 题目的最后三个测试点是纯粹的正整数和零的输入，是”真正的加法计算“的测试点，供同学们测试自己的加法代码。

示例代码(指针)

```
#include <stdio.h>
#include <string.h>
#define N 10007
char a[N], b[N];
int ans[N];
int main()
{
    while (~scanf("%s%s", a, b))
    {
        char *s1 = a[0] == '-' ? &a[1] : &a[0];
        char *s2 = b[0] == '-' ? &b[1] : &b[0];
        int sgn1 = a[0] == '-' ? -1 : 1;
        int sgn2 = b[0] == '-' ? -1 : 1;
        int sgn = strcmp(s1, s2) < 0 ? -1 : 1;
        int i = (int)strlen(s1) - 1, j = (int)strlen(s2) - 1, k = 0, w = 3;
        if (i < j || (i == j && sgn < 0))
        {
            char *temp = s1;
            s1 = s2;
            s2 = temp;
            sgn = -1;
        }
        else
            sgn = 1;
        i = (int)strlen(s1) - 1;
```

```
        j = (int)strlen(s2) - 1;
        while (i >= 0)
        {
            ans[k + 1] -= 1;
            s1[i] -= '0';
            ans[k] += w + s1[i];
            if (j >= 0)
            {
                s2[j] -= '0';
                ans[k] += sgn1*sgn2*s2[j];
            }
            ans[k + 1] += ans[k] / w;
            ans[k] %= w;
            k++; i--; j--;
        }
        while (ans[k] == 0 && k > 0)
            k--;
        if (!(k == 0 && ans[k] == 0))
            if ((sgn1 < 0 && sgn2 < 0) || (sgn1 > 0 && sgn2 < 0 && sgn < 0) ||
(sgn1 < 0 && sgn2 > 0 && sgn > 0))
                printf("-");
        while (k >= 0)
            printf("%d", ans[k--]);
        memset(ans, 0, sizeof(ans));
    }
    return 0;
}
```

示例代码(数组)

```
#include <stdio.h>
#include <string.h>
#define N 10007
char s1[N], s2[N], temp[N];
int ans[N];
int main()
{
    while (~scanf("%s%s", s1, s2))
    {
        int i = (int)strlen(s1), j = (int)strlen(s2), k = 0, w = 3;
        int sgn1 = s1[0] == '-' ? -1 : 1;
        int sgn2 = s2[0] == '-' ? -1 : 1;
        if (sgn1 == -1)
            for (k = 0; k < i; k++)
                s1[k] = s1[k + 1];
        if (sgn2 == -1)
            for (k = 0; k < j; k++)
                s2[k] = s2[k + 1];
```

```
        int sgn = strcmp(s1, s2) < 0 ? -1 : 1;
        i = (int)strlen(s1);
        j = (int)strlen(s2);
        if (i < j || (i == j && sgn < 0))
        {
            strcpy(temp, s1);
            strcpy(s1, s2);
            strcpy(s2, temp);
            sgn = -1;
        }
        else
            sgn = 1;
        k = 0;
        i = (int)strlen(s1) - 1;
        j = (int)strlen(s2) - 1;
        while (i >= 0)
        {
            ans[k + 1] -= 1;
            s1[i] -= '0';
            ans[k] += w + s1[i];
            if (j >= 0)
            {
                s2[j] -= '0';
                ans[k] += sgn1*sgn2*s2[j];
            }
            ans[k + 1] += ans[k] / w;
            ans[k] %= w;
            k++; i--; j--;
        }
        while (ans[k] == 0 && k > 0)
            k--;
        if (!(k == 0 && ans[k] == 0))
            if ((sgn1 < 0 && sgn2 < 0) || (sgn1 > 0 && sgn2 < 0 && sgn < 0) ||
(sgn1 < 0 && sgn2 > 0 && sgn > 0))
                printf("-");
        while (k >= 0)
            printf("%d", ans[k--]);
        memset(ans, 0, sizeof(ans));
    }
    return 0;
}
```