A Ange的火箭发动机

签到题，可以直接使用出题人给出的代码，也可以只用两个变量来做。

```
#include<stdio.h>
int T, n, a[20];

int main()
{
    int i, j;
    scanf("%d", &T);
    for (i = 1; i <= T; ++i)
        scanf("%d", &a[i]);
    for (i = 1; i <= T; i++) {
        for (j = 1; j <= a[i]; j++)
            printf("#");
        printf("\n");
    }
    return 0;
}
```

```
#include<stdio.h>
int T, n;

int main()
{
    scanf("%d", &T);
    while (T--) {
        scanf("%d", &n);
        while (n--) putchar('#');
        putchar('\n');
    }
    return 0;
}
```

B Ausar的DDL

签到题，冒泡排序可以通过。

```
#include<stdio.h>
int x, n, i, j, temp, da[10010];

int main()
{
    while (scanf("%d", &x) == 1)
        da[++n] = x;
    for (i = n; i >= 1; --i)
        for (j = 1; j < i; ++j)
            if (da[j] > da[j + 1]) {
                temp = da[j];
                da[j] = da[j + 1];
                da[j + 1] = temp;
            }
    for (i = 1; i <= n; ++i)
        printf("%d ", da[i]);
    return 0;
}
```

注意到每个数值都很小，我们可以使用桶排序。

```
#include<stdio.h>
int num[100010];

int main()
{
    int i, x;
    while (scanf("%d", &x) == 1)
        ++num[x];
    for (i = 0; i <= 100000; ++i)
        while (num[i])
            printf("%d ", i), --num[i];
    return 0;
}
```

C 一道绿色题

这个题也很简单，在输入的时候检查下有没有相邻的2，如果有就将他们都变成0。

```
#include<stdio.h>
int n, m, i, j, seat[110][110];

int main()
{
    scanf("%d%d", &n, &m);
    for (i = 1; i <= n; ++i)
        for (j = 1; j <= m; ++j) {
            scanf("%d", &seat[i][j]);
            if (seat[i][j] == 2 && seat[i][j - 1] == 2)
                seat[i][j] = seat[i][j - 1] = 0;
        }
    for (i = 1; i <= n; ++i)
        for (j = 1; j <= m; ++j)
            printf("%d%c", seat[i][j], j == m ? '\n' : ' ');
    return 0;
}
```

D bzb拯救世界

这个题是一道字符串替换题，出题人在下面给出了很多的提示。

因为字符串中有空格，可以使用fgets()读入，不过它会读到末尾的换行符。

如果不会fgets()也可以使用getchar()，出题人保证了末尾是一个分号。

```
#include<stdio.h>
#include<string.h>
int n, m, l;
char s[110], t[110], str[110];

int read(char *ch) {
    int len;
    fgets(ch, 10010, stdin);
    len = strlen(ch);
    while (ch[len - 1] != ';') --len;
    ch[len - 1] = '\0';
    return len - 1;
}
```

```
int main()
{
    int i, j;
    //读入以及处理末尾的分号
    n = read(s);
    m = read(t);
    l = read(str);
    for (i = 0; i < n; ++i)
    {
        //从i位置开始匹配
        int match = 1;
        for (j = 0; j < m && match; ++j)
            if (s[i + j] != t[j])
                match = 0;
        //match=1表示匹配成功，直接输出用来替换的字符串，并且跳过这一段(i = i + m - 1)
        //match=0表示匹配失败，直接输出当前这个位置的字符即可
        if (match) printf("%s", str), i = i + m - 1;
        else printf("%c", s[i]);
    }
    printf(";");
    return 0;
}
```

E 统计

题目中规定，只要不是空格、换行、制表，就表示是一个单词，而这三种字符（我们下面将他们统称为空字符）是分隔两个单词的标志。

我们一个一个地读入字符，会发生三种情况：

- ①上一个读的是字母，下面读到了空字符，就说明我们结束了上一个单词。
- ②上一个读的是空字符，下一个读的是字母，说明发现了一个新单词。
- ③前后两个都是空字符或都是字母，则什么也不会发生。

我们不用数组，因为我们不需要重复使用已经被用过的数据。

```
#include <stdio.h>
#define IN 1
#define OUT 0
//两个宏定义增加了代码可读性
int main()
{
    char c;
    int n = 0, state = OUT;
    while ((c = getchar()) != EOF) //这里写(c = getchar()) > 0也行
    {
        //1情况和连续空字符3情况会进入这里
        if (c == ' ' || c == '\n' || c == '\t')
            state = OUT;
        else if (state == OUT) //2情况会进入这里
        //注意我们发现新单词的标志是新单词的起点，而不是终点
        {
            state = IN;
            ++n;
        }
        //连续字母的3情况会直接出去
    }
```

```
    printf("%d\n", n);
    return 0;
}
```

F 小明的学号

这个题其实是一道简单题。

对于读入的处理有很多种方法，比如一个字符一个字符的读入，或者直接读一个字符串，或者读一个大整数再通过取模得到每一位。

这里给出的是scanf("%1d")这种做法，只读一个数字。

```
#include<stdio.h>
int i, j, k, l, r, ans, sum, id[10];

int main()
{
    for (i = 0; i < 8; ++i)
        scanf("%1d", &id[i]);
    scanf("%d", &k);
    while (k--) {
        sum = 0;
        scanf("%d%d", &l, &r);
        for (i = l; i <= r; ++i)
            sum += id[i];
        ans += sum % 8;
    }
    printf("%d\n", ans);
    return 0;
}
```

G CWD的套圈游戏

我们首先找出来价值最高的，再找出难度最低的第一个就ok了。

其实我们并不需要排序，因为我们只要找出来唯一一个答案，不用关心后面的奖品谁大谁小。

我们也不用二维数组，因为在奖品按照位置排序的过程中，一维数组就已经可以胜任了。

c语言中二维数组的单、双下标转化公式：$a[i]=a[i/n][i\%n]$; 其中n是方阵的尺寸。

```
#include <stdio.h>
//大数组放外面，因为main函数里的空间很有限
int vi[10010] = {0};
int hi[10010] = {0};
int maxValueIndex[10000]={0};//最大价值栈，里面装了有最大价值的物品的索引
int maxValueNum=0;//栈顶

int main()
{
    int i, n, maxValue = 0, lowHard=101, ans, N;
    scanf("%d", &N);
    n = N * N;
    //输入价值
    for (i = 0; i < n; ++i)
```

```
    {
        scanf("%d", &vi[i]);
        if (vi[i] > maxValue)// 顺便找出最大价值
            maxValue = vi[i];
    }
    //输入难度
    for (i = 0; i < n; ++i)
        scanf("%d", &hi[i]);
    //找出最高价值，压栈
    for (i = 0; i < n; ++i)
        if (vi[i] == maxValue)
        {
            maxValueIndex[maxValueNum] = i;
            ++maxValueNum;
        }
    //找出第一个最小难度奖品
    for (i = 0; i < maxValueNum; ++i)
        if (hi[maxValueIndex[i]] < lowHard)
        {
            lowHard = hi[maxValueIndex[i]];
            ans = maxValueIndex[i];
        }
    printf("%d %d", ans / N + 1, ans % N + 1);
    return 0;
}
```

H 二分查找

首先，我们将这两个排名整合到一起，由于两个班的排行都是实现排好序的，所以直接用一个简单的归并(详见代码)就可以了，得到了一个降序排列的数组all。

然后，就是使用二分法查找其中某个成绩(grade)第一次出现的位置就对应他的名次。

定义二分的区间为[l, r)，以及mid = (l + r) / 2。

很容易想到：如果all[mid] > grade，就让left变成mid，如果all[mid] < grade，就让right变成mid。

那么当all[mid] == grade时呢?

看看下面的数组:{ 3, 2, 2, 2, 2, 2, 1 }，l = 0, r = 6, mid = 3, grade = 2，如果下一步让l = mid，就会失去第一次出现的2了。

所以应该让r = mid。

那么，我们再来看看二分的终点，如果我们以left = right - 1为终点（一个只有一个整数的区间），并将all[l]的值作为结果。

就可能会出现问题：all[r]实际是第一个出现的grade，而all[l]<grade。所以我们要以l == r作为二分终点。

这时就会出现新的问题：二分无法结束!

比如说l = 2, r = 3, mid = 2, all = { 5, 4, 3, 2, 1}, grade = 2, 这时，all[mid] > grade，于是l = mid，l还等于2！就死循环了。

因此我们要规定all[mid] > grade时，l = mid + 1，来解决这个问题（为什么不能是另一边-1呢？）。

以上几个问题解决后，我们就得到了下面的示例代码了。

示例代码1

```c
#include <stdio.h>
int a[10005], b[10005], all[20005];

int m, n;
//将两个班的成绩整合到一起(归并)
void getTogether()
{
    int i_a, i_b, i_all;
    i_a = n - 1, i_b = 0, i_all = 0;
    while (i_a >= 0 && i_b < m)
        if (a[i_a] > b[i_b])
            all[i_all++] = a[i_a--];
        else
            all[i_all++] = b[i_b++];
    while (i_a >= 0)
        all[i_all++] = a[i_a--];
    while (i_b < m)
        all[i_all++] = b[i_b++];
}

//从left到right的区间，左闭右开，用来寻找成绩grade的排名
int getRank(int left, int right, int grade)
{
    int mid;
    mid = (left + right) >> 1;
    if (left == right)
    {
        if (grade == all[left])
            return left;
        return -2; //为什么是-2呢? 因为后面输出有个+1
    }
    if (all[mid] > grade)
        return getRank(mid + 1, right, grade);
    return getRank(left, mid, grade);
}

int main()
{
    int i, grade, k;
    while (scanf("%d %d %d", &n, &m, &k) > 0)
    {
        //输入
        for (i = 0; i < n; ++i)
            scanf("%d", &a[i]);
        for (i = 0; i < m; ++i)
            scanf("%d", &b[i]);
        //归并
        getTogether();
        all[m + n] = -1;
        for (i = 1; i < k; ++i)
        {
            scanf("%d", &grade);
            //递归得到答案
            printf("%d ", getRank(0, m + n, grade) + 1);
        }
        //由于是多行输入，每组输出的结尾要是\n,不是空格
        scanf("%d", &grade);
        printf("%d\n", getRank(0, m + n, grade) + 1);
    }
    return 0;
```

```
}
```

示例代码2

```
#include <stdio.h>
#include <limits.h>
#define ms 10005
//升序数组找下界，即找到[l,r)中第一个大于等于val的数字
//也可以理解为有序数组中可以插入val，而不会破坏数组顺序的第一个位置
//个人认为左闭右开的区间比较好理解，你也可以试试闭合区间的二分查找
int *lower_bound(int* l, int* r, int val)
{
    int *m;
    while (l < r)
    {
        m = l + ((r - l) >> 1);
        if (*m < val)
            l = m + 1;
        else
            r = m;
    }
    return l;
}
//升序数组找上界，即找到[l,r)中第一个大于val的数字
//也可以理解为有序数组中可以插入val，而不会破坏数组顺序的最后一个位置
//对比一下和lower_bound有什么不同
int *upper_bound(int* l, int* r, int val)
{
    int * m;
    while (l < r)
    {
        m = l + ((r - l) >> 1);
        if (*m <= val)
            l = m + 1;
        else
            r = m;
    }
    return l;
}
//降序数组找下界，即找到[l,r)中第一个小于等于val的数字
//仍然可以理解为有序数组中可以插入val，而不会破坏数组顺序的第一个位置
int *lower_bound1(int* l, int* r, int val)
{
    int * m;
    while (l < r)
    {
        m = l + ((r - l) >> 1);
        if (*m > val)
            l = m + 1;
        else
            r = m;
    }
    return l;
}
//降序数组找上界，即找到[l,r)中第一个小于val的数字
//可以理解为有序数组中可以插入val，而不会破坏数组顺序的最后一个位置
int *upper_bound1(int* l, int* r, int val)
{
    int *m;
    while (l < r)
```

```
    {
        m = l + ((r - l) >> 1);
        if (*m >= val)
            l = m + 1;
        else
            r = m;
    }
    return l;
}

int a[ms], b[ms];
int n, m;

int main()
{
    int k, i;
    while (~scanf("%d%d%d", &n, &m, &k))
    {
        for (i = 0; i < n; i++)
            scanf("%d", &a[i]);
        for (i = 0; i < m; i++)
            scanf("%d", &b[i]);
        while (k--)
        {
            int c; scanf("%d", &c);
            //a[l]>c
            int l = upper_bound(a, a + n, c) - a;
            //b[r]>=c
            int r = lower_bound1(b, b + m, c) - b;
            //两个数组中大于c的分别有n-l个和r个，因为要计算排名，所以最终结果+1
            int res = n - l + r + 1;
            //如果存在c，则c必须在a[l-1]和b[r]处出现一次
            //注意边界条件的判断
            if ((l > 0 && a[l - 1] == c) || (r < m&& b[r] == c))
                printf("%d ", res);
            else
                printf("-1 ");
        }
        printf("\n");
    }
    return 0;
}
```