C2 - Solution

A 这里是 ASCII！

难度	考点
1	多组数据输入，ASCII 码

题目分析

本题要求根据输入的ASCII码输出其对应的字符。

Hint 2 提供了一种输入一个 `char` 型变量，将其强制类型转换为 `int` 型变量输出的方法，即将字符转化为 ASCII 码。本题要求将 ASCII 码转化为字符，因此，对于每一组数据，我们可以先读入一个 `int` 型的变量，然后通过强制类型转换将其转变为 `char` 型的变量，并将其输出。

示例代码

```
#include <stdio.h>
int main()
{
    int n;
    while (scanf("%d", &n) != EOF)
    {
        printf("%c\n", (char)n);
    }
    return 0;
}
```

B 这里是 BUAA

难度	考点
1	字符输入输出，条件语句

题目分析

每次读入 1 个字符，遇到 `C,V,B` 则什么都不输出，否则原样输出这个字符。

字符的输入可以用 `scanf("%c",&c)` 或 `c=getchar()` 来实现，都可以达到读入下一个字符的效果。字符的输出可以用 `printf("%c",c)` 或 `putchar(c)` 来实现，都可以达到输出一个字符 `c` 的效果。

注意到读入第一行的整数 n 之后，下一个字符实际上是行末的一个换行符，而不是第二行的第一个字母。因此，我们在 `scanf("%d",&n)` 读入整数 n 后，还需要想办法处理掉那一个换行符。

处理方法可以有很多种，这里介绍比较常见的两种：

- 采用题目中 Hint 的做法，使用一个 `getchar()` 读掉这个换行符。
- 使用 `scanf(" ")`，在控制占位符字符串中加入一个空格，这个空格会使 scanf 函数在读操作中略去输入中的一个或多个空白字符，这也是题解 std 中使用的做法。当然，这种做法在某些特定情境下有一定的局限性，比如当你需要在第二行读入一个可能有行首空格的字符串时，这可能会产生一些错误（想一想会造成什么后果？）。

示例代码

```
#include <stdio.h>

int main()
{
    int n;
```

```c
    char c;
    scanf("%d", &n);
    getchar();
    //scanf("%d ", &n); //直接在 scanf 的格式化字符串末加入一个空格，达到略去空白字符的效果
    for (int i = 1; i <= n; i++)
    {
        scanf("%c", &c);//字符读入
        if (c != 'C' && c != 'V' && c != 'B') //判断是否是特定字符
            printf("%c", c);
    }
    return 0;
}
```

C 小亮的乱码书信

难度	考点
2	ASCII码

题目分析

本题要求对读入的字符进行转换，分为数字和小写字母两种情况，因此需要我们先对字符类型进行判断。

- 如果字符为数字，则表达式 `s>='0' && s<='9'` 为真。
- 如果字符为小写字母，则表达式 `s>='a' && s<='z'` 为真。

另外也可以使用 `ctype.h` 库中的 `isdigit()` 和 `islower()` 两个函数分别判断是否为数字或小写字母。

根据题目给出的翻转规则，我们可以得出下面两个表达式。

- 如果字符s为数字，则翻转后的字符表达式为`'9'-(s-'0')`。
- 如果字符s为小写，则翻转后的字符表达式为`'z'-(s-'a')`。

公式解释：

通过表达式 `s-'0'` 我们可以得到 s 与 `'0'` 的距离，也就是 s 对应的数字的值。

因此表达式 `'9'-(s-'0')` 所得到的就是翻转后的字符的ASCII码值。

参考代码

```c
#include <stdio.h>

int main() {
    char s;
    while(scanf("%c", &s) != EOF) //每次循环读取单个字符，直至输入结束
    {
        if (s >= 'a' && s <= 'z') //如果s是小写字母
        {
            printf("%c", 'z' - (s - 'a'));
        }
        else if (s >= '0' && s <= '9') //如果s是数字
        {
            printf("%c", '9' - (s - '0'));
        }
        else //如果不是数字、小写字母，依据题意只能是空格
        {
            printf(" ");
        }
    }
}
```

D violet 的回寝之路

难度	考点
3	时间比较 判断结构

题目分析

依据题意，violet 到达寝室的总时长由三部分组成：

- 从麦当劳走到任意一个门的时间 t_1
- 在校门口的等待时间 t_2
- 从校门口到达宿舍的时间 t_3

由于到达两个们的时间相同，可以暂时忽略t_1，仅考虑两种选择下 t_2+t_3 的大小关系。通过分析，当北门的等候时间 $t_2 \leq 10$ 时，应选择北门，其余情况下应选择东门。即到达门口的时间处于 $6:20$ 到 $22:30$ 之间时应选择北门（包含边界），考虑 t_1 后，向前推 10 分钟，即出发时间处于 $6:10$ 到 22：20 之间时选择北门（包含边界）。在代码编写时只需要将小时数 $h=6$ 和 $h=22$ 的情况单独考虑即可。

注意：根据题意，出发时间恰好为 $6:10$ 或 22：20 时，应选择北门。

示例代码

```c
#include <stdio.h>
int main()
{
    int n, i;
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        int h, m;
        scanf("%d%d", &h, &m);
        if (h == 6 && m >= 10)
            printf("N\n");
        else if (h == 22 && m <= 20)
            printf("N\n");
        else if (h > 6 && h < 22)
            printf("N\n");
        else
            printf("E\n");
    }
    return 0;
}
```

E violet 打保龄球

难度	考点
3	判断结构 循环结构

题目分析

通过分析，第 i 轮的得分和第 $i-1$ 和 $i-2$ 轮击中的瓶子数有关，可以建立两个变量来存储这两个值，并且每遍历到下一个数，就更新这两个变量。

学习数组后，也可以直接用数组存下所有的数，自然也可以方便的访问前一个和前两个数。

示例代码

```c
#include <stdio.h>
int main()
{
    int n, i, p1 = 0, p2 = 0;
    scanf("%d", &n);
    int a = 0, b = 0, c; // 分别代表第i-2, i-1, i个数

    for (i = 0; i < n; i++)
    {
        scanf("%d", &c);
        if (a == 10 || b == 10)
            p1 += c * 2;
        else
            p1 += c;

        a = b; // 更新变量
        b = c;
    }

    a = b = 0; // 计算 Lilsio的得分前把前两个数清零
    for (i = 0; i < n; i++)
    {
        scanf("%d", &c);
        if (a == 10 || b == 10)
            p2 += c * 2;
        else
            p2 += c;

        a = b; // 更新变量
        b = c;
    }

    printf("%d %d\n", p1, p2);
    if (p1 > p2)
        printf("violet");
    else if (p1 < p2)
        printf("Lilsio");
    else
        printf("Let's play again!");
    return 0;
}
```

F 向量计算器

难度	考点
4	模拟

题目分析

按照要求模拟运算即可，需要注意的是计算时，变量较多，要注意到底是哪个变量和哪个变量进行运算。本题难度不大，考验同学们仔细程度，并且在发现错误后，要耐心寻找错误点。

示例代码

```c
#include<stdio.h>
int main()
{
    long long a1, a2, a3, b1, b2, b3, c1, c2, c3;
    scanf("%lld%lld%lld%lld%lld%lld%lld%lld%lld", &a1, &b1, &c1, &a2, &b2, &c2, &a3, &b3, &c3);
    //r1 + r2
```

```
    printf("%lld %lld %lld\n", a1 + a2, b1 + b2, c1 + c2);
    //r1 - r2
    printf("%lld %lld %lld\n", a3 - a2, b3 - b2, c3 - c2);
    //r1 . r2
    printf("%lld\n", a3 * a2 + b3 * b2 + c3 * c2);
    //r1 x r2
    printf("%lld %lld %lld\n", b1 * c2 - c1 * b2, c1 * a2 - a1 * c2, a1 * b2 - a2 * b1);
    //(r1 x r3) . r2
    printf("%lld\n", a1 * b3 * c2 + a3 * b2 * c1 + a2 * b1 * c3 - a2 * b3 * c1 - a3 * b1 * c2 - a1 *
b2 * c3);
    //(r3 x r2) x r1 = (r3 . r1) . r2 - (r2 . r1) . r3
    long long t1 = a1 * a3 + b1 * b3 + c1 * c3, t2 = a1 * a2 + b1 * b2 + c1 * c2;
    printf("%lld %lld %lld\n", t1 * a2 - t2 * a3, t1 * b2 - t2 * b3, t1 * c2 - t2 * c3);
    return 0;
}
```

拓展思考

如果给你 n 个向量坐标，对其中任意两个向量进行操作，你该如何做呢？

G 解方程

难度	考点
4	if 的使用

问题分析

由题意分析可知，该方程由于 a, b, c 取值的变化，可能为二次方程、一次方程，还有可能不含未知量 x。

对于二次方程，需要判断判别式的大小，分为两根、一根和无实根的三种情况讨论；

对于一次方程，直接求得一个实根即可；

对于不含未知量的方程，若 $c = 0$，则有无穷多解，否则无解。

参考代码

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

int main()
{
    int a, b, c, d;
    double x1, x2, t;
    scanf("%d%d%d", &a, &b, &c);
    d = b * b - 4 * a * c; // 判别式
    if (a) // a不为0，为二次方程
    {
        if (d < 0)
            printf("No real root");
        else if (d == 0)
            printf("%.2f", -b / 2.0 / a);
        else
        {
            x1 = (-b + sqrt(d)) / 2 / a;
            x2 = (-b - sqrt(d)) / 2 / a;
            if (x1 > x2)
            {
                t = x1;
                x1 = x2;
                x2 = t;
            }
            printf("%.2f %.2f", x1, x2);
        }
```

```
    }
    else // a为0
    {
        if (b) // b不为0
            printf("%.2f", -1.0 * c / b);
        else if (c) // b为0, c不为0
            printf("No real root");
        else // a,b,c均为0
            printf("infinite solutions");
    }
    return 0;
}
```

H Gino 的 rks

难度	考点
5	浮点数的计算、浮点数比大小的特殊处理

题目分析

由题意，在总共 $a+b+c+d$ 个音符中，a 个能够拿到全部准度，b 个能够拿到 65% 的准度。由此可以得到 acc 与 a,b,c,d 的关系 $acc=\dfrac{a+0.65b}{a+b+c+d}\cdot 100\%$

写成代码时注意避免出现两整型相除的情况，这样会使得得数的小数部分被截断

输出要求是以百分数形式输出，别被这句话吓到了，百分数其实就是将原数字乘以 100，并在后面添上百分号（注意打印百分号需要转义），就像这样：

```
printf("%f%%", acc * 100);
```

题目里一个比较难的考点就是保留四位有效数字的问题。题目中给出了 acc 的范围为 $1\%<acc\le 100\%$，我们由此可以得出 acc 在不同的取值范围时百分号前应该保留的小数位数。当 $1\%<acc<10\%$ 时，应当保留三位小数，如 `9.328%` ，当 $10\%\le acc<100\%$ 时，应当保留两位小数，如 `99.50%` ，当 $acc=100\%$ 时，应当保留一位小数，如 `100.0%` 。特别的，由于浮点数计算时的误差，acc 的计算结果有可能被存储为 `0.0999999999` ，如果你的评测结果是 `WA 0.2` ，那么极有可能是被这种情况卡了，你可以试试以下数据，看看你的程序会怎样输出：

```
15.8 5 8 44 45
```

为了解决这个问题，我们需要对于取值区间的端点进行特殊判断。

利用 `<math.h>` 的 `fabs` 函数，我们可以这样判断 acc 是否等于 10%：

```
const double eps = 1e-10;
if (fabs(acc - 0.1) < eps) //acc 与 0.1 的差距不到 10 的 -10 次方
{
    printf("10.00%%");
}
/* 注意不要使用 abs 函数！abs 只能用于求整型数据的绝对值，不能用于浮点型数据！*/
```

即将 $acc\ge 0.1$ 拆成两种情况 $acc>0.1$ 和 $acc=0.1$ 。当然也可以使用 `acc > 0.1 - eps` 来判断。

同理，在求 rks 的时候，$acc=70\%$ 也需要特殊判断，不过出题人似乎并没有在这里卡大家。所以大家的代码也许能够 `AC` ，但仍然有漏洞。

为了规避这个问题，建议大家在计算 acc 时尽量使用整型而不是浮点型，避免浮点型计算的累计误差，本题求 acc 的两种代码展示：

```
acc = (a + 0.65 * b) / (a + b + c + d);
/* 直接由题意写出的计算过程 */

acc = (double)(a * 100 + 65 * b) / (100 * (a + b + c + d));
/* 将分子和分母各扩大 100 倍，使两者都为整数，记得两者相除之前将分子或分母强制转换成 double */
```

示例代码

```c
#include <stdio.h>
#include <math.h>
const double eps = 1e-10;
int main(void)
{
    int a, b, c, d;
    double m, acc, rks;
    while (scanf("%lf%d%d%d%d", &m, &a, &b, &c, &d) != EOF)
    {
        // acc = (double)(a + 0.65 * b) / (a + b + c + d); // 不建议这样写
        acc = (double)(a * 100 + 65 * b) / (100 * (a + b + c + d));
        //if (acc >= 0.7) // 不建议这样写
        if (acc > 0.7 || fabs(acc - 0.7) < eps)
        {
            rks = ((acc - 0.55) / 0.45) * ((acc - 0.55) / 0.45) * m;
        }
        else
        {
            rks = 0;
        }
        if (fabs(acc - 1) < eps)
            printf("100.0%% ");
        else if (fabs(acc - 0.1) < eps)
            printf("10.00%% ");
        else if (acc > 0.1)
            printf("%.2f%% ", acc * 100);
        /*以上两个else if可以统一为:
        else if(acc > 0.1 - eps)
            printf("%.2f%% ", acc * 100);
        */
        else
            printf("%.3f%% ", acc * 100);
        printf("%.4f\n", rks);
    }
    return 0;
}
```

拓展

也许有同学会注意到，有时不得不使用浮点型来进行运算，累计误差难以避免，最后有可能得出类似 `13.4999999999` 的数值，此时进行四舍五入很容易出问题。对此，一般的解决方式是将得数加上一个较小的数 `eps` 之后再进行四舍五入。可是，这个较小的数具体取多少合适，有点难计算出来。对于这个题，合适的取值大约为 `eps = 1e-9` 或 `eps = 1e-10` 。不过为了不让大家在这种位置钻了牛角尖，助教团队保证**在题面中没有特殊说明的情况下，这种在进位临界点的数据不会出现在测试点中，或者会允许输出的结果有一定的误差范围。**

`I` Permutation!

难度	考点
5	构造

题目分析

本题构造方法不唯一。下面介绍一种较为简单的思路。

首先，考虑一些较小的情况。

- 当 $n = 1$ 时，显然排列 (1) 符合题目要求。
- 当 $n = 2, 3$ 时，容易得知此时无解。
- 当 $n = 4$ 时，我们可以得到一组可行的解 $(3, 1, 4, 2)$

当 n 继续增大时，之后每次遇到偶数就放到排列的 $(3, 1, 4, 2)$ 的最前面，遇到奇数就放到最后面，最后得到的序列形如 $(\cdots, 10, 8, 6, 3, 1, 4, 2, 5, 7, 9, \cdots)$ ，可以证明该方案满足题目要求。

示例代码

```
#include <stdio.h>
int main()
{
    int n;
    scanf("%d", &n);
    if(n == 1)
        printf("1\n1");
    else if(n == 2 || n == 3)
        printf("-1");
    else
    {
        printf("%d\n", n);
        for(int i = n - n % 2; i > 4; i -= 2) //从最大的不大于n的偶数开始，递减输出大于4的偶数
            printf("%d ", i);
        printf("3 1 4 2 ");
        for(int i = 5; i <= n; i += 2) //从5开始输出奇数，到最大的不大于n的奇数为止
            printf("%d ", i);
    }
    return 0;
}
```

其他思路

首先，考虑一些较小的情况。

- 当 $n = 1$ 时，显然排列 (1) 符合题目要求。
- 当 $n = 2, 3$ 时，容易得知此时无解。
- 当 $n = 4$ 时，我们可以得到一组可行的解 $(2, 4, 1, 3)$ ，我们发现此时可以把这种形式的解推广到所有形如 $4k$ 的数上，每当 k 增加 1 时，我们就给序列后加上一段 $(4k - 2, 4k, 4k - 3, 4k - 1)$ 。
- 当 $n = 5$ 时，我们可以得到 $(1, 3, 5, 2, 4)$ 是可行的，有了上面的经验，我们可以构造出所有形如 $4k + 1$ 的数的解，每当 k 增加 1 时，我们给序列后加上一段 $(4k - 1, 4k + 1, 4k - 2, 4k)$ 。
- 同理当 $n = 6$ 时，得到一组解 $(1, 4, 2, 5, 3, 6)$ ，可以构造出所有形如 $4k + 2$ 的数的解，每当 k 增加 1 时，我们给序列后加上一段 $(4k, 4k + 2, 4k - 1, 4k + 1)$ 。
- 同理 $n = 7$ 时，可以得到所有形如 $4k + 3$ 的数的解，构造如下：
  $(1, 3, 6, 4, 2, 5, 7, 9, 11, 8, 10, \cdots, 4k + 1, 4k + 3, 4k, 4k + 2)$ 。

示例代码

```
#include<stdio.h>
int main()
{
    int n;
    scanf("%d", &n);
    if(n == 1)
        printf("1\n1");
    else if(n < 4)
        printf("-1");
    else
```

```
    {
        printf("%d\n", n);
        if(n % 2 == 0)
        {
            if(n % 4 == 0) //n被4整除
            {
                for(int i = 1; i <= n / 4; i++)
                    printf("%d %d %d %d ", 4 * i - 2, 4 * i, 4 * i - 3, 4 * i - 1);
            }
            else           //n除以4余2
            {
                printf("1 4 2 5 3 6 ");
                for(int i = 2; i <= n / 4; i++)
                    printf("%d %d %d %d ", 4 * i, 4 * i + 2, 4 * i - 1, 4 * i + 1);
            }
        }
        else
        {
            if(n % 4 == 1) //n除以4余1
            {
                printf("1 3 5 2 4 ");
                for(int i = 2; i <= n / 4; i++)
                    printf("%d %d %d %d ", 4 * i - 1, 4 * i + 1, 4 * i - 2, 4 * i);
            }
            else           //n除以4余3
            {
                printf("1 3 6 4 2 5 7 ");
                for(int i = 2; i <= n / 4; i++)
                    printf("%d %d %d %d ", 4 * i + 1, 4 * i + 3, 4 * i, 4 * i + 2);
            }
        }
    }
    return 0;
}
```

J czx 与小羊函数

难度	考点
7	数学，动态规划

题目分析

这道题的输入 n 大小达到了 10^5 位的整数，显然用 `int` 类型或是 `long long` 类型都放不下，考虑作为一个字符串来读入和处理。

n 非常大，从 1 枚举到 n 需要非常长的时间，是不现实的。考虑 **数位 dp**，从低位逐步向高位进行转移。下面给出数位 dp 的思考过程。

为了方便考虑，我们不妨令 $p(0)=1$，并取 $sum(l,r)=p(l)+p(l+1)+...+p(r)$，则有：
$p(1)+p(2)+...+p(n)=sum(0,n)-1$

我们尝试求解 $sum(0,n)$。

令 a_i 表示 n 的 **最后 i 位数字** 组成的数（若 $n=789645$，则 $a_1=5$，$a_2=45$，$a_3=645$，...，$a_6=789645$），dp_i 表示 $sum(0,a_i)$。待求解的 $sum(0,n)$ 即为 dp_{len}，其中 len 为 n 的位数。

考虑从低位向高位进行转移，思考如果已知 dp_i，如何推出 dp_{i+1}？

假设 n 的 **倒数第 i 位数** 为 x_i，则 $a_{i+1}=x_{i+1}\times 10^i+a_i$。

当 $x_{i+1}=0$ 时，显然 $dp_{i+1}=dp_i$。

当 $x_{i+1}\neq 0$时，不难发现：

$$sum(x_{i+1}\times 10^i, x_{i+1}\times 10^i+a_i)=x_{i+1}\times sum(0,a_i)=x_{i+1}\times dp_i$$

要求 dp_{i+1}，现在只需求 $sum(0, x_{i+1} \times 10^i - 1)$。注意到：

$$\begin{aligned}
&sum(0, x_{i+1} \times 10^i - 1) \\
&= sum(0, 10^i - 1) + sum(1 \times 10^i, 2 \times 10^i - 1) + sum(2 \times 10^i, 3 \times 10^i - 1) + \ldots + sum((x_{i+1} - 1) \times 10^i, x_{i+1} \times 10^i - 1) \\
&= sum(0, 10^i - 1) + 1 \times sum(0, 10^i - 1) + 2 \times sum(0, 10^i - 1) + \ldots + (x_{i+1} - 1) \times sum(0, 10^i - 1) \\
&= (1 + 1 + 2 + \ldots + (x_{i+1} - 1)) \times sum(0, 10^i - 1) \\
&= (1 + \frac{x_{i+1}(x_{i+1} - 1)}{2}) \times sum(0, 10^i - 1)
\end{aligned}$$

思考如何求 $sum(0, 10^i - 1)$。不妨令 $b_i = sum(0, 10^i - 1)$。类比上述的做法，有

$$\begin{aligned}
b_i &= sum(0, 1 \times 10^{i-1} - 1) + sum(1 \times 10^{i-1}, 2 \times 10^{i-1} - 1) + \ldots + sum(9 \times 10^{i-1}, 10^i - 1) \\
&= (1 + 1 + 2 + \ldots + 9) \times sum(0, 1 \times 10^{i-1} - 1) \\
&= 46 \times b_{i-1}
\end{aligned}$$

又因为 $b_1 = sum(0, 9) = 46$，因此得到通式 $b_i = 46^i$。

结合上述分析和推导可以发现：

$$dp_{i+1} = \begin{cases} (1 + \dfrac{x(x-1)}{2}) \times 46^i + x \times dp_i & x_{i+1} > 0 \\ dp_i & x_{i+1} = 0 \end{cases}$$

而 $dp_1 = 1 + \frac{x_1(1+x_1)}{2}$，从而我们可以根据上式一步一步推出 dp_2，dp_3，…，dp_{len}，从而求解出最终答案。另外由于在 dp 过程中每次只需要用到前一次的结果，我们可以不必开一个 dp 数组来转移，而是直接对结果进行转移。

最终结果为 $dp_{len} - 1$ 。

最后，不要忘了一边运算一边取模。

示例代码

```
#include <stdio.h>
#include <string.h>
#define maxN 100000

const int mod = 1e9 + 7;

long long res, base = 1;
char s[maxN + 5];
int num[maxN + 5];

int main() {
    scanf("%s", s);
    int len = strlen(s);
    for (int i = 0; i < len; i++)
        num[i] = s[i] - '0';
    res = 1 + (num[len - 1] + 1) * num[len - 1] / 2; //最低位
    for (int i = len - 2; i >= 0; i--)
    {
        base = base * 46 % mod;
        if (num[i] != 0)
            res = (num[i] * res + ((num[i] - 1) * num[i] / 2 + 1) * base % mod) % mod;
    }
    res = (res - 1 + mod) % mod;
    printf("%lld\n", res);
    return 0;
}
```

- End -