1 - 判断可逆素数

封装实现判断素数的函数跟逆置的函数即可。

2 - 矩阵相交

编程实现提示给的思路即可：

> 输入的两点可以是矩形任一对角线上的端点，求相交的面积可以先求矩形在 X 轴和 Y 轴上的交集。
>
> 矩形在 X 轴上的交集可以按照如下算法进行求解：
>
> 假设 AX1 和 AX2 中的较大值为 MAX_AX，较小值为 MIN_AX；BX1 和 BX2 中的较大值为 MAX_BX，较小值为 MIN_BX。用 MAX_AX 和 MAX_BX 中的较小者减去 MIN_AX 和 MIN_BX 中的较大者，结果为正表示两矩形在 X 轴上的交集，若为负则表示不相交。

Y 方向其实也同理。

3 - 求差集

先把集合 A 读进来，并求出 A 中元素的个数 `cnt`。读入 B 集合中的元素 x 时，遍历 A，如果 `A[i] = x`，则标记 `A[i] = -1`。最后输出 A 中所有不等于 -1 的元素。

4 - 矩阵运算

读入符号和进行矩阵计算的过程没什么特殊的，最后右对齐补空格要使用格式化字符串 %5d（如果在前面补 0 则是 %05d）。

5 - 文件拷贝 2

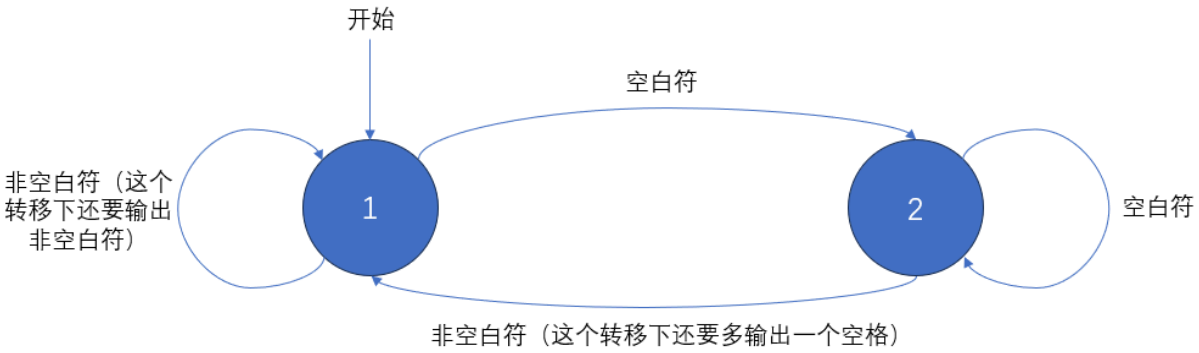
我们可以用状态转移的思想来解决这个问题，状态转移思想的关键就在于**定义状态以及状态之间的转移。**

在逐个字符处理的过程中，我们需要定义以下两个状态：

- **非空白状态（状态 1）**：当我们遇到非空白符，我们进入这个状态。

- **空白状态（状态 2）**：当我们遇到空白符（包括空格符、制表符等），我们进入这个状态；在这个状态下，我们连续遇到多个空白符时，依然保持在这个状态。

两个状态之间的转移过程如下：



然后编程实现这张图即可，这种状态转移的思想在文本格式化、输入验证等题目中还是比较常用的。

文件读入与输出可以参考 **选填板子** 下的 **文件** 部分，里面有一些代码示例，在这里为了方便我就直接使用 `freopen` 了。

6 - 求两组整数的异或集

这题其实有一个偷懒的办法：既然输入保证了`每组整数中元素不重复`，那我们直接把两组整数读进一个数组，最后看看对每个元素数组中有没有相同元素就可以，如果有的话将两个数字都删除掉。

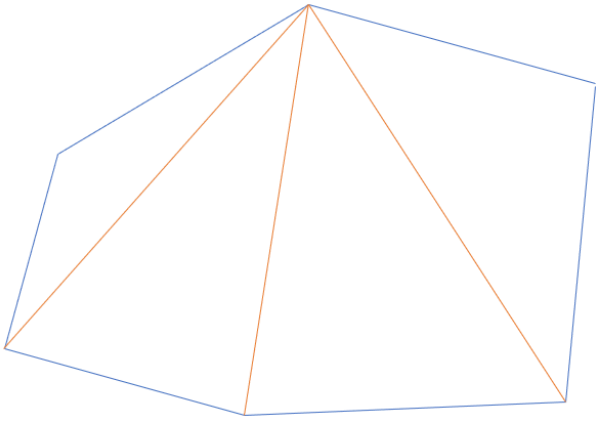
如果想真正区分两组整数的话也可以，可以使用 `scanf("%d%c", &a, &ch)` 进行读入，当发现 `ch` 不是空格时就说明一组数已经读完了。

删除的时候可以使用打标记的方法，而不真正删除。由于还涉及到一个排序操作，所以我们可以最后再去重。

代码实现使用了偷懒的那种办法，别忘了不定组输入如何结束输入（Windows 下是 `ctrl + z`，Linux 和 Mac 是 `ctrl + d`）。

7 - 凸多边形面积

可以将凸多边形拆成若干三角形，求解每个三角形面积后求和，如图所示：



求三角形面积可以用海伦公式或者叉乘，代码实现用的是叉乘的方法。

8 - 整数的N进制字符串表示

输入如果是个负数，先打个标记转换为正数，然后再进行一个简单的进制转换就可以了。进制转换的代码在去年的程设课上进行过，在这里我们不多讲解，可以参考实现。

要注意很有可能在特殊数据 `0` 的输出上出现错误（可能没有输出），在这类数字按照字符串输出的问题中，0 是一个特殊数据。

9 - 最长升序子串

非常经典的动态规划问题：最长不下降子序列。这里直接引用去年 `C8 - H - 圣诞老人送礼物` 的题解：

> 有一排高低不一的楼房，然后求出这些楼房中最长的高度单调递增的子序列（一排数，从左到右依次挑出任意个数，它们之间无需连续，这些挑出的数构成的序列就构成了原序列的一个子序列）的长度。
>
> 可以定义一个数组`dp[]`，其元素`dp[i]`表示`以第i个楼房为结尾楼房的递增子序列中最长的那个子序列的长度`，例如：对于三个楼房`3 1 2`，有`dp[1] == 1; dp[2] == 1; dp[3] == 2;`，因为以第一个房子（高度为3）为结尾的递增子序列只有3本身一个元素，后面的高度都比3小；以第二个房子为结尾的递增子序列同理；以第三个房子为结尾的最长递增子序列为`1 2`，长度为2。
>
> 那么可以知道一个递推关系：`dp[i] = max(1, dp[j] + 1) (其中, j < i && height[j] <= height[i])`，也就是说，以第`i`个房子为结尾的最长递增子序列，可能只有它一个数，此时长度为1，也除了它自己之外，前面还有若干个数，此时，前面的这若干个数本身也是一个单调递增子序列，且一定以`i`之前的某个更低的房子为结尾。相当于：以第`i`个房子为结尾的最长递增子序列 一定

> 是 由以第`i`个房子之前的某个房子为结尾的递增子序列再添加上更高一些的第`i`个房子后构成的。所以遍历`dp[1] dp[2] dp[3] ... dp[i - 1]`，看看哪个`i`之前的最长序列的末尾房子比`i`房子低，便可以让其跟上一个`i`房子，将这些遍历后的结果取个最大值遍得到了`dp[i]`。
>
> 根据这个递推关系，便可以写循环，**依次**求出`dp[1] dp[2] dp[3] ... dp[n]`，取这`n`个数的最大值即为结果。

10 - 合并字符串

两个有序数组的合并问题，可以参考去年程设 `C6 - G - 合并多项式` 的思路：

> 注意到l两个数组都是以严格升序给出的，因此可以考虑这样一种做法：
>
> 1. 用两个变量 i, j 记录当前正在处理第一个数组的第 i 项和第二个数组的第 j 项，最初 $i = j = 1$。
> 2. 比较第一个数组的第 i 项和第二个数组的第 j 项：
>
>   a. 若第一个数组的第 i 项较小，输出，并将 i 向后移动一位；
>
>   b. 否则，输出第二个数组的第 j 项，并将 j 向后移动一位；
>
>   c. 如果某一数组中的元素都输出了，则直接输出另一个数组剩下的元素。
>
> 由于只会移动最多 $m + n$ 次，因此总循环次数不超过 $m + n$ 次，可以在题目要求的时间范围内完成计算。

当然，数据结构题目的数据都比较弱，直接把两个数组当成一个数组后排序也可以。参考代码里给出的是归并的做法。

11 - 连续正整数的和

首先，如何验证有以数字 i 开头的连续正整数，其和是 n 呢？我们可以维护一个变量 sum，从整数 i 开始，如果 $\mathrm{sum} < \mathrm{n}$，我们就加上下一个数。这样，最后 $\mathrm{sum} \geq \mathrm{n}$，如果等于 n 就说明有以数字 i 开头的连续正整数和为 n，否则没有。

第二点就是如果存在多于一种的可行方案，则选取等式右边项的个数最多的那一种，显然起始的 i 越小项就越可能更多，所以我们从小到大枚举 i 。

12 - 合数分解

这道题算是用试除法分解质因数的模板问题。其中有几点关键的结论：

- 枚举 2 到 $\sqrt{\mathrm{n}}$ 的所有数字 i，如果 n 能被 i 整数，就把 n 中的所有因子 i 除掉
- 如果对 2 到 $\sqrt{\mathrm{n}}$ 的所有数字 i 进行如上处理后，最后 n 的值不为 1，那么说明 n 有一个比 $\sqrt{\mathrm{n}}$ 大的因子，也就是这个剩下的值，输出剩下的值（显然 n 最多只有一个比 $\sqrt{\mathrm{n}}$ 大的因子，比如 $15=3\times5$）
- **为什么不用特意先预处理出素数**：如果 n 有一个合数因子 d，显然 d 的所有质数因子比 d 小，那么它们都会在处理 d 之前被除去，所以当处理的 d 时，此时的 n 不能被 d 整数

13 - 字母频率统计

首先，统计字母的出现次数可以用一个简单的哈希思想，比如用 `ha[0]` 表示字母 `a` 出现的次数，用 `ha[1]` 表示字母 `b` 出现的次数，···

得到了这些字母的出现次数后，我们要画柱状图。我们可以发现从上往下看，字母出现次数越多的，出现 `*` 就越早。所以我们可以统计出所有字母中出现次数最多的，出现次数记为 `cnt`，每次不断让 `cnt` 减小，如果某个字母的出现次数比 `cnt` 多，那就在这一行输出它。

14 - 计算公式：求π的值b

我们去年就做过类似的，给定公式求圆周率的题目，有几个要点：

- 注意数据类型，如果两个整型变量直接相除，那它们的结果还是整型变量，就算你用一个浮点变量接收，那也是将这个整型变量转换为浮点变量，这个时候已经是一个错误的结果了
- 能用 `double` 就不要用 `float`，精度太低
- 每次分子分母不用重新计算，可以用两个变量分别记录当前分子分母的结果，下一次循环的时候在这两个变量的基础上计算

15 - 文件排版（非文件）

首先虽然规则中没有说明，但是通过样例，我们可以看出需要将连续的空格和制表符合并为一个空格；假设我们将处理前读入的字符串存入 `str` 中；要把处理后 `:` 前的串存入 `str2`，处理后 `:` 后的串存入 `str3`，那么我们就可以通过遍历 `str` 的每一位，当遇到空格或者制表符时，如果当前 `str2` 或者 `str3` 的最后一位是空格（你遇到 `:` 前处理的就是 `str2`，遇到 `:` 后处理的就是 `str3`），那么我们就忽略这一

位，开始枚举下一位；否则就在 `str2` 或者 `str3` 的末尾存入空格。这样就能合并连续的空格和制表符，变成一个空格。

我们得到 `str2` 和 `str3` 后，可以对 `str2` 按照指定位对齐格式化输出后，再输出 `:` `str3`。有的时候 `:` 前面和后面可能会有空格，有的时候则没有，为了统一处理，我们把最后要求输出时加在 `:` 前面的空格单独当做一个空格输出，而不是 `str2` 的一部分，也就是说处理完后如果 `str2` 最后有空格我们就把它去掉；把 `:` 的空格作为 `str3` 的一部分，也就是说在处理 `str3` 之间赋值 `str3[0]` 为空格。

最后要处理的问题是我们虽然知道应该把 `str2` 按照 `n - 2` 位右对齐输出（-2 是因为 `:` 和空格），但是 `n` 每次是个变量。实际上我们上学期程设也遇到过类似的问题，一种解决方法是用 `sprintf` 动态地改变格式串（`sprintf` 即输出到字符串里，而非控制台上），即 `sprintf(geshi, "%-%%ds", n - 2);` ，其中负号代表右对齐。

16 - 注释比例

这道题无非是统计出一共有多少字符，多行注释内有多少字符。一共有多少字符很好统计，我们可以一个字符一个字符读入，每读入一个字符就把当前的总字符个数 + 1。那么该如何统计注释内有多少字符呢？其实跟统计一共有多少字符差不多，只不过我们需要知道当前是在多行注释内，再增加当前注释内的总字符个数 + 1。分别标记注释内的总字符个数为 `zhushi`，总的字符个数为 `total`。

假设我们定义一个变量 `sta`，当这个变量的值为 2 时表示我们在多行注释中，其它值为不在多行注释中，那么很显然当 $sta = 2$ 时，我们同时增加 `zhushi` 和 `total` 的值；当 $sta \neq 2$ 时，我们只增加 `total` 的值。

当程序开始时，我们让 `sta` 的值为 0，标记这是最一般的状态。那么什么时候有可能进入到多行注释呢？实际上，当我们在 `sta = 0` 时，遇到一个 `/` ，我们就要小心了，因为如果下一个字符是 `*`，那么我们就进入了一个多行注释。所以，当我们在状态 `0` 下，读入了一个 `/`，那么我们就可以让 `sta = 1`，标记这是一个危险的状态。当 `sta = 1` 时，如果我们新读入的字符是 `*`，就证明我们真的进入了一个多行注释，标记 `sta = 2`；否则，如果是其它字符，证明我们刚才不过是虚惊一场，可以把 `sta` 重新赋值为 0，代表我们回到了一个普通状态，等待下一个 `/` 的出现。（因为按照题意，不会有单行注释，所以我们不必担心紧跟着 `/` 的这个字符还是 `/`）。

同样，在多行注释中时，我们还要时刻关注何时多行注释结束了。当我们遇到一个 `*` 时，如果下一个字符是 `/` ，那么我们就离开多行注释了。所以当 `sta = 2` 时，当前读入字符为 `*`，我们可以让 `sta = 3`，表示准备离开多行注释的状态。当 `sta = 3`，且当前读入字符为 `/` 时，表示我们真的离开了一个多行注释，回到普通状态，重新让 `sta = 1`；否则证明我们只是虚惊一场，还是在注释内，让 `sta` 回到 2，标记

在多行注释内。不过这里要注意，我们在 `sta = 3` 时可能还读入一个 `*`，这个 `*` 仍然可能是一个多行注释结束的表示，所以当 `sta = 3` 且读入字符是 `*` 时，我们仍要保持 `sta = 3`，这是一个非常容易错的地方（第四次作业括号匹配的题目可能就有这种风险）。

这就是状态转移的思想，在第四次作业中我们可以用这种思想去处理单行注释、多行注释、字符常量、字符串常量。我们在第五道题中已经利用过了这个思想，同学们在这道题可以自己尝试画出状态转移图。

最后截断小数点后输出可以直接利用整型除法向 0 靠齐的特点，输出 `%` 别忘了转义。

另外这种题都有个易错点，就是在多行注释里（或者多行注释最后）出现一个 `*`，可能会导致状态的错误转移，同学们一定要注意。

17 - 删除子串

直接最朴素的思想，枚举原字符串的每一位，表示可能与子串匹配的起始位置。用一个变量 `i` 标记枚举到原串的位置，如果第 `i` 位不等于子串的第一位，那么显然以 `i` 开始的串不会是子串，直接把第 i 位存到答案输出数组中；否则比较一下，看看以 `i` 开始的若干位（子串长度位）是不是子串，如果是的话，那么就跳过这一段，直接让 `i` 的值增加子串长度位，下次从子串长度位之后开始枚举。

也可以用 `strstr` 查找原串中子串的位置，如果原串中没有子串，直接输出原串；否则如果找到子串的位置是 `pos`，那么就将 `pos` 之前的所有位存入答案数组，再标记查找的起始位置为 `pos + 子串长度` ，直到找不到子串，再将剩下的位存入答案数组。

给出的代码就是第二种实现方式。