实验一：线性表

实验目的

- 在熟练掌握线性表的结构及其相关的查找、插入、删除等基本操作的基础上，可以编程实现基本的线性表，特别是栈、队列及其基本操作的实现。
- 将课程的基本原理、技术和方法与实际应用相结合，训练和提高复杂问题的数据结构设计能力，及实际应用问题的解决能力。

实验环境

- 操作系统、编程语言、集成开发环境不限。
- 实验中涉及到的数据结构需自己实现，如单链表、栈及队列等。
- 编程语言中若有字符串，可直接使用，无需实现。

实验验收

- 检查解决每个问题所设计的数据结构。
- 检查解决每个问题设计的算法。
- 检查程序能否正确运行。
- 任意一行代码可以清楚解释，**严禁抄袭**。
- 第一次实验课验收，过期不候。

实验内容

Problem 1.1

目标:

给定一元多项式:

$$a(x) = a_n x^n + ... + a_1 x + a_0$$
$$b(x) = b_m x^m + ... + b_1 x + b_0$$

设计函数实现一元多项式的乘法。

输入分为三行:

- Line1：n m，即两个多项式的最高指数（中间以空格分隔），$n \geq 0, m \geq 0$。
- Line2：多项式 $a(x)$ 各指数对应的系数（中间以空格分隔），应有 $n+1$个数字。
- Line3：多项式 $b(x)$ 各指数对用的系数（中间以空格分隔），应有 $m+1$个数字。

输出为一行，以指数递减的方式依次输出乘积多项式的非零项的系数和指数，数字间以空格分隔。

说明:

- $a(x)$ 和 $b(x)$ 各项的系数都是整数。
- 若结果中存在非零的常数项，将其指数 0 一并输出。
- 不考虑 $a(x)$ 或 $b(x)$ 为 0 的情况。

样例 1:

- 输入:

```
2 3          // n=2, m=3
2 1 0        // 表示 2x^2 + x
3 2 1 0      // 表示 3x^3 + 2x^2 + x
```

- 输出：

```
6 5 7 4 4 3 1 2          // 表示 6x^5 + 7x^4 + 4x^3 + x^2
```

Problem 1.2

目标：

给定一元多项式：

$$a(x) = a_n x^n + ... + a_1 x + a_0$$
$$b(x) = b_m x^m + ... + b_1 x + b_0$$

其中 $a(x)$ 作为被除多项式，$b(x)$作为除多项式，设计函数实现一元多项式的除法。

输入分为三行：

- Line1：n m，即两个多项式的最高指数（中间以空格分隔），$n \geq 0, m \geq 0$。
- Line2：多项式 $a(x)$ 各指数对应的系数（中间以空格分隔），应有 $n+1$个数字。
- Line3：多项式 $b(x)$ 各指数对用的系数（中间以空格分隔），应有 $m+1$个数字。

输出为两行：

- 第一行为商多项式。
- 第二行为余多项式。
- 以指数递减的方式依次输出多项式的非零项的系数和指数，数字间以空格分隔。

说明：

- $a(x)$ 和 $b(x)$ 各项的系数都为整数。
- 若结果中存在常数项，将其指数 0 一并输出。
- 方便起见，除多项式最高指数项的系数为 1，即输入时按 $b_m = 1$输入。
- 不考虑 $a(x)$ 或 $b(x)$ 为 0 的情况。
- 如果某结果多项式为 0，那么输出时将该多项式表示为 `0 0`。

样例 1：

- 输入：

```
5 2              // n = 5, m = 2
6 3 3 2 0 1      // 表示 6x^5 + 3x^4 + 3x^3 + 2x^2 + 1
1 0 1            // 表示 x^2 + 1
```

- 输出：

```
6 3 3 2 -3 1 -1 0        // 表示 6x^3 + 3x^2 -3x -1
3 1 2 0                  // 表示 3x + 2
```

样例2：

- 输入：

```
2 3          // n=2, m=3
2 1 0        // 表示 2x^2 + x
3 2 1 0      // 表示 3x^3 + 2x^2 + x
```

- 输出：

```
0 0          // 表示商为0
2 2 1 1      // 表示 2x^2 + x
```

Problem 2

有没有想过计算机是如何处理表达式的？计算机在进行运算时，通常会用逆波兰式（Reverse Polish notation，RPN，或逆波兰记法）进行计算。 逆波兰式也叫后缀表达式（将运算符写在操作数之后），我们平时写 $a+b$，这是中缀表达式，写成后缀表达式就是 $ab+$。

例如 $(1 + 2) / 3 + 6 * 7 + 10$ 转化为后缀形式为

$$1\ 2\ +\ 3\ /\ 6\ 7\ *\ +\ 10\ +$$

目标：根据输入的逆波兰式，求表达式的值。

说明：

- 输入为一个字符串数组，其包含每个参与运算的整数和运算符。
- 有效的运算符仅包括 $+$，$-$，$*$，$/$ 。
- 输入的逆波兰表达式总是有效的，即表达式总会得出有效数值且不存在除数为 0 的情况。
- 整数除法仅保留整数部分。
- 操作数和结果均为小于等于$2^{31}-1$，且大于$-2^{31}-1$的整数。

样例 1：

- 输入：["2", "1", "+", "3", "*"]
- 输出：9

样例 2：

- 输入：["4", "13", "5", "/", "+"]
- 输出：6

样例3：

- 输入：["10", "6", "9", "3", "+", "-11", "*", "/", "*", "17", "+", "5", "+"]
- 输出：22

Problem 3

串中任意个连续的字符组成的子序列称为该串的子串。

目标：输入一个非空的字符串，判断它是否可以由它的一个子串重复多次构成。

说明：

- 输入的字符串只含有小写英文字母，并且长度不超过10000。
- 输出的可以是布尔类型的值，也可以是 1 或 0，其中 1 表示可以由它的一个子串重复多次构成，0 相反，能表达出意思即可。

样例1：

- 输入: "abab"
- 输出: True （或 1）

样例2：

- 输入："aba"
- 输出：False （或 0）

样例3：

- 输入："abcabcabcabc"
- 输出：True （或 1）