部分乘积

给定 `N` 个 `int` 类型的整数，计算前 `i` 个值的乘积，其中 `i` 为 `1, 2, ..., N`。

输入描述

首先是 `int` 类型的整数 `N`，接下来是 `N` 个 `int` 类型的整数，其中 `N >= 1`，保证为正数。

输出描述

对于第 `i` 个数，输出前 `i` 个数的乘积，如果结果溢出，输出 `-1` 并结束处理。

示例

示例 1

输入

```
9 1 2 3 4 5 6 7 8 9
```

输出

```
1 2 6 24 120 720 5040 40320 362880
```

示例 2

输入

```
5 512 1024 2048 4096 8192
```

输出

```
512 524288 1073741824 -1
```

特殊元素

给定若干个 `int` 类型整数，其中一个数出现了一次，其他数出现了两次，请找出出现了一次的那个数。

输入描述

`N` 个 `int` 类型整数，以空格隔开，`N >= 1`。

输出描述

输出出现了一次的那个数。

示例

示例 1

输入

```
5 5 2
```

输出

```
2
```

示例 2

输入

```
1
```

输出

```
1
```

提示

考虑输入的读取方式与位运算。

Base64 编码

给定一个文件，输出这个文件的 Base64 编码。

Base64 编码规则如下：

1. 每 3 个字节作为一组，一共是 24 个二进制位；
2. 将这 24 个二进制位分为四组，每个组 6 个二进制位；
3. 每组前面加 `00`，扩展成 32 个二进制位，即四个字节；
4. 根据下表，得到扩展后的每个字节的对应符号，这就是 Base64 的编码值。

索引	编码	索引	编码	索引	编码	索引	编码
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	D	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

如果字节数不足 3，如下处理：

- 2 个字节的情况：将这两个字节的共 16 个二进制位分成分别包含 6、6、4 个二进制位的 3 组，最后一组除了前面加 `00` 以外，后面再加 `00`. 这样得到一个三字符的 Base64 编码，再在末尾补上 `=`
  - 比如，`"Ma"` 这个字符串是两个字节，对应的 ASCII 码为 `01001101 01100001`，首先分成三组 `010011`、`010110` 和 `0001`，填充 之后得到 `00010011`、`00010110`、`00000100` 对应 Base64 编码分别为 `T`、`W`、`E`，再补上 `=`，因此 `"Ma"` 的 Base64 编码就是 `TWE=`
- 1 个字节的情况：将这一个字节的 8 个二进制位分成分别包含 6、2 个二进制位的 2 组，最后一组除了前面加 `00` 以外，后面再加 `0000`. 这样得到一个两字符的 Base64 编码，再在末尾补上 `==`
  - 比如，`"M"` 这个字符串是一个字节，对应的 ASCII 码为 `01001101`，首先分为两组 `010011` 和 `01`，填充之后得到 `00010011` 和 `00010000`，对应的 Base64 值分别为 T、Q，再补上 `==`，因此 `"M"` 的 Base64 编码就是 `TQ==`。

样例：

编码前	1								2								3							
位模式	0	0	1	1	0	0	0	1	0	0	1	1	0	0	1	0	0	0	1	1	0	0	1	
索引	12						19						8				51							
编码后	M						T						I				z							

输入描述

待编码的字符串。

输出描述

输出字符串对应的 Base64 编码。

示例

示例 1

输入

```
0123456789
```

输出

```
MDEyMzQ1Njc4OQ==
```

示例 2

输入

```
this
is
a
multi-line string
```

输出

```
dGhpcwppcwphCm11bHRpLWxpbmUgc3RyaW5nCg==
```

提示

- 使用位运算
- 不需要考虑平台差异
- 自测的输出请使用 Base64Encoder 生成

单词计数

实现如下功能：给出一段文本，输出这段文本包含的字符数、单词数（以空格和换行符为界）和行数。

输入描述

保证输入中只会出现英文字母、空格（`' '`）和换行符（`'\n'`）。输入保证每行末尾都有一个换行符。

输出描述

字符数、单词数和行数，用空格分隔。

示例

> 示例中用 `[SPC]` 表示空格，用 `[RET]` 表示换行符

示例 1

输入

```
hello[SPC]world[RET]
```

输出

```
12 2 1
```

解释

- `' '` 和 `'\n'` 也计入字符数内
- 单词以空格和换行为界，其中的两个单词分别为 `hello` 和 `world`

示例 2

输入

```
this[SPC][SPC][SPC]is[SPC]a[SPC]test[SPC]input[SPC][SPC][RET]
```

输出

```
25 5 1
```

提示

不需要考虑平台差异。

翻转数字

给定一个 `int` 类型的整数 `N`，请你将它*数字部分*翻转，正负性不变，并输出。

输入描述

输入仅包含一个 `int` 类型的整数 `N`，保证 `N` 在 `int` 类型的范围之内（即，$-2^{31} <= N < 2^{31}$）。

如果翻转后大于 `int` 最大值或小于 `int` 最小值，输出 `-1`.

输出描述

翻转后的数。

示例

示例 1

输入

```
123
```

输出

```
321
```

示例 2

输入

```
-120
```

输出

```
-21
```

示例 3

输入

```
2147483647
```

输出

```
-1
```

解释

翻转之后超过 `int` 最大值。

提示

可以使用 `<limits>` 头文件中的 `numeric_limits<int>::max()` 和 `numeric_limits<int>::min()` 获取 `int` 的最大值和最小值。