Programming Task 3

Write a program **for the TC computer, in TCL programming language** (**make sure** to have read their description in Edux, in "Materials" section), computing the square of the given number, in a way which avoids an arithmetic overflow.

The program should assume that the architecture is 32-bit, so one register can hold (using the signed convention) an integer from the range $[-2^{31}, 2^{31}-1]$. While the input number n will fit within that range, its square not necessarily so. In such case, the program should print out the square in two pieces (see the “Output” section below).

Input

The input to the program consists of a single positive integer n, lower than 10^8.

Output

If the number n^2 has 8 or less digits, the output should contain n^2 and nothing more.

Otherwise, the output should consist of two lines:

- In the first line, you should print out n^2 except for its last 8 digits;
- In the second line, you should print out the number formed from the last 8 digits of n^2 (removing the leading zeroes; see Example 3).

Examples

Example 1

For the input

```
1234
```

the output should consist of one line:

```
1522756
```

Example 2

For the input

```
12345678
```

the output should consist of two lines:

```
1524157
65279684
```

because $12345678^2 = 152415765279684$.

Example 3

For the input

```
449556
```

the output should consist of two lines:

```
2021
597136
```

because $449556^2 = 202100597136$. Since the last 8 digits of that number is 00597136, and TCL always prints positive integers without leading zeroes, we exceptionally expect only 6 digits (597136) in the second line of the output.

Bonus (20%)

You can gain **additional 20% of points** by printing **additionally** (i.e. after the output described above) all the digits of n^2, each in a separate line, in **reversed order** (this requirement is introduced here for your convenience). For example, for the input from Example 3:

```
449556
```

your program in the bonus version should print out:

```
2021
597136
6
3
1
7
9
5
0
0
1
2
0
2
```

Note that you should print out exactly **all the** digits of n^2 – don't skip any inner zeroes; also don't add any leading zeroes around the whole number.

Assumptions

- The input to the program will consist of one positive integer, lower than 10^8.
- The program will be run with the following values of the constructional parameters of the TC computer:
  - number of registers (N): 100
  - register size in bits (K): 32
  - number of data memory cells (S): 2000
  - the address of the beginning of stack (B): 1000
  - number of instruction memory cells (T): 1000

  In other words, the program will be run in the TCE emulator with the command:

  `python3 tce.py 100 32 2000 1000 1000 <your_tcl_file>`

Requirements

Your solution **must** be implemented in the **TCL** language described in the “Materials” section!

Caution: Our assembly-level TCL language has **nothing in common** with another language known more widely (e.g. in Wikipedia) under the name "**Tcl**". Do not use that other Tcl; such programs will not be accepted.

Hint

Since the result of an ordinary multiplication *n* * *n* may not fit within 1 register, one needs to perform this multiplication “piece by piece”, in more steps.

An intuitive approach here is the so-called *long multiplication* – which we all learned at school. By splitting the number to *digits* and acting digit by digit, we’re all able to compute a square of an *arbitrarily large* number, while *de facto* only touching numbers below 100 for the whole time!

That’s a good lead already; you can use just it to solve the problem, though at the cost of writing much code.

However, an even better idea is to split *n* into fragments that are not so small. Why work on single digits when a register is as large as 32 bits? Maybe it’d be simpler to use e.g. pairs of digits? (Or, in more technical terms, to perform long multiplication in a positional system of base 100, rather than the traditional decimal one?) Or maybe use even coarser fragments? I’m leaving nailing down the details to you.

(Side note: the “thick digits” trick is recommended also for those who attack the bonus version).

Final remark

In the "Materials" section in Edux, I published an **emulator for the TC computer and TCL language**. I strongly recommend using it - it should greatly help in testing your work in progress.

Submitting solutions

Solutions should be sent by **June 21** to the address:
dominika.pawlik+tak21l+prog3@pjwstk.edu.pl .