Important message on plagiarism

The single most important point for you to realize before the beginning of your studies at ShanghaiTech is the meaning of “plagiarism”:

Plagiarism is the practice of taking someone else's work or ideas and passing them off as one's own. It is the misrepresentation of the work of another as your own. It is academic theft; a serious infraction of a University honor code, and the latter is your responsibility to uphold. Instances of plagiarism or any other cheating will be reported to the university leadership, and will have serious consequences. Avoiding any form of plagiarism is in your own interest. If you plagiarize and it is unveiled at a later stage only, it will not only reflect badly on the university, but also on your image/career opportunities.

Plagiarism is academic misconduct, and we take it very serious at ShanghaiTech. In the past we have had lots of problems related to plagiarism especially with newly arriving students, so it is important to get this right upfront:

You may…

• … discuss with your peers about course material.
• … discuss generally about the programming language, some features, or abstract lines of code. As long as it is not directly related to any homework, but formulated in a general, abstract way, such discussion is acceptable.
• … share test cases with each other.
• … help each other with setting up the development environment etc.

You may not …

• … read, possess, copy or submit the solution code of anyone else (including people outside this course or university)!
• … receive direct help from someone else (i.e. a direct communication of some lines of code, no matter if it is visual, verbal, or written)!
• … give direct help to someone else. Helping one of your peers by letting him read your code or communicating even just part of the solution in written or in verbal form will have equal consequences.
• … gain access to another one’s account, no matter if with or without permission.
• … give your account access to another student. It is your responsibility to keep your account safe, always log out, and choose a safe password. Do not just share access to your computer with other students without prior lock--out and disabling of automatic login functionality. Do not just leave your computer on without a lock even if it is just for the sake of a 5--minute break.
• … work in teams. You may meet to discuss generally about the material, but any work on the homework is to be done individually and in privacy. Remember, you may not allow anyone to even just read your source code.

With the Internet, "paste", and "share" are easy operations. Don’t think that it is easy to hide and that we will not find you, we have just as easy to use, fully automatic and intelligent tools that will identify any potential cases of plagiarism. And do not think that being the original author will make any difference. Sharing an original solution with others is just as unethical as using someone else’s work.

CS100 Homework 3 (Spring, 2021)

In this homework, you are required to do more practices with C programming language, starting to get familiar with the concepts of points, function, array and character strings, etc. This homework tries to pave the way for your later homework and projects. Have fun and enjoy coding. Practice, practice and practice.

Submission deadline:

2021-03-31 23:59:59

Late submission will open for 24 hours when the deadline is reached, with -50% point deduction.

Problem 1. Traffic Restriction (50 points)

There are just too many cars in China. Many cities suffer from heavy traffic and severe pollution. As you may know, many local governments are forcing policies on traffic. Shanghai, for example, restricts cars with non-Shanghai plates from driving on highways during rush hours.

However, a more widely adopted way of traffic restriction is based on digits in car plate numbers. Whether the last digit of your car plate number is odd or even decides which day you can drive your car on, as is the case in Gezi Wang's hometown, Gugu city. **Your task for this problem is to help Gezi Wang decide whether a car is allowed to drive on a given day.**

Let's look a little bit into car plate numbers first. A car plate number, for example, "沪 A • R3061", is composed of a Chinese character "沪" , a provincial abbreviation; a letter "A", specifying the city in province; and the rest five digits and letters "R3061". Since the first two parts are the same for every domestic car in Gugu city, we can focus only on the final part, the five digits or letters.

There are certainly rules[1] for a plate number to be valid, or one can easily make a fake plate with any number. Specifically, these are rules for plate numbers in Gugu city:

0. The letters 'I' and 'O' look too similar to digits '1' and '0', so these two letters are not allowed. For the rest of the problem, when the word "letter" is mentioned, it always means a letter excluding 'I' and 'O'.
1. **Auto-generated numbers**
   These numbers are generated in order at local DMV (department for motors and vehicles). One may get a random plate number after purchasing a new car.

   An auto-generated number is composed of either all five digits, a letter at the first place, followed by four digits, or four digits and then a letter at the last place.

   *Taxis: The number for a taxi belongs to this type. It always begins with a letter 'T' or 'X', and the rest four places are digits.

   Examples for this type: 30281, P2347, 6559X, T6280 (a taxi).

[1] Rules may differ in different cities, but are generally similar. Please stick to the scenario in this problem, rather than try to find a car whose number doesn't fit these rules and raise a question on piazza.

2. **Self-designed numbers**
   More people prefer to get a personalized plate number this way, as they can choose the combination of numbers. The rules, however, are more complex:

   A self-designed number could contain two letters and three digits. However, the two letters must be consecutive, or respectively lie on the first and last place.

   It could also contain one letter and four digits. In this case, it's only acceptable if the letter lies on the second or third place among all five places.

   However, when people can design their numbers, it's not surprising that they prefer "lucky numbers" that contain three straight digits, like "888" and "666". Unfortunately, one is not allowed to choose such plate numbers. The DMV says there cannot be three consecutive same digits in self-designed numbers.

   Examples for this type: 355BG, P996M, 88RM8, 2T001, 66C77.

3. **New energy cars**
   New energy cars are qualified for a green-colored plate, with a special number that contains 6 places, rather than 5. It either begins or ends with a 'D' or 'F', and the rest five places should all be digits.

   Examples for this type: D38324, 10795F.

Back to the scenario. The traffic restriction rule in Gugu city is that, on a **weekday**, cars whose **last digit** of plate number is **odd** can only drive if the **date** is an **odd** number, and cars whose last digit is **even** can only drive if the **date** is **even**. This rule does not apply to new energy cars or taxis.

Input description:
The input contains **3 lines**.

Each line contains three strings, separated by space.

The first string indicates a car plate number. It is guaranteed to only contain digits and uppercase letters (may contain 'I' and 'O'), but it may not be valid. The length of this string will not exceed 10 characters.

The second string is a date, in MM/DD format. (2 digits for month / 2 digits for date)

The third string is a day in the week in 3-letter abbreviation, with the first letter capitalized. Specifically, one in {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"}.

Output description:
Your output should contain **3 lines**. Printing one line after each line of input or printing all 3 lines together in the end are both OK.

For each line of input, you should first verify if the plate number is valid. If it is not, you should print:

"This number is invalid.", with a new line ('\n').

If the plate number is valid, you should then judge if it is allowed to drive, according to the restriction rules. You should print one of the following:

"This car is allowed to drive.", with a new line ('\n').
"This car is not allowed to drive.", with a new line ('\n').

Sample input:

```
3U306 03/27 Sat
T0378 10/21 Thu
N05M8 05/03 Mon
```

Sample output:

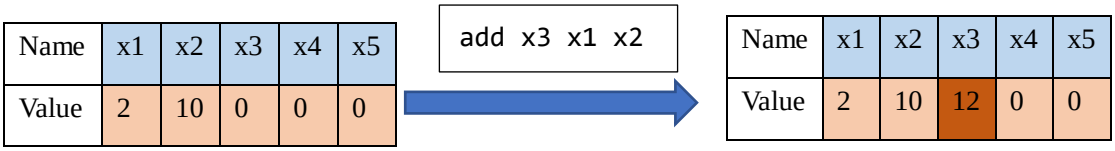
```
This car is allowed to drive.
This car is allowed to drive.
This number is invalid.
```

Problem 2. Mini emulator (50 points)

This is the third week of your programming course, I think you are more familiar with coding, debugging and testing in C. Now, let's think more about how does your program been executed (instead of: write code on OJ and submit – debug – ask classmates and TAs for help – give up).

To realize how does the computer execute your program, let me give a short introduction to CPU. The CPU basically contains 2 parts: **registers** (https://en.wikipedia.org/wiki/Processor_register) and **ALU**(https://en.wikipedia.org/wiki/Arithmetic_logic_unit). You can consider registers as an array, each cell of this array has a name (for example, x0, x1, x2, …, x31 in RISC-V), and can store a single number; and arithmetic calculations (addition, subscription, multiplication, division, …) are done by ALU.

When executing your program, the assembly codes (written in binary) are sent to CPU, each line of assembly code specifies some operations of registers, and CPU will use a special register named **PC** (program counter, https://en.wikipedia.org/wiki/Program_counter) to identify the current execution line number. For example, if the code is "`add x3 x1 x2`", then CPU will execute "value in register x3 = value in register x1 + value in register x2", which is shown in Fig.1 (only 5 registers shown here), and add PC by 1 to execute next instruction.



Values in registers: before Values in registers: after

Fig 1. Instruction execution in CPU

In this problem, you are required to implement a toy emulator which are able to execute a very simple assembly language (similar to RISC-V, https://en.wikipedia.org/wiki/RISC-V). Your only need to consider 5 registers, named `x1, x2, x3, x4` and `x5`, all in **lower case**, and each register contains a single integer (Hint: you can use an array of int with length = 5 to simulate it). Also, there are only has 7 instructions in this assembly language: `add, sub, mul, div, let, print, bge`, which are introduced below.

The instructions `add, sub, mul, div` are similar, the syntax is shown in Table 1.

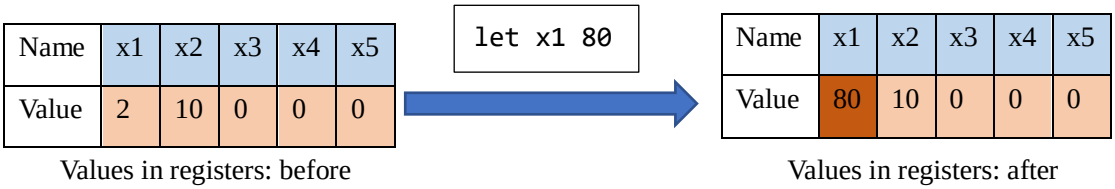
Instruction	Syntax	Explanation	Example
add	add r1 r2 r3	value in r1 = value in r2 + value in r3	add x1 x2 x3
sub	sub r1 r2 r3	value in r1 = value in r2 - value in r3	sub x1 x2 x3
mul	mul r1 r2 r3	value in r1 = value in r2 * value in r3	mul x1 x2 x3
div	div r1 r2 r3	value in r1 = value in r2 / value in r3	div x1 x2 x3

Table 1. add, sub, mul, div instructions

The "`let`" instruction is used to assign value to a register. The syntax is:

```
let reg value
```

Where "`reg`" is name of some register (x1 to x5), and "`value`" is a decimal integer between 1 and 1000. For example, the execution of instruction "`let x1 80`" is shown in Fig. 2.



Values in registers: before

Values in registers: after

Fig 2. Execution of "let x1 80"

Also, we need a "print" instruction to print the value in some register such that we can test whether the execution result is correct. Syntax of print instruction is shown below:

```
print reg
```

For example, suppose the value in register x1 is 5, when instruction "print x1" has been executed, your emulator should print "`x1 = 5`" on the screen. For your convivence, you can just copy and modify this line to print the value of `xi`:

```
printf("x%d = %d\n", i, registers[i]);
```

The "`bge`" (branch greater or equal) instruction is a bit more complex; it's used for branch comparison. The syntax is:

```
bge r1 r2 line_no
```

The following pseudo-code shows the function of this instruction.

```
if (value in register r1 ≥ value in register r2):
    PC = line_no  # jump to line number = line_no
else:
    PC += 1       # execute the next instruction
```

That is, if the value in `r1` greater or equal than which in `r2`, your emulator should jump to "`line_no`" (set PC to `line_no`) and execute from this line (including this line). The following example shows how "`bge`" works.

```
1   let x1 2
2   let x2 3
3   bge x1 x2 5
4   print x1
5   print x2
```

Fig 4. bge - nojump

```
1   let x1 2
2   let x2 3
3   bge x2 x1 5
4   print x1
5   print x2
```

Fig 5. bge - jump

```
x1 = 2
x2 = 3
```

Output of Fig. 4

```
x2 = 3
```

Output of Fig. 5

Notes:

(1) The line number of codes starts from 1, not 0, and the range of "`line_no`" is guaranteed to be valid in all testcases (`1 ≤ line_no ≤ total_lines`)

(2) The "`div`" instruction will round down the result division result (i.e., 7 / 3 = 2), and the denominator will always be non-zero.

(3) Your emulator should exit if and only if the last line of code in executed, and no more code you need to execute (PC = last line number + 1).

(4) All of these instructions are all written in **lower case**, the operation keyword and register names are spilt by a **single space**. Each line will only contain one instruction.

(5) Value in all registers should be initialized to **0** before execution.

(6) There will not be any infinite loops.

Input description:

The first line of the input is a number `n`, indicates the total lines of code. The following `n` lines are the assembly codes. You can refer to "notes" for more details. It's guaranteed that all testcases are within 20 lines, and the immediate number in "`let`" instruction is between 0 and 1000. Also, no overflows and underflows will occur in all testcases, use `int` is enough. All inputs are guaranteed to be legal (i.e., only the mentioned instructions will occur, one line per instruction, all in lower case, split by single space and registers will be either x1, x2, x3, x4 or x5). No empty testcases will occur.

Output description:

Your emulator should execute the input program, and only print the result when a "`print`" instruction is executed. It's guaranteed that the input and output are both non-empty.

A correct program runs like this: (red indicates input and black indicates your output)

(Simple):

```
5
let x1 20
let x2 50
add x3 x1 x2
mul x4 x3 x1
print x4
x4 = 1400
```

```
4
let x3 15
let x2 4
div x1 x3 x2
print x1
x1 = 3
```

(Loop):

```
8
let x1 4
let x2 1
let x3 6
let x5 1
mul x4 x3 x2
print x4
add x2 x2 x5
bge x1 x2 5
x4 = 6
x4 = 12
x4 = 18
x4 = 24
```

Acknowledgements: Thanks to Cunhan You's hard work on verification and testing for this homework.