

M&Ms partners



How many M&Ms do you think are in there?

You are going to implement a program that first picks a random number between 0 and 1023 inclusive, and then asks you to guess what it is. The program should keep asking you to guess until you guess right at which point it should congratulate you for guessing the correct number.

Output

The output of your program is totally up to you, but here is one possible design. Assume that the boldfaced text is what some user has typed in.

I am thinking of a number between 0 and 1023. What is it?

0

Too low. Guess again.

512

Too High. Guess again.

256

...

Too High. Guess again.

430

Too High. Guess again.

428

You guessed it!

Credit Cards

Odds are you have a credit card in your wallet. Though perhaps the bill does not (yet) get sent to you! That card has a number, both printed on its face and embedded (perhaps with some other data) in the magnetic stripe on back. That number is also stored in a database somewhere, so that when your card is used to buy something, the creditor knows whom to bill. There are a lot of people with credit cards in this world, so those numbers are pretty long:

American Express uses 15-digit numbers, MasterCard uses 16-digit numbers, and Visa uses 13- and 16-digit numbers. And those are decimal numbers (0 through 9), not binary, which means, for instance, that American Express could print as many as $10^{15} = 1,000,000,000,000,000$ unique cards!

Now that's a bit of an exaggeration, because credit card numbers actually have some structure to them. American Express numbers all start with 34 or 37; MasterCard numbers all start with 51, 52, 53, 54, or 55; and Visa numbers all start with 4. But credit card numbers also have a "checksum" built into them, a mathematical relationship between at least one number and others. That checksum enables computers (or humans who like math) to detect typos (e.g., transpositions), if not fraudulent numbers, without having to query a database, which can be slow. (Consider the awkward silence you may have experienced at some point whilst paying by credit card at a store whose computer uses a dial-up modem to verify your card.) Of course, a dishonest mathematician could certainly craft a fake number that nonetheless respects the mathematical constraint, so a database lookup is still necessary for more rigorous checks. So what's the secret formula? Well, most cards use an algorithm invented by Hans Peter Luhn, a nice fellow from IBM. According to Luhn's algorithm, you can determine if a credit card number is (syntactically) valid as follows:

Multiply every other digit by 2, starting with the number's second-to-last digit, and then add those products' digits together.

Add the sum to the sum of the digits that weren't multiplied by 2.

If the total's last digit is 0 (or, put more formally, if the total modulo 10 is congruent to 0), the number is valid!

That's kind of confusing, so let's try an example with this AmEx:

378282246310005

For the sake of discussion, let's first underline every other digit, starting with the number's second-to-last digit:

378282246310005

Okay, let's multiply each of the underlined digits by 2:

$$7*2 + 2*2 + 2*2 + 4*2 + 3*2 + 0*2 + 0*2$$

That gives us:

$$14 + 4 + 4 + 8 + 6 + 0 + 0$$

Now let's add those products' digits (i.e., not the products themselves) together:

$$1 + 4 + 4 + 4 + 8 + 6 + 0 + 0 = 27$$

Now let's add that sum (27) to the sum of the digits that weren't multiplied by 2:

$$27 + 3 + 8 + 8 + 2 + 6 + 1 + 0 + 5 = 60$$

Yup, the last digit in that sum (60) is a 0, so the card is legit!

So, validating credit card numbers isn't hard, but it does get a bit tedious by hand. Let's write a program.

Write a program that prompts the user for a credit card number and then reports whether it is a valid American Express, MasterCard, or Visa card number, per the definitions of each's format herein.

So that we can automate some tests of your code, we ask that your program's last line of output be

AMEX or
MASTERCARD or
VISA or

INVALID

nothing more, nothing less. For simplicity, you may assume that the user's input will be entirely numeric (i.e., devoid of hyphens, as might be printed on an actual card).

Here are a few examples (user input is in bold):

Number: 378282246310005
AMEX

Number: 6175230925
INVALID

Test out your program with a whole bunch of inputs, both valid and invalid. (We certainly will!)

Here are a few card numbers that PayPal recommends for testing:

https://www.paypalobjects.com/en_US/vhelp/paypalmanager_help/credit_card_numbers.htm

Google (or perhaps a roommate's wallet) should turn up more.