

# Project A

## Credit

A credit (or debit) card, of course, is a plastic card with which you can pay for goods and services. Printed on that card is a number that's 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!

Actually, that's a bit of an exaggeration, because credit card numbers actually have some structure to them. All American Express numbers start with 34 or 37; most MasterCard numbers start with 51, 52, 53, 54, or 55 (they also have some other potential starting numbers which we won't concern ourselves with for this problem); and all Visa numbers 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. 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.

## Luhn's Algorithm

So what's the secret formula? Well, most cards use an algorithm invented by Hans Peter Luhn of 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 Sidharth's Visa: 4003600000000014.

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

4003600000000014

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

$1 \cdot 2 + 0 \cdot 2 + 0 \cdot 2 + 0 \cdot 2 + 0 \cdot 2 + 6 \cdot 2 + 0 \cdot 2 + 4 \cdot 2$

That gives us:

$2 + 0 + 0 + 0 + 0 + 12 + 0 + 8$

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

$2 + 0 + 0 + 0 + 0 + 1 + 2 + 0 + 8 = 13$

Now let's add that sum (13) to the sum of the digits that weren't multiplied by 2 (starting from the end):

$13 + 4 + 0 + 0 + 0 + 0 + 0 + 3 + 0 = 20$

Yup, the last digit in that sum (20) is a 0, so Sidharth's 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.

## Implementation Details

Write a program that prompts the user for a credit card number and then reports (via printf) 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\n or MASTERCARD\n or VISA\n or INVALID\n, 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). But do not assume that the user's input will fit in an int!

Consider the below representative of how your own program should behave when passed a valid credit card number (sans hyphens).

./credit

Number: 4003600000000014

VISA

./credit

Number: 4003-6000-0000-0014

Number: foo

Number: 4003600000000014

VISA

./credit

Number: 6176292929

INVALID