Table of Contents

Credit

tl;dr

Implement a program that determines whether a provided credit card number is valid according to Luhn's algorithm.

```
$ ./credit
Number: 378282246310005
AMEX
```

Background

Odds are you or someone you know has a credit card. 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! (That's, ahem, a quadrillion.)

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 start with 51, 52, 53, 54, or 55 (technically, they also have some other potential starting numbers which we won’t concern ourselves with for this problem); 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:

0. Multiply every other digit by 2, starting with the number’s second-to-last digit, and then add those products' digits together.
1. Add the sum to the sum of the digits that weren’t multiplied by 2.
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 my AmEx: 378282246310005.

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

   3<u>7</u>8<u>2</u>8<u>2</u>2<u>4</u>6<u>3</u>1<u>0</u>0<u>0</u>5

   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

1. 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

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

So, validating credit card numbers isn’t hard, but it does get a bit tedious by hand.

Specification

- In `credit.c` in `~/workspace/pset1/credit/`, 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, and that `main` always return `0`.
- 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).
- Do not assume that the user’s input will fit in an `int`! Best to use `get_long_long` from CS50’s library to get users' input. (Why?)

Walkthrough

Usage

Your program should behave per the example below. Assumed that the underlined text is what some user has typed.

```
$ ./credit
Number: 378282246310005
AMEX
```

```
$ ./credit
Number: 3782-822-463-10005
Number: foo
Number: 378282246310005
AMEX
```

```
$ ./credit
Number: 6176292929
INVALID
```

Testing

Correctness

```
check50 cs50/2018/x/credit
```

Style

```
style50 credit.c
```

Staff Solution

```
~cs50/hacker1/credit
```

Hints

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://developer.paypal.com/docs/classic/payflow/payflow-pro/payflow-pro-testing/#credit-card-numbers-for-testing (https://developer.paypal.com/docs/classic/payflow/payflow-pro/payflow-pro-testing/#credit-card-numbers-for-testing)

Google (or perhaps a roommate's wallet) should turn up more. (If your roommate asks what you're doing, don't mention us.) If your program behaves incorrectly on some inputs (or doesn't compile at all), time to debug!