Credit

Um cartão de crédito (ou débito), é claro, é um cartão de plástico com o qual você pode pagar por bens e serviços. Impresso nesse cartão está um número que também é armazenado em um banco de dados em algum lugar, de modo que quando seu cartão for usado para comprar algo, o credor saiba a quem cobrar. Há muitas pessoas com cartões de crédito no mundo, então esses números são bem longos: A American Express usa números de 15 dígitos, MasterCard usa números de 16 dígitos e Visa usa números de 13 e 16 dígitos. E esses são números decimais (0 a 9), não binários, o que significa, por exemplo, que a American Express poderia imprimir até 10^15 = 1.000.000.000.000.000 de cartões exclusivos! (Isso é, hum, um quatrilhão.)

Na verdade, isso é um pouco exagerado, porque os números de cartão de crédito realmente têm alguma estrutura. Todos os números American Express começam com 34 ou 37; a maioria dos números MasterCard começa com 51, 52, 53, 54 ou 55 (eles também têm alguns outros números iniciais com os quais não vamos nos preocupar neste problema); e todos os números Visa começam com 4. Mas os números de cartão de crédito também têm uma "soma de verificação (*checksum)*" embutida, uma relação matemática entre pelo menos um número e outros. Essa soma de verificação permite que computadores (ou humanos que gostam de matemática) detectem erros de digitação (por exemplo, transposições), ou números fraudulentos, sem ter que consultar um banco de dados, o que pode ser lento. É claro que um matemático desonesto certamente poderia criar um número falso que, no entanto, respeite a restrição matemática, portanto, uma consulta ao banco de dados ainda é necessária para verificações mais rigorosas.

Algoritmo de Luhhn

Então, qual é a fórmula secreta? Bem, a maioria dos cartões usa um algoritmo inventado por Hans Peter Luhn, da IBM. De acordo com o algoritmo de Luhn, você pode determinar se um número de cartão de crédito é (sintaticamente) válido da seguinte maneira:

1. Multiplique todos os dígitos por 2, começando com o penúltimo dígito do número (pegando número sim, número não), em seguida, some os dígitos desses produtos.
2. Adicione a soma à soma dos dígitos que não foram multiplicados por 2.
3. Se o último dígito da soma for 0 (ou, mais formalmente, se o resto da divisão por 10 for igual a 0), o número é válido!

Isso é meio confuso, então vamos tentar um exemplo com o número Visa: 4003600000000014.

1. Para fins de exemplo, vamos primeiro destacar todos os outros dígitos, começando com o penúltimo dígito do número:

```
4003600000000014
```

destacamos

```
4   0   6   0   0   0   0   1
```

que multiplicando por 2 e somando tudo

```
2x4 + 2x0 + 2x6 + 2x0 + 2x0 + 2x0 + 2x0 + 2x1
```

que resulta em `13`.

2. Agora, somamos 13 com a soma dos outros números que não foram multiplicados por 2.

```
13 + 4 + 0 + 0 + 0 + 0 + 0 + 3 + 0 = 20
```

3. Ótimo, o último dígito da soma anterior é zero, então o cartão tem um **número válido**.

Detalhes de implementação

Em um arquivo chamado `credit.c` no diretório `~/pset1/credit/`, escreva um programa que solicite ao usuário um número de cartão de crédito e, em seguida, informe (via `printf`) se é um número de cartão `American Express`, `MasterCard` ou `Visa` válido , conforme as definições de formato de cada um aqui. Para que possamos automatizar alguns testes do seu código, a última linha de saída do seu programa seja `AMEX\n` ou `MASTERCARD\n` ou `VISA\n` ou `INVALID\n`, nada mais, nada menos. Para simplificar, você pode assumir que a entrada do usuário será totalmente numérica (ou seja, sem hífens, tal como impresso em um cartão real). Mas não presuma que a entrada do usuário caberá em um `int`! Melhor usar `get_long` da biblioteca do CS50 para obter a entrada dos usuários. (Por quê?)

Considere os exemplos abaixo de como seu próprio programa deve se comportar ao receber um número de cartão de crédito válido (sem hífens).

```
$ ./credit
Number: 4003600000000014
VISA
```

Como `get_long` rejeita hifens e outros caracteres, temos:

```
$ ./credit
Number: 4003-6000-0000-0014
Number: foo
Number: 4003600000000014
VISA
```

Mas depende de você verificar entradas que não sejam números de cartão de crédito (por exemplo, um número de telefone), mesmo que seja numérico:

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

Teste seu programa com um monte de entradas, válidas e inválidas. (Certamente o faremos!) Aqui estão alguns números de cartão que o PayPal recomenda para teste.

Se o seu programa se comporta incorretamente em algumas entradas (ou não compila), é hora de depurar!

Teste e Verifique seu código

Execute o comando `check50` como abaixo

```
check50 cs50/problems/2020/x/credit
```

Para verificar se seu código está devidamente **estilizado**, utilize o comando `style50`

```
style50 credit.c
```

Note que devemos informar o nome do arquivo de código-fonte.

Informações adicionais e OBRIGATÓRIAS

As primeiras linhas do seu código devem consistir de um comentário de várias linhas contendo sua matrícula, seu nome completo e seu nome de usuário do github como no exemplo abaixo.

```
MATRÍCULA: ................
NOME: ..................
USUÁRIO: ...............
```

substitua os pontos com suas informações.

Enviando seu programa

Uma vez que você tenha verificado o funcionamento, `check50` e a estilização do código, `style50`, execute o comando abaixo, logando com sua conta do GitHub através de **nome de usuário** e **senha**. Por questões de segurança, você verá asteríscos (*) em vez dos caracteres da sua senha.