## Exercício 4: Crédito (desafio))

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 o 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: 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 têm alguma estrutura. Todos os números American Express começam com 34 ou 37; a maioria dos números do MasterCard começa com 51, 52, 53, 54 ou 55 (eles também têm alguns outros números iniciais potenciais com os quais não nos preocupamos neste problema); e todos os números Visa começam com 4. Mas os números de cartão de crédito também têm um “checksum” embutido, uma relação matemática entre pelo menos um número e outros. Essa soma de verificação permite que os computadores (ou humanos que gostam de matemática) detectem erros de digitação (por exemplo, transposições), se não números fraudulentos, sem ter que consultar um banco de dados, 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 pesquisa no banco de dados ainda é necessária para verificações mais rigorosas.

## Algoritmo de Luhn

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 cada segundo digito por 2, começando com o penúltimo dígito do número e, em seguida, some os dígitos desses produtos.
2. Adicione essa soma à soma dos dígitos que não foram multiplicados por 2.
3. Se o último dígito do total for 0 (ou, mais formalmente, se o módulo total 10 for congruente com 0), o número é válido!

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

****1- Para fins de discussão, vamos primeiro sublinhar todos os outros dígitos, começando com o penúltimo dígito do número:****

4003600000000014

Ok, vamos multiplicar cada um dos dígitos sublinhados por 2:

1 • 2 + 0 • 2 + 0 • 2 + 0 • 2 + 0 • 2 + 6 • 2 + 0 • 2 + 4 • 2

Isso nos dá:

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

Agora vamos adicionar os dígitos desses produtos (ou seja, não os próprios produtos):

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

****2- Agora vamos adicionar essa soma (13) à soma dos dígitos que não foram multiplicados por 2 (começando do final):****

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

****3- Sim, o último dígito dessa soma (20) é 0, então o cartão de David é legítimo!****

Portanto, validar números de cartão de crédito não é difícil, mas se torna um pouco tedioso manualmente. Vamos escrever um programa.

## ****Detalhes de Implementação****

## Este vídeo irá te ajudar a entender o problema ;)

****Atenção:**** para adicionar legendas ao vídeo clique no botão CC localizado no Player e selecione a opção "Português (Brasil)".

Em um arquivo chamado ****credit.c**** em um 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 , de acordo com as definições de formato de cada um neste documento. Para que possamos automatizar alguns testes do seu código, pedimos que 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 o input do usuário será inteiramente numérica (ou seja, sem hífens, como pode ser impresso em um cartão real). Mas não presuma que o input do usuário caberá em um ****int**** ! Melhor usar ****get\_long**** da biblioteca do CS50 para obter o input dos usuários. (Por que?)

Considere o seguinte exemplo de como seu próprio programa deve se comportar quando um número de cartão de crédito válido é fornecido (sem hífens).

$ ./credit  
Número: 4003600000000014  
VISA

Agora, ****get\_long**** em si rejeitará hífens (e mais) de qualquer maneira:

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

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

$ ./credit  
Número: 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 com alguns inputs(ou não compila), é hora de depurar!

## **Como testar seu código no IDE do CS50?**

Execute o seguinte para avaliar se seu código está correto usando **check50**. Mas certifique-se de compilar e testar você mesmo!

check50 cs50/problems/2021/x/credit

Execute o seguinte para avaliar o style do seu código usando ****style50****.

style50 credit.c