Problem Set 1: Credit

David J. Malan, for Harvard CS50

Data de Entrega: 22/08/2022

Atenção / Attention

Português: Este documento é uma tradução não oficial e, portanto, não aprovada pelos autores originais, da tarefa de programação mais difícil a ser cumprida pelos alunos na primeira semana do curso Harvard CS50 (o curso de introdução à Ciência da Computação que Harvard oferece para alunos que nunca estudaram computação anteriormente). O intuito desta tradução é apenas acadêmico, proporcionando aos estudantes brasileiros que não são fluentes em inglês o mesmo desafio de programação disponível publicamente na internet (no site da disciplina CS50). Todo o crédito por este desafio de programação vai para os autores originais e o staff da disciplina CS50.

English: This document is an unofficial translation, and therefore not approved by the original authors, of the most difficult programming assignment that students will have to accomplish in the first week of the Harvard CS50 course (the introductory Computer Science course thar Harvard offers for students who have never studied computing before). The purpose of this translation is academic only, providing Brazilian students who are not fluent in English, the same programming challenge publicly available on the Internet (on the CS50 course website). All credit for this programming challenge goes to the original authors and staff of the CS50 discipline.

Sumário

1 Explicações básicas 2

2 Cartões de crédito 3

3 Algoritmo de Luhn 3

4 Detalhes para a implementação 4

5 Como enviar seu código 5

1 Explicações básicas

Durante a *Semana 1*[1] do curso de introdução à Ciência da Computação da Harvard University, o Harvard CS50[2], todos os alunos devem entregar 3 tarefas de programação utilizando a linguagem **C**:

- A primeira tarefa é um simples "Olá, mundo!";
- A segunda tarefa é imprimir uma pirâmide de blocos utilizando "#", inspirada no jogo Super Mario Brothers, da Nintendo. Os alunos podem optar por uma versão mais simples ou uma versão um pouco mais complexa da mesma tarefa.
- Na terceira tarefa os alunos podem novamente optar por uma dentre duas opções: a tarefa menos complexa é criar um algoritmo guloso para determinar a quantidade mínima de moedas que um caixa eletrônico deve dar como troco em uma compra, e a tarefa mais complexa é implementar o algoritmo de Luhn para a validação de números de cartões de créditos.

Este documento é uma tradução da tarefa de programação mais complexa dessa semana, a implementação do algoritmo de Luhn para a validação de números de cartões de crédito. Para conhecer os outros desafios, visite a página de tarefas da Semana 1[3], no site da CS50.

Se você se interessar, pode acompanhar gratuitamente o curso se inscrevendo em <https://cs50.harvard.edu/x/2022/>. Se preferir, pode acompanhar o curso no site da edX e obter um certificado verificado de Harvard em <https://www.edx.org/course/introduction-computer-science-harvardx-cs50x>.

Note que os alunos da CS50 utilizam a linguagem **C**, aprendendo com os recursos disponíveis online[4]. Para este PSET você poderá usar a linguagem **C** ou a linguagem **Python**, mas atenção:

- Se você optar por realizar o PSET usando a linguagem C:
  - Certifique-se de seguir o **Guia de Estilo para C** disponibilizado pela CS50, em <https://cs50.readthedocs.io/style/c/>.
  - Para facilitar as operações de entrada de dados com strings, utilize a biblioteca `cs50.h`. Maiores informações em <https://cs50.readthedocs.io/libraries/cs50/c/>.
- Se você optar por realizar o PSET usando a linguagem Python:
  - Certifique-se de seguir o **PEP-8: Style Guide for Python**, disponibilizado em <https://peps.python.org/pep-0008/>.
  - Para facilitar as operações de entrada de dados, talvez você queira utilizar a biblioteca `cs50.py`. Maiores informações em <https://cs50.readthedocs.io/libraries/cs50/python/>.
- **Leia todo o documento ANTES de começar a programar!**

[1] Note que cientistas da computação começam a contar a partir do 0 (zero), portanto a *Semana 1* é, na realidade, a segunda semana do curso. As tarefas da *Semana 0* são realizadas com o uso do Scratch, uma linguagem visual de programação desenvolvida pelo MIT e disponível gratuitamente para qualquer pessoa em <https://scratch.mit.edu/>.

[2] <https://cs50.harvard.edu/x/2022/>

[3] <https://cs50.harvard.edu/x/2022/psets/1/>

[4] <https://cs50.harvard.edu/x/2022/weeks/1/>

- **NÃO COPIE CÓDIGO** da internet ou de outras fontes! O objetivo deste desafio é que você pense e dê o seu melhor na implementação do algoritmo. Ao copiar código da internet ou de seus colegas, sem se esforçar e resolver problemas e bugs em seu próprio código, você estará apenas *enganando a si mesmo*.
- O prazo original que os alunos de Harvard têm para resolver essa tarefa é 1 semana. Tente conseguir nesse prazo!
- Divirta-se e aproveite!

2 Cartões de crédito

Um cartão de crédito (ou débito), obviamente, é um cartão plástico com o qual você pode pagar por bens e serviços. O cartão contém um número impresso que também está armazenado em um banco de dados em algum lugar de forma que, quando você usa o cartão para comprar algo, o emissor do cartão saiba para quem mandar a cobrança.

Existem muitas e muitas pessoas que possuem cartões de crédito no mundo, então os números dos cartões são bem longos: os cartões da American Express utilizam números com 15 dígitos, os da MasterCard utilizam números com 16 dígitos, e os cartões da Visa utilizam números com 13 ou 16 dígitos. Todos os dígitos são decimais (0 a 9), o que significa que a American Express poderia imprimir $10^{15} = 1\,000\,000\,000\,000\,000$ cartões únicos (ou seja: um quadrilhão de cartões).

Na verdade isso é um pouco de exagero porque os números dos cartões de crédito obedecem a uma certa estrutura interna. Todos os cartões da American Express começam com 34 ou 37; a maioria dos cartões da MasterCard começam com 51, 52, 53, 54 ou 55; e todos os cartões da Visa começam com 4.

Além disso todos os números de cartões de crédito tem um *checksum*[5] incorporado, uma relação matemática entre pelo menos um dos dígitos e os outros. Esse checksum permite que computadores (ou humanos que gostem de matemática) possam detectar erros (por exemplo, transposições) e até números fraudulentos, sem ter que consultar um banco de dados remoto, o que seria lento. Claro que um matemático desonesto poderia certamente criar um número falso com um checksum correto, então uma consulta a um banco de dados remoto ainda assim é necessária para verificações de segurança mais rigorosas.

3 Algoritmo de Luhn

Qual é a fórmula secreta desse checksum? Bem, a maioria dos cartões de crédito utiliza um algoritmo inventado por **Hans Peter Luhn**[6], da IBM. De acordo com o algoritmo de Luhn você pode dizer se um número de cartão de crédito é válido sintaticamente do seguinte modo:

1. Começando pelo penúltimo dígito e indo em direção ao primeiro, multiplique cada dígito sim, dígito não, por 2. Depois some *os dígitos* desses resultados (atenção: não é para somar os resultados, é para somar os dígitos dos resultados);

[5] Soma de verificação

[6] <https://en.wikipedia.org/wiki/Hans_Peter_Luhn>

2. Adicione a soma dos dígitos dos resultados obtidos no passo anterior, à soma dos dígitos que não foram multiplicados por 2.
3. Se o último dígito do total for 0 (ou, mais formalmente, se o total obtido no passo anterior for dividio por 10 e tiver resto 0), então o número representa um cartão de crédito válido (sintaticamente).

O algoritmo é simples mas um pouco confuso para quem está iniciando, então vamos usar como exemplo o cartão Visa do David, `4003600000000014`:

1. Para ilustrar o primeiro passo do algoritmo mais claramente, começando do penúltimo dígito e indo em direção ao primeiro, vamos sublinhar cada dígito sim, dígito não:
   4̲0̲0̲3̲6̲0̲0̲0̲0̲0̲0̲1̲4
   OK, agora vamos multiplicar cada um dos dígitos sublinhados por 2:
   $(1 \times 2) + (0 \times 2) + (0 \times 2) + (0 \times 2) + (0 \times 2) + (6 \times 2) + (0 \times 2) + (4 \times 2)$
   Isso resulta no seguinte:
   $2 + 0 + 0 + 0 + 0 + 12 + 0 + 8$
   Agora vamos *somar os dígitos* dos resultados (note que não somamos os resultados, somamos os dígitos dos resultados; preste atenção no número 12):
   $2 + 0 + 0 + 0 + 0 + 1 + 2 + 0 + 8 = 13$
2. Agora vamos adicionar a soma obtida no passo anterior (13) à soma dos dígitos que não foram multiplicados por 2 (começando pelo final):
   $13 + 4 + 0 + 0 + 0 + 0 + 0 + 3 + 0 = 20$
3. Ótimo, o último dígito na soma obtida no passo anterior (20) é 0, então o cartão do David é válido (pelo menos sintaticamente)!

Como você pôde ver, validar números de cartões de crédito não é difícil, mas é bem tedioso de se fazer manualmente. Sua tarefa é escrever um programa para fazer isso!

4 Detalhes para a implementação

Crie um arquivo chamado `credit.c` ou `credit.py`[7] e escreva um programa que solicite ao usuário um número de cartão de crédito e então relate (imprimindo uma mensagem) se é um cartão válido da American Express, MasterCard ou Visa, de acordo com as definições de cada cartão apresentadas anteriormente (Seção 2).

ATENÇÃO: o output de seu programa deve ser apenas `AMEX\n` ou `MASTERCARD\n` ou `VISA\n` ou `INVÁLIDO\n`, nada mais, nada nada menos.

Para simplificar um pouco as coisas, assuma que o input do usuário será inteiramente numérico (sem hífens, espaços ou outros caracteres que não sejam os dígitos do número) (se estiver usando C, não assuma que o input do usuário irá caber em um `int`, É melhor usar `long`).

Considere o exemplo abaixo como representativo do comportamento correto para seu programa (em C) quando passamos um número que, supostamente, representa um cartão de crédito válido (sem hífens, espaços ou outros caracteres não numéricos):

[7] Na CS50 os alunos utilizam a linguagem C para implementar essa tarefa de programação. Aqui você pode usar C ou Python

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

Fica por sua conta pegar inputs que não são números de cartões de crédito (por exemplo, um número de telefone), mesmo se totalmente numéricos:

```
$ ./credit
Número: 6176292929
INVÁLIDO
```

Se você quiser, implemente algo que rejeite hífens ou outros caracteres que não sejam dígitos (em C, se você usar o `get_long` da biblioteca CS50, incluindo a header file `cs50.h`[8], essa funcionalidade já estará implementada para você de graça!):

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

Teste seu programa com vários inputs diferentes, válidos e inválidos (nós, com certeza, faremos isso!). Você pode utilizar o conjunto de números de cartões de crédito de teste[9] disponibilizados pelo PayPal justamente para testes em programas de validação.

Se seu programa não funcionar corretamente em alguns inputs (ou não compilar corretamente), cabe a você debugar!

Se precisar de mais orientações, assista um vídeo com algumas explicações adicionais, passo a passo, em <https://youtu.be/dF7wNjsRBjI>.

5 Como enviar seu código

Você deve subir seu código para seu repositório GitHub e enviar o link para o repositório na planilha online disponibilizada pelo professor. O link para a planilha será divulgado no Portal do Aluno.

[8] Biblioteca de funções utilitárias para os alunos da CS50. Se você quiser cumprir a tarefa em C, instale a biblioteca confore as instruções em <https://github.com/cs50/libcs50>.

[9] <https://developer.paypal.com/api/nvp-soap/payflow/payflow-pro/payflow-pro-testing/>