PSET: *Cash*

Harvard CS50 Staff

2023-08-22

Resumo

Este PSET corresponde ao "Cash" original da disciplina Harvard CS50, em sua versão integral, traduzido e adaptado para o português pelo prof. Abrantes Araújo Silva Filho.

Sumário

1 Introdução

Este PSET corresponde ao "*Cash*" da disciplina **Harvard CS50**, e deve ser feito pelos alunos que ainda não estão se sentindo confortáveis com a programação na Linguagem C.

O objetivo deste PSET não é que você se torne um especialista em C mas, sim, que você comece a aprender os conceitos fundamentais da computação e da programação. A tradução e adaptação para o português foram feita com base na versão de 2023 do PSET, conforme o PSET original[1].

2 Algoritmos gulosos

Ao dar um troco em moedas para um cliente, na maioria das vezes, você quer minimizar o número de moedas que está entregando para não ficar sem moedas (ou para não irritar o cliente!). Felizmente a ciência da computação ofereceu aos caixas de todos os lugares maneiras de minimizar o número de moedas devidas: os **algoritmos gulosos**.

De acordo com o Instituto Nacional de Padrões e Tecnologia (*National Institute of Standards and Technology* — NIST), um algoritmo guloso é aquele "que sempre pega a melhor solução imediata, ou local, ao encontrar uma resposta. Os algoritmos gulosos encontram a solução ótima geral,

[1] https://cs50.harvard.edu/x/2023/psets/1/cash/

ou global, para alguns problemas de otimização, mas podem encontrar soluções não ótimas para algumas instâncias de outros problemas".

O que isso tudo significa? Bem, suponha que um caixa deva algum troco a um cliente e, na gaveta desse caixa, há moedas de cinqüenta centavos ($50¢$[2]), de vinte e cinco centavos ($25¢$), de dez centavos ($10¢$), de cinco centavos ($5¢$) e de um centavo ($1¢$). O problema a ser resolvido é decidir quantas moedas de cada valor devemos entregar para o cliente, de modo que o troco seja correto e que ele receba a menor quantidade possível de modeas.

Para resolver esse problema, pense em um caixa "guloso", em alguém que quer sempre tirar a maior fatia possível deste problema a cada moeda que tira da gaveta. Por exemplo, se um cliente deve receber $41¢$ de troco, como um caixa "guloso" faria?

- A primeira morida (ou seja, a "melhor imediata" ou "melhor local") que pode ser dada é de $25¢$ (essa mordida é a "melhor" na medida em que nos aproxima de $0¢$ mais rápido do que qualquer outra moeda faria). Note que uma mordida deste tamanho reduziria o problema de $41¢$ para um problema de $16¢$, pois $41 - 25 = 16$. Ou seja, o restante é um problema similar, mas menor;
- Obviamente outra mordida de $25¢$ seria grande demais (supondo que o caixa prefira não perder dinheiro), e então nosso caixa guloso seguiria para uma mordida de $10¢$, deixando-o com um problema de $6¢$;
- Nesse ponto, a gula pede por uma mordida de $5¢$;
- E, por último, uma mordida de $1¢$, momento em que o problema é resolvido.

O cliente recebe uma moeda de vinte e cinco centavos, uma de dez centavos, uma de cinco centavos e uma de um centavo: quatro moedas no total.

Acontece que essa abordagem gulosa (ou seja, esse tipo de algoritmo) não é apenas otimizada localmente, mas também globalmente para a moeda dos EUA, da União Européia, Brasil e alguns outros países (depende do conjunto de moedas de cada país). Isso é, desde que um caixa tenha quantidade suficiente de cada moeda, essa abordagem do maior para o menor resultará no menor número de moedas possível. Quantas são? Bem, você nos diz!

Seu trabalho, neste PSET, é criar um programa que implemente essa abordagem de algoritmos gulosos para calcular o menor número de moedas possíveis para dar de troco para algum cliente.

3 Detalhes de implementação

Utilize o *starter file* com o nome "`cash.c`". Nesse arquivo nós já implementamos a maior parte (mas não tudo!) de um programa que solicita ao usuário o número de centavos que um cliente deve receber de troco e, em seguida, imprime o menor número de moedas com o qual esse troco pode ser dado.

Na verdade a função `main` já está implementada para você, você não precisa fazer nenhuma alteração nela, mas estude o código e observe como a a função `main` chama várias outras funções que ainda não foram implementadas. São essas funções não implementadas que você deve programar!

Uma dessas funções, `get_troco`, não recebe argumentos (como indicado pelo parâmetro `void`) e retorna um tipo `int`. O restante das funções recebe um argumento, um `int`, e também retorna um `int`. No *starter file* elas não estão programadas, apenas retornam o valor padrão de 0 para que o arquivo possa ser compilado. Você precisa substituir todos os "`TODO`" e todos os "`return 0;`" pelo seu próprio código. Especificamente, complete a implementação dessas funções da seguinte maneira:

- Implemente a função "`get_troco`" de forma que a função solicite ao usuário o total do troco em centavos usando, usando a "`get_int`", e retorne esse número como um `int`. Se

[2]Neste documento utilizaremos o símbolo "¢" (centavos ou cêntimos) quando nos referirmos às moedas, para facilitar a compreensão, mas este não é um símbolo internacionalmente reconhecido: é usado apenas em alguns países.

o usuário inserir um inteiro negativo, seu código deve solicitar ao usuário que informe novamente (você não precisa se preocupar com o usuário inserindo, por exemplo, uma `string`, pois a função `get_int` cuidará disso para você). Provavelmente, você achará útil um loop `do...while`!

- Implemente a função "`calcular_moedas50`", de forma que a função calcule (e retorne como `int`) quantas moedas de $50¢$ o cliente deve receber, se tiver direito. Por exemplo: se o `troco` total for de $63¢$, "`calcular_moedas50`" deve retornar 1. Se o `troco` for de $26¢$ ou $49¢$, "`calcular_moedas50`" deve retornar 0. E assim por diante.
- Implemente a função "`calcular_moedas25`", de forma que a função calcule (e retorne como `int`) quantas moedas de $25¢$ o cliente deve receber, se tiver direito.
- Implemente a função "`calcular_moedas10`" de forma que a função calcule o mesmo para moedas de $10¢$.
- Implemente a função "`calcular_moedas05`" de forma que a função calcule o mesmo para moedas de $5¢$.
- Implemente a função "`calcular_moedas01`" de forma que a função calcule o mesmo para moedas de $1¢$.

Note que, ao contrário das funções que apenas têm **efeitos colaterais**, as funções que **retornam um valor** devem fazê-lo explicitamente com o comando `return`! Tenha cuidado para não modificar o código que já está pronto para você, apenas substitua os `TODO`s fornecidos e o valor de retorno correspondente! Note também que, lembrando da idéia de abstração, cada uma de suas funções `calcular` deve aceitar qualquer valor de `troco`, não apenas os valores que o algoritmo guloso pode sugerir. Se o `troco` for de $85¢$, por exemplo, `calcular_moedas10` deve retornar 8.

O comportamento esperado do seu programa está ilustrado abaixo:

```
$ ./cash
Informe o troco devido (em centavos): 41
4
$
```

```
$ ./cash
Informe o troco devido (em centavos): -41
Informe o troco devido (em centavos): banana
Informe o troco devido (em centavos): 41
4
$
```

3.1 Como testar seu código?

Uma parte importante é o teste de seu código final. Ele funciona conforme as especificações? E se o usuário digitar entradas inválidas, como valores negativos, letras, palavras ou caracteres especiais? E se o usuário não digitar nada, só pressionar a tecla "Enter"? Verifique o seguinte:

- Se o usuário digitar -1, o programa pergunta novamente?
- Se o usuário digitar 0, o prgrama informa 0 como a resposta?
- Se o usuário digitar 1, o prgrama informa 1 como a resposta (uma moeda de $1¢$)?
- Se o usuário digitar 4, o prgrama informa 4 como a resposta (quatro moedas de $1¢$)?
- Se o usuário digitar 5, o prgrama informa 1 como a resposta (uma moeda de $5¢$)?

- Se o usuário digitar `24`, o prgrama informa `6` como a resposta (duas moedas de `10¢` e quatro moedas de `1¢`)?
- Se o usuário digitar `25`, o prgrama informa `1` como a resposta (uma moeda de `25¢`)?
- Se o usuário digitar `26`, o prgrama informa `2` como a resposta (uma moeda de `25¢` e uma moeda de `1¢`)?
- Se o usuário digitar `99`, o prgrama informa `8` como a resposta (uma moeda de `50¢`, uma moeda de `25¢`, duas moedas de `10¢`, e quatro moedas de `1¢`)?

Lembre-se também de que seu código deve seguir todas as normas de estilo de programação C da disciplina Harvard CS50: Harvard CS50 C Style Guide[3].

3.2 Como enviar seu código?

Utilize o starter file "`cash.c`" (mantenha esse padrão de nome), preencha as informações de identificação e envie o arquivo no Autolab, no PSET denominado "Cash".

[3] https://cs50.readthedocs.io/style/c/