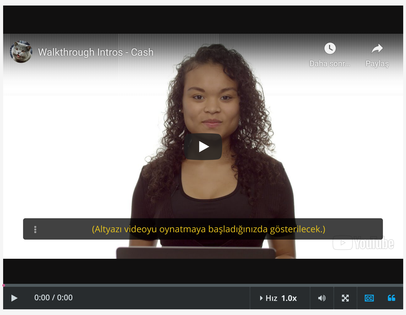
"Cash" Görevinin Tanımı

Bu sayfaya yer imi koy

Video



Altyazının başlangıcı. Sona atla.

Bir dükkandan bir şey satın aldığınızda, tam para vermediyseniz

para üstü olarak biraz bozukluk alacaksınız demektir.

Doğru miktarda para üstü aldığınızı varsayalım.

Muhtemelen para üstü olarak çok fazla bozukluk taşımak istemezsiniz.

Kasiyerin size mümkün olduğunca az bozuk para vermesini istersiniz.

Örneğin, yarım dolar alacağınız varsa,

Kasiyerin size $0.01 değerindeki penny'lerden 50 tane vermesini mi

Öncelikle, CS50 Lab'de bu sayfayı açın **ve aşağıdakileri takip edin:**

Greedy (Açgözlü) Algoritmalar

Para üstü hazırlarken, muhtemelen her müşteriye vereceğiniz bozuk para sayısını, elinizdeki bozuk paraları tüketmemek (ve diğer müşterileri geçik etmemek için) minimumda tutmak isteyeceksinizdir. Ne şanslıyız ki bilgisayar bilimleri, tüm dünyadaki kasiyerlere minimumda bozuk para üstü vermek için bir yol sunuyor: greedy algoritmalar.

Ulusal Standartlar ve Teknoloji Enstitüsü'ne (NIST) göre, greedy algoritması, bir cevaba ulaşırken her zaman en iyi ve en yakın, ya da yerel çözümü kullanandır. Greedy Algoritmaları bazı optimizasyon problemlerinde genel olarak optimum çözümü bulur. Ancak, başka problemlerin bazı örneklerinde optimum çözümden biraz daha az idealini bulabilirler.

Bütün bunlar ne demek peki? Şöyle ki, düşünün ki bir kasiyer bir müşteriye belli bir miktarda bozuk para vermek durumunda, ve yazarkasada quarter (25 cent), dime (10 cent), nickel (5 cent) ve penny (1 cent) olarak bozuk parası var (NOT: 100 CENT 1 DOLAR EDER). Problem, müşteriye hangi bozuk paradan kaç tane verileceği. Açgözlü (greedy) bir kasiyer düşünün ki yazarkasadan çıkardığı her bozuk para ile bu problemin mümkün olan en büyük payını halletmeye çalışıyor. Mesela, eğer müşteriye 41 cent vermesi gerekiyorsa, ilk olarak çıkarabileceği en büyük para 25 centlir. (Bu bizi 0 cent'e diğer tüm bozuk paralardan daha hızlı şekilde yaklaştırabildiği kadar yaklaştıracak en büyük hamledir.) Fark ettiyseniz böylesine büyük bir hamle bizi 41 centlik bir problemden 16 centlik bir probleme düşürüvermiştir, 41-25=16 olduğu için. Bu da demek oluyor ki kalan kısım, yine benzer ama daha küçük bir problem sunmaktadır. Şimdi, yazarkasadan başka bir 25 cent çıkarmanın çok fazla geleceği aşikardır, söylememize bile gerek yok sanıyoruz (tabii kasiyer para kaybetmek istemiyorsa). Böylelikle bizim açgözlü kasiyerimiz, 10 centlik diğer hamlesine geçecektir ki bu da onu 6 centlik yeni bir problemle karşılaştıracaktır. Bu noktada açgözlülüğü onu 5 centlik bir hamleye iter, ve daha sonra da 1 centlik bir hamleye, ve bu noktada da problemimiz çözülmüştür. Müşteri bir quarter (25 cent), bir dime (10 cent), bir nickel (5 cent) ve bir penny (1 cent), yani toplamda 4 bozuk para alacaktır.

Öyle ki, bu açgözlü tutum (algoritma) sadece yerel olarak optimum değil - dünya üzerinde Amerika paraları ve Avrupa Birliği paraları için de geçerli. Yani, eğer kasiyer her bozuk paradan yeterince sahipse, bu en büyükten en küçüğe taktiği, en az sayıda bozuk para ile sonuçlanacaktır. Ne kadar az? Eh onu da siz bize söyleyin!

Uygulamanın Detayları

Size yukarıda yazdığımız CS50 Lab linkinde, cash.c 'nin içeriğini bir program olarak yazın ki kullanıcıya ne kadar para üstü verilmesi gerektiğini sorsun, ve daha sonra minimum olarak kaç adet bozuk para ile bunun oluşturulabileceğini yazdırsın.

- Kullanıcın girdisini almak için get_float kullanın, ve kendi cevabınızı yazdırmak için printf kullanın. Farz edin ki halihazırda sadece quarter (25 cent), dime (10 cent), nickel (5 cent) ve penny (1 cent) olarak bozuk paralar var. Sizden kullanıcı girdisini okumak için get_float kullanmanızı istiyoruz, çünkü bu sayede dolarları ve centleri dolar işareti olmadan hesaplayabilirsiniz. Yani, eğer bir müşteriye $9.75 geri vermemiz gerekirse (mesela gazetenin 25 cent olduğu ve müşterinin 10 dolar uzattığı bir durumda), programınızın girdisinin 9.75 olacağını düşünün, $9.75 veya 975 değil. Ancak, eğer bir müşterinin para üstü tam olarak $9 ise, programınızın girdisi 9.00 olacaktır, yine $9 veya 900 değil. Tabii ki, ondalıklı rakamların (floating point) doğası gereği, programımız 9.0 ve 9.000 gibi girdiler ile de çalışabilecektir, bu konuda kullanıcının girdisini doğru bir şekilde yazıp düzenlediğini kontrol etmek durumunda değilsiniz.
- Kullanıcının girdiği rakamın bir float'a sığmayacak kadar büyük olup olmaması konusunda kontrol etmeyi denemenize gerek yoktur. get_float kullanmanız kullanıcı girdisinin ondalıklı bir sayı olduğundan emin olacaktır, ancak negatif olup olmamasını sağlamaz.
- Eğer kullanıcı negatif bir sayı girerse, programınız kullanıcıya doğru bir miktar girene kadar tekrar tekrar sormalıdır.
- Kodunuzu otomatik olarak bazı testlere tabi tutabilmemiz için, programınızın çıktısının en son satırının sadece mümkün olan en az miktardaki bozuk para sayısını verdiğinden ve \n ile bittiğinden emin olun.
- Ondalıklı sayıların kendilerinden kaynaklanan hassasiyet kayıplarına dikkat edin. Sınıftan floats.c 'yi hatırlayın. Hani eğer x 2 ise ve y 10 ise, x/y tam olarak 2/10 yapmıyordu! Bu yüzden, küçük hataların daha sonra çok daha büyük hatalara yol açmaması için, para üstünü hesaplamadan, kullanıcının girdiği dolarları cente çevirmek isteyebilirsiniz (mesela float 'tan int 'e).
- Cent'leri en yakın penny'ye yuvarlamaya dikkat edin (math.h içinde tanımlanan round kullanarak). Örnek olarak, eğer dollars kullanıcının girdisini içeren bir float ise, (mesela 0.20), o zaman şu şekilde kodlamanız:

int cents = round(dollars * 100);
güvenli şekilde 0.20 yi, (ve hatta 0.20000000298023223876953125'yi) 20'ye dönüştürecektir.

Programınız aşağıdaki örnekteki gibi davranmalıdır:

$ ./cash
Change owed: 0.41
4

$ ./cash
Change owed: -0.41
Change owed: foo
Change owed: 0.41
4

Ekibimizin Çözümü

Ekibimizin çözümünü denemek için bu sandbox içinde aşağıdaki komutu çalıştırın:

./cash

Kodunuzun Nasıl Test Edileceği

Kodunuz aşağıdaki girdileri yazdığınızda beklenildiği gibi çalışıyor mu?

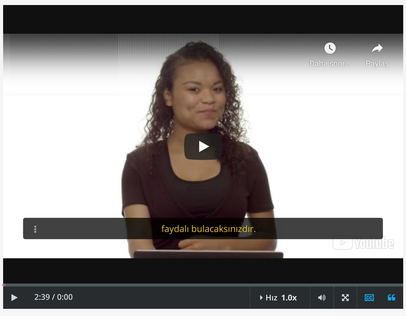
- -1.00 (veya başka negatif sayılar)?
- 0.00?
- 0.01 (veya diğer pozitif sayılar)?
- harfler ve kelimeler?
- hiç bir girdi olmayınca, yani sadece Enter tuşuna bastığınızda?

Şimdi sizin ödeviniz:

Ödevinizi, bir sonraki ekranda çıkan kutuya kodunuzu yazarak göndereceksiniz. Ödevinizi bize göndermeden, aşağıdaki ödev tanımını okuduğunuzdan emin olun. Sonra da gönderirken dikkat etmeniz gerekenler şunlar:

- Bu ödevden bir puan alacaksınız ve ilerleyişinize işlenecek.
- Ödevi sistemde bize göndermeden önce, mutlaka CS50 Lab'de test etmenizi öneririz.
- Eğer sonucunuz yanlışsa 0 puan, doğruysa 1 puan alacaksınız. Doğru yapana kadar birkaç kez deneme şansınız var.
- Eğer sonucunuz yanlışsa, aşağıda "See Output" kısmından neler olduğunu inceleyebilirsiniz.
- **Cevabınız doğru olduğunda, mutlaka Gönder tuşuna basıp sonucu bize göndermeyi unutmayın!**
- İnternet bağlantınız ve problemlere bağlı olarak, sonuçların otomatik değerlendirilmesi bazen uzun sürebilir. Bu durumda 5-10 dakika beklemeniz gerekebilir. Eğer çalışmıyorsa sayfanızı yenileyebilirsiniz.
- Ödev için ayrıca aşağıdaki ipucu videosunu da izleyebilirsiniz.

İpucu videosu



Kullanılmış olan bozuklukların takibini yapmak ve toplamda kaç tane bozukluk kullanıldığını yazdırmak istiyoruz.

O halde önce kullanıcıdan girdi üstüme üzerine konuşalım.

Kullanıcıdan ne zaman bir girdi istesek her seferinde

problemin çözümünde işimize yarayacak girdiler olduğundan emin olmalıyız.

Bu durumda, CS50 kütüphanesindeki get_float fonksiyonunu epey

faydalı bulacaksınızdır.

İşte size kullanıcının geçerli bir kayan nokta

sağlamasını sağlayacak bir kod snippet'i.

Burada, CS50 kütüphanesinde bulunan get_float fonksiyonunu

kullanarak kullanıcıdan girdi alan döngümüm gövdesini en az bir kez

çalıştıracak bir do-while döngüsü var.

Bundan sonra, bu kayan nokta geçersiz olduğu sürece,

kullanıcıdan yeniden girdi istemeye devam edeceğim.