Görevlerin Türkçe çevirisi “courses.kodluyoruz.org” dan alınmıştır.

"Mario" Görevinin Tanımı | Problem seti 1 | CS50x Ders yazılımları

6-8 dakika

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

Bu ödev, ünlü Mario oyunu üzerine kurulu! Göreviniz, Mario'nun üzerinden atlayacağı tuğlaları oluşturmak olacak. Bu problem seti sayesinde fonksiyonlar, değişkenler, döngüler ve daha fazlasını pratik edeceksiniz. Haydi başlayalım!

World 1-1

Nintendo’nun Super Mario Brothers oyununda World 1-1’in sonuna doğru, Mario sağa doğru hizalanmış blocktan piramitin üstüne tırmanmalıdır, aşağıdaki gibi.

Haydi şimdi bu piramiti C’de tekrar yaratalım, ama bu sefer yazı halinde, tuğlalar yerine hash yani kare (#) karakterleri kullanarak, tıpkı aşağıdaki gibi. Her kare işareti genişliğine göre biraz daha uzun, o nedenle piramidin kendisi de genişliğinden daha uzun.

```
$ python mario.py
Height: 5
    #
   ##
  ###
 ####
#####

$ python mario.py
Height: 3
  #
 ##
###
```

Yazacağımız program mario olarak adlandırılacak. Aynı zamanda, kullanıcının piramidin ne kadar yüksek (height) olacağına karar vermesine de ondan 1 ile 8 arasında (8 dahil) bir sayı girmesini isteyerek izin vereceğiz.

İşte size programın kullanıcı 8 girdiğinde nasıl çalışacağı:

```
$ ./mario
Height: 8
        #
       ##
      ###
     ####
    #####
   ######
  #######
########
```

İşte size programın kullanıcı 4 girdiğinde nasıl çalışacağı:

```
$ ./mario
Height: 4
    #
   ##
  ###
####
```

İşte size programın kullanıcı 2 girdiğinde nasıl çalışacağı:

```
$ ./mario
Height: 2
 #
##
```

İşte size programın kullanıcı 1 girdiğinde nasıl çalışacağı:

```
$ ./mario
Height: 1
#
```

Eğer kullanıcı sorulduğunda 1 ve 8 arasında (ve bunlar dahil) pozitif bir rakam girmezse, program onlara tekrar tekrar ta ki kullanıcı istenilene uyana kadar sormaya devam etmeli:

```
$ ./mario
Height: -1
Height: 0
Height: 42
Height: 50
Height: 4
   #
  ##
 ###
####
```

Deneyin! Ekibimizin çözümünü denemek için bu linke gidin, ./mario yazın, ve sonucu test edin!

Nereden mi başlayacaksınız? Haydi bu problemi beraber adım adım nasıl çözebileceğimize bakalım:

Pseudocode (sözde kod)

Öncelikle, kodunuzu nasıl yazacağınızdan emin değilseniz sözde kodlam başlayın: Bu sayfada göreceğiniz sağdaki pseudocode.txt içine, bu programı uygulayacak bir pseudocode (sözde kod) yazın. Sözde kodu yazmanın tek bir doğru yolu yoktur, ama kısa cümleler yeterli olacaktır. finding Mike Smith (Mike Smith'i bulmak) için nasıl sözde kod yazdığımızı hatırlayın (ders notlarında 44. slaytta) Muhtemelen sözde kodunuz bir veya daha çok fonksiyon, durum, Boolean ifadesi, tekrar, ve/veya değişken kullanacak (veya kullanılacağını ima edecek).

Spolier (çözümün sonunu söylüyoruz):

Sözde kodunuzu yapmanın bir çok yolu var, işte size örnek bir sözde kod:

1. Kullanıcıya yüksekliği sor.
2. Eğer (if) yükseklik 1 den küçükse veya 8'den büyükse (veya bir tam sayı yani integer değilse), bir adım geri git.
3. 1'den yüksekliğe kadar şunu uygula: Uygulamanın i. basamağında, önce i kadar kare yazdırın ve sonra yeni bir satıra geç.

Kendinizinkini bu sözde kodu gördükten sonra değiştirmenizde bir sorun yok, ama bizim çözümümüzü direk olarak kedinizinkine kopyala yapıştır yapmayın!

Kullanıcıya Girdi için Soru Yöneltmek

Sözde kodunuz ne olursa olsun, öncelikle kullanıcıya girdi yapması için soru yöneltecek (ve gerekirse tekrar soracak) olan C kodunu yazalım.

Özellikle, sağda mario.c yu öyle bir değiştirin ki, önce kullanıcıya piramidin yüksekliğini sorsun, girdikleri rakamı bir değişkene kaydetsin, ve girdileri değer 1 ile 8 arasında (dahil olarak) bir rakam değilse, tekrar tekrar sorsun. Sonra basitçe bu değişkenin değerini yazdırsın ki gerçekten kullanıcının girdiği değeri başarılı bir şekilde kaydettiğinizden emin olun. Aynı aşağıdaki gibi:

```
$ ./mario
Height: -1
Height: 0
Height: 42
Height: 50
Height: 4
Stored: 4
```

İpucu:

- Hatırlayın: Programınızı make ile derleyebilirsiniz
- Hatırlayın: Bir int değerini printf ile %i.kullanarak yazdırabilirsiniz.
- Hatırlayın: Kullanıcıdan bir rakam girdisini get_int ile alabilirsiniz
- Hatırlayın: get_int aslında cs50.h içinde tanımlanmıştır.
- Hatırlayın: Derste positive.c üzerinden kullanıcıya pozitif bir sayı girmesi için do while döngüsü kullandık.

Tersini İnşa Etmek

Şimdi programınız (umarız ki) öngörüldüğü gibi bir girdi kabul ediyordur. Şimdi bir adım daha atma vakti.

Öyle görünüyor ki sola doğru yaslanan bir piramit inşa etmek, sola yaslanan bir tanesini inşa etmekten daha kolay, aşağıdaki gibi.

```
#
##
###
####
#####
######
#######
########
```

O nedenle haydi öncelikle sola yaslanan piramidi yapalım, ve bunu çalıştırınca, bunun yerine sağa yaslarız!

Şimdi, açtığınız CS50 Lab linkinde mario.c 'yi öyle değiştirelim ki, artık sadece kullanıcının girdiği rakamı yazdırmasın, onun yerine girilen yükseklikte sola-yaslanan bir piramid yazdırsın.

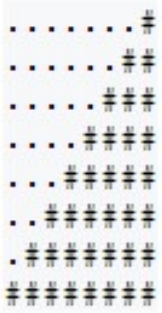
İpucu:

- Unutmayın ki kare işareti de diğerleri gibi herhangi bir karakterdir yani onu printf ile yazdırabilirsiniz.
- Aynı Scracth'in tekrar bloğu olması gibi, C de belirli bir sayı kadar uygulayabileceğiniz bir döngüye sahiptir. Muhtemelen her adımda (uygulamada/iteration), i, bu kadarlık kare yazabilirsiniz.
- Aslında döngüleri bir biri içine koyabilirsiniz (nest), dış döngüde bir değişken (örneğin i) ile adım adım ilerlersiniz, ve içerideki döngüde başka bir değişken (örneğin j) ile. Mesela, aşağıda n yükseklik ve uzunlukta bir kareyi nasıl yazdıracağınız bulunuyor. Tabii ki sizin yazdırmak istediğiniz bir kare değil!

```
for (int i = 0; i < n; i++)

{

   for (int j = 0; j < n; j++)

   {

      printf("#");

   }

   printf("\n");

}
```

Noktalar İle Sağa Hizalamak

Şimdi piramidi hash'leri yani kare karakterlerini başlarına noktalar koyarak sağa itelim ve aşağıdaki gibi sağa hizalayalım:



Açtığınız CS50 Lab linkinde mario.c'yi değiştirin ki tam da bunu yapsın!

İpucu:

Dikkat ettiniz mi? Her satırda gerekli olan nokta sayısı, o satırdaki karelerin tam "tersi". Yukarıdaki gibi 8 yükseklikteki bir piramit için, ilk satır bir kare ve 7 nokta içermekte. En alt satır ise, 8 kare ve 0 nokta içermekte. Hangi formül (veya daha doğrusu aritmetik ile) o sayıda nokta yazdırabilirsiniz?

Kodunuzu Nasıl Test Edebilirsiniz?

Kodunuz, aşağıdaki girdileri girdiğinizde beklenildiği gibi çalışıyor mu? Bu girdileri girerek kodunuzu test edebilirsiniz:

- -1 (veya diğer negatif sayılar)?
- 0
- 1 ve 8 arasındaki sayılar
- 9 veya diğer pozitif sayılar
- harfler veya kelimeler
- hiç bir girdi olmadan, yani sadece Enter tuşuna bastığınızda?

Noktaları Kaldırmak

Şimdi bitirmek için tek yapmamız gerek şey son bir küçük dokunuş. mario.c'yi düzenleyin ve noktalar yerine boşluklar yazdırmasını sağlayın. Böylece ödevinizin sizden istediği gibi, sağa yaslanan bir piramidiniz olsun.

İpucu:

Bir boşluk, sadece klavyedeki boşluk tuşuna bastığınızda oluşan boşluktur! Sadece hatırlayın ki printf iki tarafını da çift tırnak işareti (") ile çevrelemenizi gerektirmektedir.

"Mario Advanced" Görevinin Tanımı | Problem seti 1 | CS50x Ders yazılımları

3-4 dakika

"Mario Advanced" Görevinin Tanımı

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

World 1-1

Nintendo'nun Super Mario Brothers oyununun World 1-1'in başına doğru, Mario aşağıdaki gibi iki yan yana piramidin üzerinden atlamalıdır.

Şimdi bu piramitleri C ile tekrar yaratalım: Ama yine yazı halinde, tuğlalar yerine kare (#) kullanarak, aynı aşağıdaki gibi. Her kare karakteri genişliğinden biraz daha uzun. O nedenle piramitler de genişliklerinden biraz daha uzun.

```
   #  #
  ##  ##
 ###  ###
####  ####
```

Yazacağımız program mario olarak adlandırılacak. Aynı zamanda, kullanıcının piramidin ne kadar yüksek olacağına karar vermesine de ondan 1 ile 8 arasında (8 dahil) bir sayı (height) girmesini isteyerek izin vereceğiz.

İşte size programın kullanıcı 8 girdiğinde nasıl çalışacağı:

```
$ ./mario
Height: 8
       #  #
      ##  ##
     ###  ###
    ####  ####
   #####  #####
  ######  ######
 #######  #######
########  ########
```

İşte size programın kullanıcı 4 girdiğinde nasıl çalışacağı:

```
$ ./mario
Height: 4
   #  #
  ##  ##
 ###  ###
####  ####
```

İşte size programın kullanıcı 2 girdiğinde nasıl çalışacağı:

```
$ ./mario
Height: 2
 #  #
##  ##
```

İşte size programın kullanıcı 1 girdiğinde nasıl çalışacağı:

```
$ ./mario
Height: 1
#  #
```

Eğer kullanıcı, sorulduğunda 1 ve 8 arasında (bunlar dahil) pozitif bir rakam girmezse, program onlara tekrar tekrar ta ki kullanıcı istenilene uyana kadar sormaya devam etmeli:

```
$ ./mario
Height: -1
Height: 0
Height: 42
Height: 50
Height: 4
   #  #
  ##  ##
 ###  ###
####  ####
```

Piramitlerin yüksekliklerinden bağımsız olarak iki yan yana piramidin arasındaki boşluğun iki kare karakterine eşit olduğunu fark ettiniz mi? Aynı zamanda piramidin sağında hiç bir boşluk olmaması gerektiğine de dikkat edin.

Açmış olduğunuz CS50 Lab'de mario.c 'yi tarif edildiği üzere bu programı uygulayacak şekilde değiştirin!

Ekibimizin Çözümü

Ekibimizin mario çözümünü denemek için **bu sandbox içinde** aşağıdaki komutu çalıştırın.

./mario

Kodunuzu Nasıl Test Edeceğiniz

Kodunuz aşağıdaki girdileri girdiğinizde beklenildiği gibi çalışıyor mu?

- -1 (veya diğer negatif sayılar)?
- 0
- 1 ve 8 arasındaki sayılar
- 9 veya diğer pozitif sayılar
- harfler veya kelimeler
- hiçbir girdi olmadan, yani sadece Enter tuşuna bastığınızda?

"Cash" Görevinin Tanımı | Problem seti 1 | CS50x Ders yazılımları

6-8 dakika

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 gıcık 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 centtir. (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, programınız 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 0.2000000029802322387695312 50'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?

"Credit" Görev Tanımı | Problem seti 1 | CS50x Ders yazılımları

5-7 dakika

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

Kredi

Bir kredi (veya banka) kartı, tabii ki de alışveriş yapabildiğiniz plastik bir kart. Bu kartın üzerinde aynı zamanda bir yerlerdeki bir veri tabanında da saklanan bir numara var. Böylece kartınız bir şeyler almak için kullanıldığında, banka faturayı kime keseceğini biliyor. Dünyada kredi kartı sahibi birçok insan var o nedenle de bu numaralar epey uzunlar. American Express 15 haneli sayılar kullanıyor, MasterCard 16 haneli sayılar kullanıyor, ve Visa 13 ve 16 haneli rakamlar kullanıyor. Bu rakamlar ikili sistemde değil, ondalık sistemde olan rakamlar (0'dan 9'a). Bu da demek oluyor ki American Express 10^15 = 1,000,000,000,000,000 adet farklı kart basabilir! (Hmm, bu kuadrilyon demek!).

Aslında bu sayı abartılı, çünkü kredi kartı rakamlarının aslında bir sistematiği var. American Express rakamları 34 veya 37 ile başlıyor, çoğu MasterCard rakamları 51, 52, 53, 54 veya 55 ile (başka potansiyel başlama rakamları da var ama bu problem için biz bununla ilgilenmiyoruz); ve tüm Visa numaraları 4 ile başlıyor. Bunun yanında, tüm kredi kartlarının bir toplam ile kontrol mekanizması var: En az bir sayının diğerleri ile matematiksel bir ilişkisi olarak. Bu kontrol mekanizması bilgisayarların (ve matematiği seven kişilerin) hatalı girişleri veya dolandırıcılık işlemlerini herhangi bir veritabanını kullanmadan bile fark etmesini sağlıyor (numara kaydırma gibi). Tabii ki de dürüst olmayan bir matematikçi, bu matematiksel kurala uyacak sahte bir numara oluşturabilir. O yüzden, bir veritabanından kontrol etmek daha detaylı kontroller için gerekli olacaktır.

Luhn'un Algoritması

Peki gizli formül nedir? Birçok kart IBM çalışanı Hans Luhn tarafından geliştirilmiş bir algoritma kullanıyor. Luhn'un algoritmasına göre, numara dizilimi bakımından bir kartın geçerli olup olmadığını şu şekilde belirleyebilirsiniz:

1. Sondan ikinci rakamdan başlayarak her ikinci rakamı 2 ile çarpın, ve bunların çarpımlarında ortaya çıkan rakamların herbirini ayrı ayrı birbirine ekleyin.
2. Toplamı, 2 ile çarpılmamış olan sayıların rakamlarının toplamına ekleyin.
3. Eğer toplamın son hanesi 0 ise (daha resmi şekilde mod 10'da 0 veriyorsa), numara geçerlidir!

Bu biraz kafa karıştırıcı oldu, bu nedenle gelin David'in Visa kartını örnek alalım: 4003600000000014.

1. Problemi sorgularken önce sondan ikinci rakamdan başlayarak her ikinci rakamın altını çizelim:

4003600000000014

Tamam şimdi bu altı çizili rakamları 2 ile çarpalım:

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

Bu bize aşağıdaki sonucu veriyor

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

Şimdi bu ortaya çıkan rakamları (çarpımların kendi sayısını değil) birbirine ekleyin:

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

2. Şimdi bu toplamı (13), 2 ile çarpılmamış rakamların toplamına ekleyin (sondan başlayarak):

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

3. Evet! Bu toplamın (20) son rakamı 0, yani David'in kartı gerçek!

Bu şekilde kartların numaralarının gerçekliğini test etmek zor değil. Ama el ile yapınca biraz uğraştırıcı oluyor. Neden bir program yazmıyoruz?

Uygulama Detayları

Sizle yukarıda paylaştığımız CS50 Lab linkinde bulunan credit.c içinde bir program yazın. Bu program, kullanıcıya bir kredi kartı numarası girmesi için soru yöneltsin ve daha sonra bunun yukarıda tanımladığımız formattakine göre geçerli bir American Express, MasterCard, veya Visa kart numarası olup olmadığını bize (printf ile) bildirsin. Kodunuza uygulanacak bazı testleri otomatikleştirebilmemiz için programınızın son çıktısı AMEX\n veya MASTERCARD\n veya VISA\n veya INVALID\n olacak. Ne bir eksik, ne bir fazla. Kolaylık olsun diye, kullanıcının girdisinin tamamen rakamlardan oluşacağını farz edebilirsiniz (mesela bazı kartlarda olduğu gibi aralarında tire işareti olmadan). Ancak bu girdinin bir int'e sığacağını varsaymamalısınız! En iyisi kullanıcıdan girdi almak için CS50'nin kütüphanesinden get_long 'u kullanın. (Neden olduğunu tahmin edebildiniz mi?)

Aşağıda kendi programınızın geçerli (tiresiz) bir kredi kartı numarası girildiğinde nasıl davranması gerektiğine dair temsili inceleyin:

$ ./credit

Number: 4003600000000014

VISA

Şimdi, get_long tireleri (ve daha fazlasını da) geri çevirecek:

$ ./credit

Number: 4003-6000-0000-0014

Number: foo

Number: 4003600000000014

VISA

Ama, kredi kartı numarası olmayan girdileri (mesela telefon numarası) rakamlardan oluşmalarına rağmen yakalamak sizin göreviniz:

$ ./credit

Number: 6176292929

INVALID

Programınızı farklı birçok girdi ile test edin, geçerli ve geçersiz olacak şekilde. (Biz kesinlikle edeceğiz!) Burada PayPal'ın test için tavsiye ettiği bazı kredi kartı numaraları var.

Eğer programınız bazı girdiler için hatalı şekilde çalışırsa (veya hiç derlenmezse) debug etmeniz (yani hataları, bug'ları ayıklamanız) gerekiyor demektir!

Ekibimizin Çözümü

Ekibimizin credit için yazdığı çözümü denemek için bu Sandbox içinde aşağıdaki komutu girin:

./credit