Coding Exercise #2

Implement the code for a supermarket checkout that calculates the total price of a number of items. In a normal supermarket, things are identified using Stock Keeping Units, or SKUs. In our store, we'll use individual letters of the alphabet (A, B, C, and so on). Our goods are priced individually. In addition, some items are multipriced: buy n of them, and they'll cost you y cents. For example, item 'A' might cost 50 cents individually, but this week we have a special offer: buy three 'A's and they'll cost you $1.30. In fact this week's prices are:

Item	Unit Price	Special Price
A	50	3 for 130
B	30	2 for 45
C	20	
D	15	

Our checkout accepts items in any order, so that if we scan a B, an A, and another B, we'll recognize the two B's and price them at 45 (for a total price so far of 95). Because the pricing changes frequently, we need to be able to pass in a set of pricing rules each time we start handling a checkout transaction. Total can be called multiple times for a single transaction.

The interface to the checkout should look like:

```
co = CheckOut.new(pricing_rules)
co.scan(item)
co.scan(item)
...
price = co.total
```

Here's a set of example unit tests for a Ruby implementation. The helper method price lets you specify a sequence of items using a string, calling the checkout's scan method on each item in turn before finally returning the total price.

The class you write should be able to pass the following tests:

```
class TestPrice < Test::Unit::TestCase
  def price(goods)
    co = CheckOut.new(RULES)
    goods.split(//).each { |item| co.scan(item) }
    co.total
  end

  def test_totals
    assert_equal(  0, price(""))
    assert_equal( 50, price("A"))
    assert_equal( 80, price("AB"))
    assert_equal(115, price("CDBA"))

    assert_equal(100, price("AA"))
    assert_equal(130, price("AAA"))
    assert_equal(180, price("AAAA"))
    assert_equal(230, price("AAAAA"))
    assert_equal(260, price("AAAAAA"))

    assert_equal(160, price("AAAB"))
    assert_equal(175, price("AAABB"))
    assert_equal(190, price("AAABBD"))
    assert_equal(190, price("DABABA"))
  end
```

```
  def test_incremental
    co = CheckOut.new(RULES)
    assert_equal(  0, co.total)
    co.scan("A");  assert_equal( 50, co.total)
    co.scan("B");  assert_equal( 80, co.total)
    co.scan("A");  assert_equal(130, co.total)
    co.scan("A");  assert_equal(160, co.total)
    co.scan("B");  assert_equal(175, co.total)
  end
end
```

Bonus: You can set it up so that multiple discounts are available for products, for example (you might have to adjust some of the tests above for this solution):

Item	Unit Price	Special Price
A	50	3 for 130 or 4 for 170
B	30	2 for 45
C	20	
D	15	