CS5010 - Problem Set 03 - Test Results

maziji

October 10, 2012

This test suite tests your implementation of Problem Set 03

1 File: 2.rkt

This week, we codewalk problem 2
Common Definitions

```
(define inventory-1 (list (make-book 15 "How to Design Programs" "Felleisen
et al." "MIT Press" 59 49 100 (make-reorder 2 5) 1/12) (make-book 16 "A
Game of Thrones" "George R. R. Martin" "Bantam" 12 5 15 (empty-
reorder 'any)  1/20)))


(define inventory-2 (list (make-book 15 "How to Design Programs" "Felleisen
et al." "MIT Press" 49 39 2 (make-reorder 0 50) 1/12) (make-book 16 "A
Game of Thrones" "George R. R. Martin" "Bantam" 12 5 15 (empty-
reorder 'any)  1/20)))


(define line-item-1 (make-line-item 15 3))


(define order-1 (list line-item-1))


(define order-2 (list (make-line-item 42 1)))
```

1.1 Test-Group: Required Functions (2 Points)

2/2

Basic tests for the required functions not tested below

1.1.1 Test (equality)

The total value of an empty inventory should be 0
Input:

```
(stock-total-value empty)
```

Expected Output:

```
0
```

Expected Output Value:

```
0
```

Correct

1.1.2 Test (equality)

The total profit of inventory-1 should be (100 * 10 + 15 * 7)
Input:

```
(stock-total-value inventory-1)
```

Expected Output:

```
(+ (* 100 10) (* 15 7))
```

Expected Output Value:

```
1105
```

Correct

1.1.3 Test (equality)

The total volume of an empty inventory should be 0
Input:

```
(stock-total-volume empty)
```

Expected Output:

```
0
```

Expected Output Value:

```
0
```

Correct

1.1.4 Test (equality, 0.2 partial points)

The total volume of inventory-1 should be (100 / 12 + 15 / 20)
Input:

```
(stock-total-volume inventory-1)
```

Expected Output:

```
(+ (* 100 1/12) (* 15 1/20))
```

Expected Output Value:

```
109/12
```

Correct

1.1.5 Test (equality)

For an empty inventory, price-for-line-item should return false
Input:

```
(price-for-line-item empty line-item-1)
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

1.1.6 Test (equality, 0.2 partial points)

The price for line-item-1 in inventory-1 should be 3*59
Input:

```
(price-for-line-item inventory-1 line-item-1)
```

Expected Output:

```
(* 59 3)
```

Expected Output Value:

```
177
```

Correct

1.1.7 Test (equality)

An empty inventory can not fill a non-empty order
Input:

```
(fillable-now? order-1 empty)
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

1.1.8 Test (equality, 0.2 partial points)

inventory-1 should be able to fill order-1
Input:

```
(fillable-now? order-1 inventory-1)
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

1.1.9 Test (equality, 0.1 partial points)

An inventory cannot fill an order where it has not enough books on hand
Input:

```
(fillable-now? order-1 inventory-2)
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

1.1.10 Test (equality, 0.1 partial points)

An inventory cannot fill an order that contains books that are not in the inventory
Input:

```
(fillable-now? order-2 inventory-1)
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

1.1.11 Test (equality, 0.1 partial points)

The price of an order should be the sum of the prices of the line items
Input:

```
(price-for-order inventory-1 order-1)
```

Expected Output:

```
(* 3 59)
```

Expected Output Value:

```
177
```

Correct

1.1.12 Test (equality, 0.1 partial points)

The price of an order should be the sum of the prices of the line items
Input:

```
(price-for-order inventory-2 order-1)
```

Expected Output:

```
(* 3 49)
```

Expected Output Value:

```
147
```

Correct

1.2 Test-Group: days-til-fillable (2 Points)

2/2

More detailed tests for days-til-fillable

1.2.1 Test (equality)

An empty inventory can never fill a non-empty order
Input:

```
(days-til-fillable order-1 empty)
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

1.2.2 Test (equality, 0.5 partial points)

inventory-1 should be able to fill order-1 immediately
Input:

```
(days-til-fillable order-1 inventory-1)
```

Expected Output:

```
0
```

Expected Output Value:

```
0
```

Correct

1.2.3 Test (or, 0.5 partial points)

This is a tricky detail, so we also accept a slightly wrong interpretation
Test (equality)
inventory-2 should be able to fill order-1 tomorrow
Input:

```
(days-til-fillable order-1 inventory-2)
```

Expected Output:

```
1
```

Expected Output Value:

```
1
```

Wrong Output:

```
0
```

Test (equality)
It is also okay to say that inventory-2 should be able to fill order-1 today
Input:

```
(days-til-fillable order-1 inventory-2)
```

Expected Output:

```
0
```

Expected Output Value:

```
0
```

Correct

1.3 Test-Group: inventory-after-order (3 Points)

3/3

More detailed tests for inventory-after-order
Common Definitions

```
(define inventory-1-after-order-1 (list (make-book 15 "How to De-
sign Programs" "Felleisen et al." "MIT Press" 59 49 97 (make-reorder 2 5) 1/12) (make-
book 16 "A Game of Thrones" "George R. R. Martin" "Bantam" 12 5 15 (empty-
reorder 'any)  1/20)))


(define order-3 (list line-item-1 (make-line-item 16 5)))


(define inventory-1-after-order-3 (list (make-book 15 "How to De-
sign Programs" "Felleisen et al." "MIT Press" 59 49 97 (make-reorder 2 5) 1/12) (make-
book 16 "A Game of Thrones" "George R. R. Martin" "Bantam" 12 5 10 (empty-
reorder 'any)  1/20)))
```

1.3.1 Test (equality, 0.5 partial points)

An empty order should leave the inventory unchanged
Input:

```
(inventory-after-order inventory-1 empty)
```

Expected Output:

```
inventory-1
```

Expected Output Value:

```
(#(struct:book 15 "How to Design Programs" "Felleisen et al." "MIT
Press" 59 49 100 #(struct:reorder 2 5) 1/12) #(struct:book 16 "A
Game of Thrones" "George R. R. Martin" "Bantam" 12 5 15 #f 1/20))
```

Correct

1.3.2 Test (equality, 0.5 partial points)

After processing order-1, inventory-1 should have three books (of ISBN 15) less in stock
Input:

```
(inventory-after-order inventory-1 order-1)
```

Expected Output:

```
inventory-1-after-order-1
```

Expected Output Value:

```
(#(struct:book 15 "How to Design Programs" "Felleisen et al." "MIT
Press" 59 49 97 #(struct:reorder 2 5) 1/12) #(struct:book 16 "A
Game of Thrones" "George R. R. Martin" "Bantam" 12 5 15 #f 1/20))
```

Correct

1.3.3 Test (equality, 1 partial points)

After processing order-3, inventory-2 should have three books less of ISBN 15 and 5 books less of ISBN 16.
Input:

```
(inventory-after-order inventory-1 order-3)
```

Expected Output:

```
inventory-1-after-order-3
```

Expected Output Value:

```
(#(struct:book 15 "How to Design Programs" "Felleisen et al." "MIT
Press" 59 49 97 #(struct:reorder 2 5) 1/12) #(struct:book 16 "A
Game of Thrones" "George R. R. Martin" "Bantam" 12 5 10 #f 1/20))
```

Correct

1.4 Test-Group: increase-prices (2 Points)

0/2

More detailed tests for increase-prices
Common Definitions

```
(define inventory-1-after-increase (list (make-book 15 "How to De-
sign Programs" "Felleisen et al." "MIT Press" 59 49 100 (make-reorder 2 5) 1/12) (make-
book 16 "A Game of Thrones" "George R. R. Martin" "Bantam" 15 5 15 (empty-
reorder 'any)  1/20)))


(define inventory-3 (cons (make-book 14 "A Storm of Swords" "George
R. R. Martin" "Bantam" 20 5 3 (empty-reorder 'test)  1/20) inventory-
1))


(define inventory-3-after-increase (cons (make-book 14 "A Storm
of Swords" "George R. R. Martin" "Bantam" 25 5 3 (empty-reorder 'test)  1/20) inventory-
1-after-increase))
```

1.4.1 Test (equality)

An empty inventory should not change when prices are increased
Input:

```
(increase-prices empty "MIT Press" 5)
```

Expected Output:

```
empty
```

Expected Output Value:

```
()
```

Correct

1.4.2 Test (equality, 0.4 partial points)

Only books of the given Publisher should have their prices increased
Input:

```
(increase-prices inventory-1 "Bantam" 25)
```

Expected Output:

```
inventory-1-after-increase
```

Expected Output Value:

```
(#(struct:book 15 "How to Design Programs" "Felleisen et al." "MIT
Press" 59 49 100 #(struct:reorder 2 5) 1/12) #(struct:book 16 "A
Game of Thrones" "George R. R. Martin" "Bantam" 15 5 15 #f 1/20))
```

Error occured when calculating result
#(struct:exn:fail:contract string=?: contract violation expected: string? given: 25 argument position: 2nd other arguments...: "MIT Press" #<continuation-mark-set>)

1.4.3 Test (equality, 0.4 partial points)

All books of the given Publisher should have their prices increased
Input:

```
(increase-prices inventory-3 "Bantam" 25)
```

Expected Output:

```
inventory-3-after-increase
```

Expected Output Value:

```
(#(struct:book 14 "A Storm of Swords" "George R. R. Martin" "Bantam" 25 5 3 #f 1/20) #(str
to Design Programs" "Felleisen et al." "MIT Press" 59 49 100 #(struct:reorder 2 5) 1/12) #
Game of Thrones" "George R. R. Martin" "Bantam" 15 5 15 #f 1/20))
```

Error occured when calculating result
#(struct:exn:fail:contract string=?: contract violation expected: string? given: 25 argument position: 2nd other arguments...: "Bantam" #<continuation-mark-set>)

1.5 Test-Group: daily-update (3 Points)

3/3

More detailed tests for daily-update
Common Definitions

```
(define inventory-1-after-update (list (make-book 15 "How to De-
sign Programs" "Felleisen et al." "MIT Press" 59 49 100 (make-reorder 1 5) 1/12) (make-
book 16 "A Game of Thrones" "George R. R. Martin" "Bantam" 12 5 15 (empty-
reorder 'any)  1/20)))
```

```
(define inventory-2-after-update (list (make-book 15 "How to De-
sign Programs" "Felleisen et al." "MIT Press" 49 39 52 (empty-reorder 'any)  1/12) (make-
book 16 "A Game of Thrones" "George R. R. Martin" "Bantam" 12 5 15 (empty-
reorder 'any)  1/20)))
```

```
(define inventory-4 (list (make-book 53 "Book1" "Author1" "Publisher1" 100 50 14 (make-
reorder 1 20) 1/4) (make-book 54 "Book2" "Author2" "Publisher2" 50 40 7 (make-
reorder 0 3) 1/3) (make-book 55 "Book3" "Author3" "Publisher3" 20 5 59 (make-
reorder 2 50) 1/2)))
```

```
(define inventory-4-after-1-day (list (make-book 53 "Book1" "Author1" "Publisher1" 100 50
reorder 0 20) 1/4) (make-book 54 "Book2" "Author2" "Publisher2" 50 40 10 (empty-
reorder 4) 1/3) (make-book 55 "Book3" "Author3" "Publisher3" 20 5 59 (make-
reorder 1 50) 1/2)))


(define inventory-4-after-2-days (list (make-book 53 "Book1" "Author1" "Publisher1" 100 50
reorder 'any)  1/4) (make-book 54 "Book2" "Author2" "Publisher2" 50 40 10 (empty-
reorder 'any)  1/3) (make-book 55 "Book3" "Author3" "Publisher3" 20 5 59 (make-
reorder 0 50) 1/2)))
```

1.5.1 Test (equality)

daily-update should not change an empty inventory
Input:

```
(daily-update empty)
```

Expected Output:

```
empty
```

Expected Output Value:

```
()
```

Correct

1.5.2 Test (equality, 0.5 partial points)

daily-update should deduct one from each expected shipment time and leave books without outstanding reorders unchanged
Input:

```
(daily-update inventory-1)
```

Expected Output:

```
inventory-1-after-update
```

Expected Output Value:

```
(#(struct:book 15 "How to Design Programs" "Felleisen et al." "MIT
Press" 59 49 100 #(struct:reorder 1 5) 1/12) #(struct:book 16 "A
Game of Thrones" "George R. R. Martin" "Bantam" 12 5 15 #f 1/20))
```

Correct

1.5.3 Test (equality, 0.5 partial points)

daily-update should add incoming shipments to the stock and leave books without outstanding reorders unchanged
Input:

```
(daily-update inventory-2)
```

Expected Output:

```
inventory-2-after-update
```

Expected Output Value:

```
(#(struct:book 15 "How to Design Programs" "Felleisen et al." "MIT
Press" 49 39 52 #f 1/12) #(struct:book 16 "A Game of Thrones" "George
R. R. Martin" "Bantam" 12 5 15 #f 1/20))
```

Correct

1.5.4 Test (equality, 0.5 partial points)

daily-update should update shipments and stock according to their interpretation
Input:

```
(daily-update inventory-4)
```

Expected Output:

```
inventory-4-after-1-day
```

Expected Output Value:

```
(#(struct:book 53 "Book1" "Author1" "Publisher1" 100 50 14 #(struct:reorder 0 20) 1/4) #(s
```

Correct

1.5.5 Test (equality, 0.5 partial points)

daily-update should update shipments and stock according to their interpretation
Input:

```
(daily-update (daily-update inventory-4))
```

Expected Output:

```
inventory-4-after-2-days
```

Expected Output Value:

```
(#(struct:book 53 "Book1" "Author1" "Publisher1" 100 50 34 #f 1/4) #(struct:book 54 "Book2
```

Correct

2 Results

Successes: 24
Wrong Outputs: 0
Errors: 2
Achieved Points: 10
Total Points (rounded): 10/12