REVIEW 1&2
15 CSE 402 - SICP

TEAM NUMBER - 16

PROBLEM STATEMENT - 4

CB.EN.U4CSE17014	CH. SAI PHANI JASWANTH
CB.EN.U4CSE17043	NANDITHA MENON
CB.EN.U4CSE17049	RISHI VIGNESH
CB.EN.U4CSE17065	T. SAI SRIHITHA REDDY

PROBLEM STATEMENT

Assume that a local bookstore (that sells story books) has contacted you to provide an inventory system (which is a collection of information related to all the books) for their book website. Since you wanted to try your hand in functional programming paradigm, you chose to implement the inventory system in Racket.

You are fully aware that the first step towards realizing inventory system is to first abstract a book. The bookstore feels that the key information in the inventory system are the name of the book and its price. You have decided to implement the inventory system as a simple collection of these two information.

Once you have the inventory, finding a book in the inventory is an obvious facility you would like to provide. Having developed this far, the bookstore realized that number in stock and number of copies sold for each book are other key information to be incorporated into the inventory. So is the option to find the number of books in stock given a book name. You may have to re-design/re-write your inventory and provide a procedural abstraction for the same. Also, the bookstore wanted to have provisions to find out how many books are not in stock and how many books have not been sold at all (even a single copy).

Since every business sector has started to embrace analytics, the bookstore wanted you to add a genre for each book so that it can find out which genre is fast selling and worst selling. Similarly adding author and finding which author is fast selling and worst selling are also straightforward tasks.

The tasks you must do in this case study problem have all been provided in bold text. The quality of the solutions is assessed by the abstractions (both data and procedure) you have designed.

DATA ABSTRACTION

- book(title, (author, genre, price, noInStock, noSold))
    - This data structure contains 6 attributes: title, author genre, price, noInStock, noSold
    - Title - title of the book
    - Author- author of the book
    - Genre - genre of the book
    - Price - Price of the book
    - noInStock - number of books in stock under a given title
    - noSold - number of book sold under a given title

- inventory(list of book nodes)
    - This data structure contains the list of books
    - Each element in the list is a node of the data structure ‘book’

- genre-count(genre,count)
    - Each element in this data structure has 2 fields: genre, count
    - Gives the number of books sold in each genre

- author-count(author,count)
    - Each element in this data structure has 2 fields: author, count
    - Gives the number of books sold under each author

PROCEDURE ABSTRACTION

- find-book(inventory,title)
    - accepts ‘inventory’ and ‘title’ as parameters
    - returns pointer pointing to the book requested for.

- get-stock-count(inventory,title)
    - ‘inventory’ and ‘title’ as parameters
    - returns the number of books in stock under the title given
    - When the pointer reaches the end and does not find a match, this procedure will return “no such book”.

- get-sold-count(inventory,title)
    - ‘Inventory’ and ‘title’ as parameters
    - Returns the number of books sold fo the given title
    - When the pointer reaches the end and does not find a match, this procedure will return “no such book”

- find-not-in-stock-books-count(inventory)
    - Accepts ‘inventory’ as parameter
    - Iterates and returns the count of books which have noInStock value as 0
    - noInStock gives the count of books of a certain title. If 0, the store no longer has that title
- find-not-sold-books-count(inventory)
    - Accepts ‘inventory’ as parameter
    - Iterates through and returns the count of books which have noSold value as 0
    - noSold gives the count of books, under a certain title, that have been sold

- sell-book(inventory,title,count)
    - Accepts ‘inventory’, ‘title’ and ‘count’ as parameters
    - When the pointer reaches the end and does not find a match, this procedure will return “no such book”
    - If match is found:
        - check if noInStock>count,
            - if yes, noInStock-=count and noSold+=count
                - Get genre
                - Go to genre-count, iterate, find the genre
                - Increase genre-count by count
                - Get author
                - Go to author-count, iterate, find the author
                - Increase author-count by count
            - If no, return “count is more than what the store has”

- increase-stock(inventory, title, count)
    - Accepts ‘inventory’, ‘title’ and ‘count’ as parameters
    - When the pointer reaches the end and does not find a match, this procedure will return “no such book”
    - If match is found increase noInStock by count

- add-book(inventory, title, price, noInStock, author, genre)
    - Accepts ‘inventory’, ‘title’, ‘price’, ‘noInStock’, ‘author’, ‘genre’
    - Creates a new node ‘book’ with given details. noSold=0 by default
    - Appends inventory
    - Will check if the given author is already in ‘author-count’. Else append with given author’s name
    - Will check if the given genre is already in ‘genre-count’, Else append with given genre

- best-genre(genre-count)
    - Accepts ‘genre-count’ as parameter
    - Iterates and finds the title with highest count and returns the associated genre

- worst-genre(genre-count)
    - Accepts ‘genre-count’ as parameter
    - Finds title with lowest count and returns the associated genre

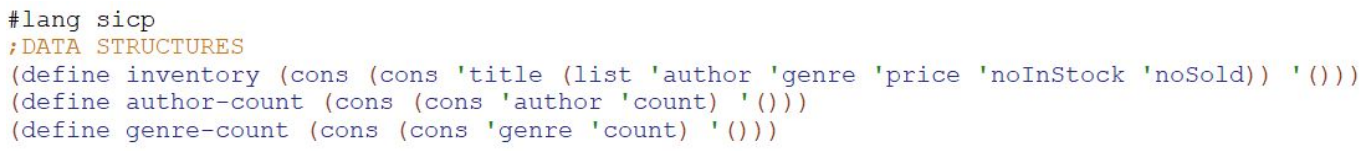
- best-author(author-count)
    - Accepts ‘author-count’ as parameter
    - Iterates and finds the highest count. Returns the associated author

- worst-author(author-count)
    - Accepts ‘author-count’ as parameter
    - Iterates and finds the lowest count. Returns the author name associated

- get-field-from-book(field, book)
    - Accepts ‘book’ and ‘field’ as parameter
    - Returns the given field from the given book

IMPLEMENTATION

DATA STRUCTURES

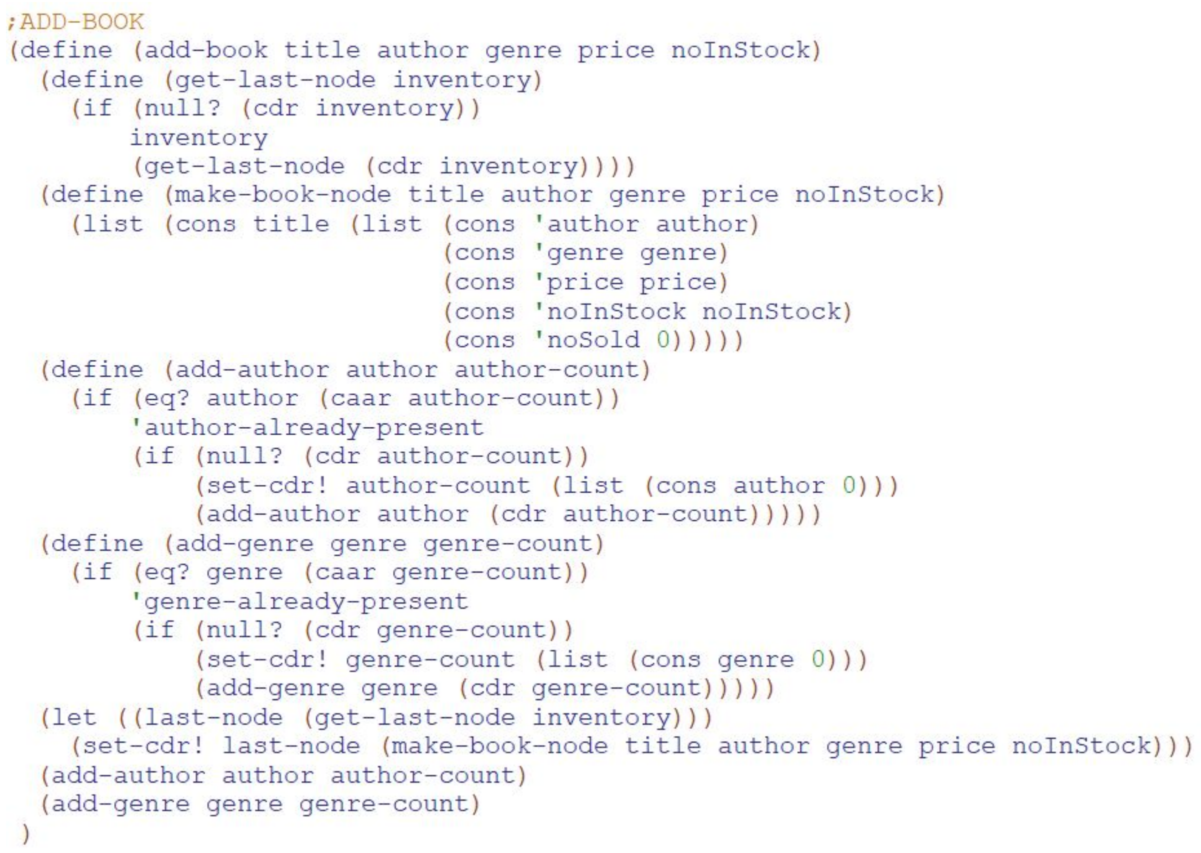


```
#lang sicp
;DATA STRUCTURES
(define inventory (cons (cons 'title (list 'author 'genre 'price 'noInStock 'noSold)) '()))
(define author-count (cons (cons 'author 'count) '()))
(define genre-count (cons (cons 'genre 'count) '()))
```

1. "inventory" stores each book in the format of [title, (author, genre, price, noInStock, noSold)]
   a. author - author of the book title
   b. genre - genre of the book
   c. price - price of the book
   d. noInStock - Number of copies of the book in stock
   e. noSold - Number of copies of the book sold
2. "author-count" stores each author available in the bookstore and the respective count of books sold in the format of [author, count]
3. "genre-count" , similar to author-count, stores each genre available in the bookstore and the respective count of books sold in the format of [genre, count]

PROCEDURES

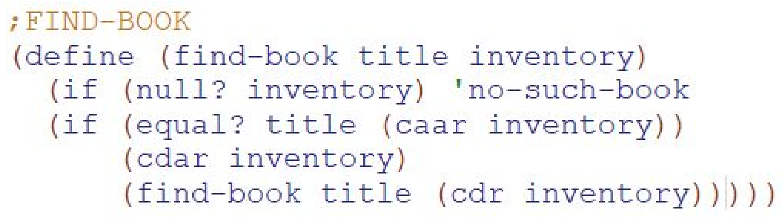
1. add-book



```
;ADD-BOOK
(define (add-book title author genre price noInStock)
  (define (get-last-node inventory)
    (if (null? (cdr inventory))
        inventory
        (get-last-node (cdr inventory))))
  (define (make-book-node title author genre price noInStock)
    (list (cons title (list (cons 'author author)
                            (cons 'genre genre)
                            (cons 'price price)
                            (cons 'noInStock noInStock)
                            (cons 'noSold 0)))))
  (define (add-author author author-count)
    (if (eq? author (caar author-count))
        'author-already-present
        (if (null? (cdr author-count))
            (set-cdr! author-count (list (cons author 0)))
            (add-author author (cdr author-count)))))
  (define (add-genre genre genre-count)
    (if (eq? genre (caar genre-count))
        'genre-already-present
        (if (null? (cdr genre-count))
            (set-cdr! genre-count (list (cons genre 0)))
            (add-genre genre (cdr genre-count)))))
  (let ((last-node (get-last-node inventory)))
    (set-cdr! last-node (make-book-node title author genre price noInStock)))
  (add-author author author-count)
  (add-genre genre genre-count)
 )
```

- The procedures - get-last-node, add-author, make-book-node, add-genre and add-author are all declared locally in the add-book procedure as they are not used anywhere else in the whole program.
- After iterating through inventory and getting the last node in the last-node variable, a book node is created through make-book-node procedure and is added to the inventory using the local binding procedure let and set-cdr! procedure.
- After adding the book node to the inventory, the current author is to be added to the author-count list using the add-author procedure. In the add-author procedure, it is first checked if the author is present in author-count. If not present, set-cdr! Is used to add the current new author to the author list.
- Similarly, the same is done with genre using add-genre procedure.

2. find-book

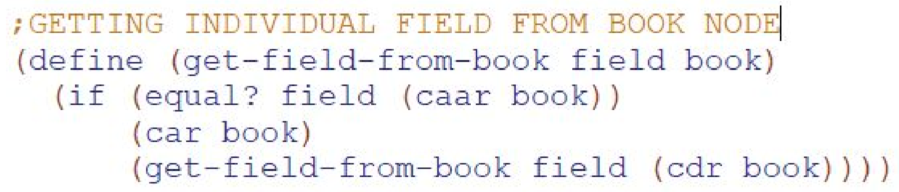


```
;FIND-BOOK
(define (find-book title inventory)
  (if (null? inventory) 'no-such-book
  (if (equal? title (caar inventory))
      (cdar inventory)
      (find-book title (cdr inventory)))))
```

- find-book recursively iterates through the inventory and searches for the book with the given title

- If the iteration reaches the end, the procedure return ‘no-such-book indicating there is no book with the given title in the inventory

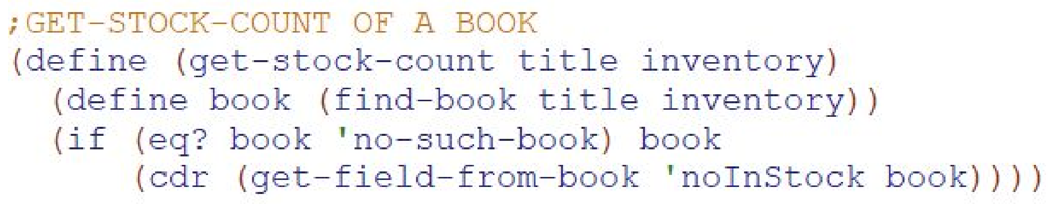
3. get-field-from-book



```
;GETTING INDIVIDUAL FIELD FROM BOOK NODE
(define (get-field-from-book field book)
  (if (equal? field (caar book))
      (car book)
      (get-field-from-book field (cdr book))))
```

- book here is the output of the find-book procedure which contains ((title, author, genre, price, noInStock, noSold))
- The above function simply extracts the given field by iterating recursively.
- The output is in the format of (field, value)

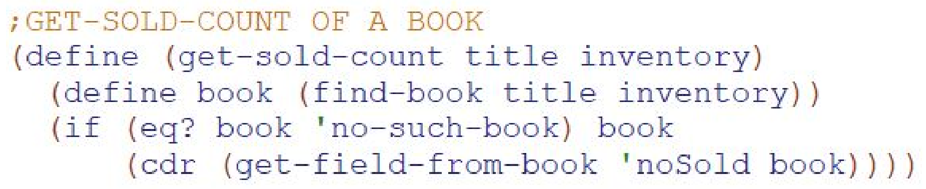
4. get-stock-count



```
;GET-STOCK-COUNT OF A BOOK
(define (get-stock-count title inventory)
  (define book (find-book title inventory))
  (if (eq? book 'no-such-book) book
      (cdr (get-field-from-book 'noInStock book))))
```

- This procedure returns the stock count a title in the inventory
- It uses the already defined find-book procedure to get the book node.
- If a book with the given title exists, get-field-from-book procedure is passed with the book and ‘noInStock as the parameters to get the stock count field - (‘noInStock, count) from the book node.
- Since the value is in the latter part of the field, cdr is used to directly get the stock count from the field.

5. get-sold-count



```
;GET-SOLD-COUNT OF A BOOK
(define (get-sold-count title inventory)
  (define book (find-book title inventory))
  (if (eq? book 'no-such-book) book
      (cdr (get-field-from-book 'noSold book))))
```

- Similar to get-stock-count, this procedure uses the same steps to get the ‘noSold field from the book node obtained from the find-book procedure.

6. increase-stock

```
;INCREASE STOCK COUNT OF A BOOK
(define (increase-stock title count inventory)
  (let ((book (find-book title inventory)))
    (if (eq? book 'no-such-book) book
        (let ((stock-field (get-field-from-book 'noInStock book)))
          (let ((current-stock (cdr stock-field)))
            (let ((new-stock (+ current-stock count)))
              (set-cdr! stock-field new-stock)))))))
```

- This procedure increases the stock count of a book with a given title by given count.
- Local binding procedure - let has been put to thorough use here to maintain code-readability and abstraction.
- After getting the stock count field from the book node obtained from the find-book procedure, the current stock count is first stored in the current-stock variable.
- Then, a new stock count is calculated and stored in the new-stock variable.
- This new-stock variable replaces the old stock count value from the stock count field using set-cdr! procedure.

CONCLUSION

1. Data abstraction and procedure abstraction completed.
2. 50% of the implementation also finished for review 1 and 2.
3. Used abstraction and local binding methods for clean code and readability