UNC COMP 590-145
Midterm Exam 3 Solutions

Monday, April 6, 2020
General comments

- Haven't quite finished grading E3 (sorry), but will by Wednesday.
- But let's walk through the solutions.
- Random question order and answer order
Write a Clojure protocol for git objects using `defprotocol`, including methods and parameters. Assume that the implementation will provide a way to access the type and data of a git object. (Hint: what does your program need to do with every git object, regardless of what type it is?)

```clojure
(defprotocol GitObject
  (address [_])
  (write-object [_]))
```
Exam 3 solutions

Question 4: Testing

Write a test for the following code using Speclj. Use a different namespace.

```clojure
(ns math)

(defn product
  "Compute the product of a function mapped over a sequence of numbers. example, (product inc [1 2 3]) is 24, and (product inc []) is 1." [f xs]
  (reduce * 1 (map f xs)))
```
(ns math-spec
  (:require [speclj.core :refer [describe it should=]]
    [math :as sut]))

(describe "product"
  (it "returns 1 when the sequence is empty"
      (should= 1 (sut/product inc [])))
  (it "returns the correct answer for a non-empty sequence"
      (should= 24 (sut/product inc [1 2 3]))))
Exam 3 solutions

Question 5: Implement a protocol

Implement the following protocol for lists (which have type `clojure.lang.PersistentList`). You may assume your code is in the same namespace as this code. (Hint: you may find the following core functions useful: `cons`, `empty?`, `first`, and `rest`.)

```clojure
(defprotocol Stack
  ;; A last-in-first-out stack data structure.
  (stack-push [__ item] "Push the given item onto the top of the stack.")
  (stack-pop [__] "Pop the top item off of the stack and return a sequence.
  (stack-empty? [__] "Return whether the stack is empty."))
```

Question 5: Implement a protocol

\[
\text{(extend-protocol Stack}
\]
\[
\text{clojure.lang.PersistentList}
\]
\[
\text{(stack-push [list item] (cons item list))}
\]
\[
\text{(stack-pop [list] [(first list) (rest list)])}
\]
\[
\text{(stack-empty? [list] (empty? list)))}
\]
I want to experiment with the Hiccup Clojure library in a REPL. What do I need to do before I can require the `hiccup.core` namespace? Be specific.

I need to add hiccup as a dependency to the project by adding `hiccup {:mvn/version "1.0.5"}` to the :deps map of the `deps.edn` file.
Exam 3 solutions

Question 7.1 (check all that apply)

A git tree object can include:

1. ( ) the sizes of the files in the corresponding directory
2. (X) the names of the files in the corresponding directory
3. ( ) the addresses of all descendant files under the corresponding directory
4. (X) the addresses of the blob objects for the files in the corresponding directory
Exam 3 solutions

Question 7.2 (check all that apply)

A git commit object can include:

1. ( ) the names of the top-level directory contents
2. ( ) the directory that the object database is stored in
3. (X) the address of the commit that this commit is based on
4. (X) the address of the tree object that the commit describes
Exam 3 solutions

Question 7.3 (check all that apply)

Which of these are good steps to take when debugging?

1. ( ) Read manuals until you understand the system well enough that you know what the bug is without investigating, a.k.a. "given sufficient understanding, all bugs are easy to solve."
2. ( ) Try the solution suggested in that Stack Overflow post, even if you don't understand it, just to see if it works.
3. (X) Trigger the bug.
4. (X) Take notes on what exactly you're doing.
Exam 3 solutions

Question 7.4 (check all that apply)

What's a good perspective when encountering a bug?

1. ( ) bugs happen because the computer is wrong or glitchy sometimes
2. (X) bugs are an opportunity to pursue mastery
3. ( ) bugs are not forward progress, so we should seek to minimize the time we spend fixing them
Question 7.5 (check all that apply)

1. (X) allow us to separate specification from implementation
2. ( ) are a mechanism to achieve inheritance
3. ( ) capture the differences between different kinds of nouns
4. ( ) must be paired with a noun in order to be meaningful
Exam 3 solutions

Question 8: git effects

Given the following sequence of idiot commands (with abbreviated addresses):

```
idiot init
idiot write-wtree  # outputs "d10e2581d043696388010f2049a2260fa2bdb184\n"
idiot commit d10e -m "Initial commit"  # commit address: cf169ae15476bed5990
idiot switch -c feature-x
# ...some work happens...
idiot write-wtree  # outputs "cabece4d68bd7327113931dd7f80534b52f1c31d\n"
idiot commit cabe -m "Add feature X" -p cf16  # commit address: 9ad9ee391696
idiot switch -c master
```
Exam 3 solutions

Question 8.1

What does HEAD contain? Be specific.

ref: refs/heads/master

UNC COMP 590-145
Midterm Exam 3 Solutions
Question 8.2

What does the master ref contain?

cf169ae15476bed5990fee10764e5b3c045b7893

16 / 26
Exam 3 solutions

Question 8.3

What does the \texttt{feature-x} ref contain?

\texttt{9ad9ee39169674ac4488ac757d97d985141e15ce}\n
Exam 3 solutions

Question 8.4

What is printed by the command `git cat-file -t cabe`?

tree

UNC COMP 590-145
Midterm Exam 3 Solutions
Exam 3 solutions

Question 8.5

What is printed by the command `idiot rev-list feature-x`? Be specific.

`9ad9ee39169674ac4488ac757d97d985141e15ce\ncf169ae15476bed5990fee10764`
Rewrite this code to use an implementation of the IFn protocol on something other than functions.

```clojure
(defn age-category->tax-message [age-category]
  (case age-category
    :minor "You're too young for income tax."
    :adult "Your new tax due date is July 15, 2020."))
```
Exam 3 solutions

Question 9: IFn implementations

```
(def age-category->tax-message
  {:minor "You're too young for income tax."
   :adult "Your new tax due date is July 15, 2020."})
```
Exam 3 solutions

Question 10.1 (multiple choice)

What is the foundational assumption in programming?

1. (X) The computer is only doing what it's told.
2. ( ) You can the computer work together to determine the right answer.
3. ( ) One can represent arbitrarily complex computations with ones and zeroes.
4. ( ) There's no wrong answer.
What is the essence of software design?

1. ( ) writing thorough automated tests
2. ( ) using good names and module boundaries
3. (X) eliminating incidental complexity
4. ( ) balancing indirection with abstraction
Question 10.3 (multiple choice)

What is the most important debugging tool?

1. ( ) print statements
2. ( ) the command-line interface
3. (X) your mind
4. ( ) the REPL
5. ( ) a stepwise debugger
Exam 3 solutions

Question 11: divide and conquer

If a bug is somewhere in this code, and you're applying the "divide and conquer" rule, list the 7 smaller places the bug could be. Assume that the `clojure.core` functions `keys` and `contains?` are bug-free.

```clojure
(let [names (keys people)]
  (if (contains? "Rich Hickey" names)
    (add-feeling self :awed)
    (carry-on self)))
```
Exam 3 solutions

Question 11: divide and conquer

1. the people value
2. how we're calling keys
3. how we're calling contains?
4. how we're calling add-feeling
5. the add-feeling function
6. how we're calling carry-on
7. the carry-on function