Programming Abstractions

Week 5-1: Exam 1 Review

Stephen Checkoway

Exam Format

6 implementation problems ("Write a procedure to do *x*")

1 extra credit problem

Write all of your solutions in DrRacket

Turn in your completed exam via Blackboard

Exam will be released at midnight on Thursday

Your solutions are due by 23:59 on Thursday

Class time

During Thursday's class, I will be in the class's Zoom meeting, feel free to hang out in there

If you have a question, send me a private chat either with the question itself or just say "I have a question" and I'll bring you into a breakout room and you can ask your question privately there

Possible question topics

Basic Scheme/Racket functions and special forms

- `cons`, `first` (`car`), `rest` (`cdr`), `list`, `append`, `member`, `empty?`, `filter`, etc.
- `define`, `lambda`, `if`, `cond`, `let`, `letrec`, `and`, `or`, etc.

`map` and `apply`

`foldl` and `foldr` and how they differ

Recursion

- Tail recursion
- "Accumulator passing style"

Closures: how to create and use them

Given a list `lst` and an element `x`, how can we create a new list that consists of `x` prepended to `lst`? E.g., if `lst` is `'(1 2 3)` and `x` is 4, we want '(4 1 2 3)

A. `(prepend x lst)`

B. `(cons x lst)`

C. `(append x lst)`

D. It's not possible to modify `lst`

E. None of the above

Given a list `lst` and an element `x`, how can we create a new list that consists of `x` appended to `lst`? E.g., if `lst` is `'(1 2 3)` and `x` is 4, we want '(1 2 3 4)

A. `(cons lst x)`

B. `(append lst x)`

C. `(append lst '(x))`

D. `(append lst (list x))`

E. None of the above

Given a list of lists lsts, how do you get a list containing the second element of each list, in order?

A. `(map second lsts)`

B. `(map rest lsts)`

C. `(apply second lsts)`

D. `(apply rest lsts)`

E. None of the above

Example

Represent a student as a three-element list (name, year, gpa), e.g., `'("Jane" 2 3.5)` represents Jane who is a second-year and has a 3.5 GPA

Write a procedure `(select lst)` that takes a list of students and returns the name of all second or third year students with a GPA that's at least `3.0`

Enumerate

Write a recursive procedure `(enumerate lst)` that takes a list and returns a list of 2-element lists `(index elem)` where `elem` is in `lst` and `index` is its index, in order.

E.g., `(enumerate '(a b c))` returns `'((0 a) (1 b) (2 c))`

Tail-recursive enumerate

Write a **tail-recursive** procedure `(enumerate2 lst)` that takes a list and returns a list of 2-element lists `(index elem)` where `elem` is in `lst` and `index` is its index, in order.

E.g., `(enumerate2 '(a b c))` returns `'((0 a) (1 b) (2 c))`

Flip

Write a procedure `(flip f)` that that takes a 2-argument procedure `f` and returns a 2-argument closure that, when called, calls `f` with its arguments in the opposite order. I.e., `((flip f) x y)` is the same as `(f y x)`

Write `(flip* f)` that takes any procedure f and returns a closure that, when called, calls f with all of its arguments reversed. E.g.,

- `((flip* f))` is `(f)`;
- `((flip* g) x)` is `(g x)`;
- `((flip* h) x y)` is `(h y x)`;
- `((flip* i) x y z)` is `(i z y x)`; and so forth

Compose two functions

Write a procedure `(compose f g)` that takes two 1-argument procedures f and g and returns a procedure that when called with the argument `x` returns `(f (g x))`

Reverse a structured (non-flat) list

Write a procedure `(reverse-all lst)` that takes a non-flat list and reverse it, including all contained lists

E.g., `(reverse-all '(1 () (2 3 (4 5)) 6))` returns
`'(6 ((5 4) 3 2) () 1)`