CS 61A
Fall 2020

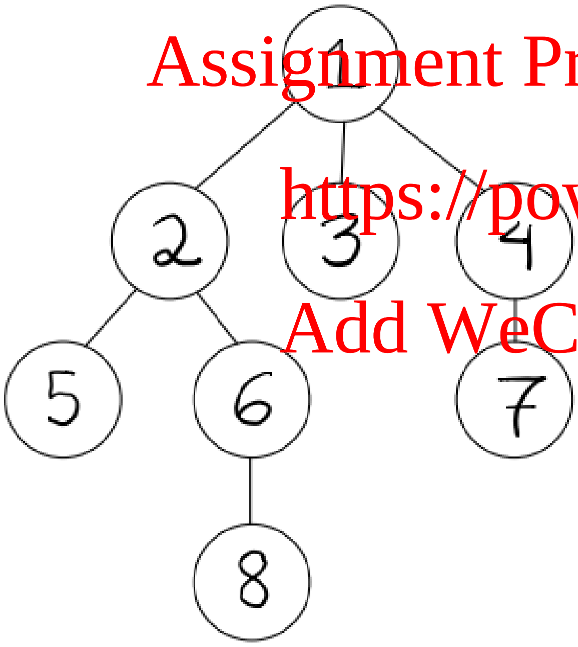
Structure and Interpretation of Computer Programs

Quiz 9 Solutions

INSTRUCTIONS

- Please review this worksheet before the exam prep session. Coming prepared will help greatly, as the TA will be live solving without allocating much time for individual work.
- Either Sean or Derek will be on video live solving these questions. The other TA will be answering questions in the chat. It is in your best interest to come prepared with **specific** questions.
- This is not graded, and you do not need to turn this in to anyone.

Below is a tree, which will be referred to as `t1` in future questions.



1. **Reverse It!**

   Implement the `reversed` procedure in Scheme, which takes a list `lst` and returns a Scheme list with the same elements in reverse order. You may not use `append`.

```
(define (reversed lst)

  (define (helper a b)

    (if (null? a) b

        (helper (cdr a) (cons (car a) b))))

  (helper lst nil))

scm> (reversed '())
()
scm> (reversed '(1))
(1)
scm> (reversed '(1 3 2 5))
(5 2 3 1)
```

2. **Slice It!**

Implement the `get-slicer` procedure in Scheme, which takes integers `a` and `b` and returns an *a-b slicing function*. A `a-b` slicing function takes in a list as input and outputs a new Scheme list with the values of the original scheme list from index `a` (inclusive) to index `b` (exclusive).

Your implementation should behave like Python slicing, but should assume a step size of one with no negative slicing indices.

```
(define (get-slicer a b)

  (define (slicer lst)

    (define (slicer-helper c i j)

      (cond ((or (= j 0) (> i j) (null? c)) nil)

            ((= i 0) (cons (car c) (slicer-helper (cdr c) i (- j 1))))

            (else (slicer-helper (cdr c) (- i 1) (- j 1)))))

    (slicer-helper lst a b))

  slicer)
```

```
scm> (define a '(0 1 2 3 4 5 6))
a
scm> (define one-two-three (get-slicer 1 4))
one-two-three
scm> (define one-end (get-slicer 1 10))
one-end
scm> (define zero (get-slicer 0 1))
zero
scm> (define empty (get-slicer 4 4))
empty
scm> (one-two-three a)
(1 2 3)
scm> (one-end a)
(1 2 3 4 5 6)
scm> (zero a)
(0)
scm> (empty a)
()
```

3. **Find It!**

   Implement the `pather` procedure in Scheme, which takes a Scheme tree `t`, a target value `goal` and returns a Scheme list containing the node values on a path from the root of `t` to a node containing `goal`. If no such path exists, `pather` returns an empty Scheme list.

   Complete `path-helper` for ease of implementation.

   Below is the Scheme tree ADT:

```
(define (tree label branches) (cons label branches))
(define (label t) (car t))
(define (branches t) (cdr t))
(define (is-leaf t) (null? (branches t)))

(define (pather t goal)

  (if (eq? (car t) goal)

        (list goal)

        (let ((path (path-helper (branches t) goal)))

                 (if (null? path) nil

                                  (cons (label t) path)))))


(define (path-helper bs goal)

  (if (null? bs)

        nil

        (let ((path (pather (cars bs) goal)))

                (if (not (null? path)) path

                                       (path-helper (cdr bs) goal)))))




scm> (pather t1 7)
(1 4 7)
scm> (pather t1 1)
(1)
scm> (pather t1 12)
()
```