CS 61A Midterm #3 - November 9, 1998

61A MidtermCS 61A Midterm #3 - November 9, 1998

Your name

login cs61a-

Discussion section number

TA's name

This exam is worth 20 points, or about 13% of your total course grade. It includes two parts: The individual exam (this part) is worth 17 points, and the group exam is worth 3 points. The individual exam contains five substantive questions, plus the following:

Question 0 (1 point): Fill out this front page correctly and put your name and login correctly at the top of each of the following pages.

This booklet contains seven numbered pages including the cover page. Put all answers on these pages, please; don't hand in stray pieces of paper. This is an open book exam.

When writing procedures, don't put in error checks. Assume that you will be given arguments of the correct type.

Question 1 (3 points):

What will the Scheme interpreter print in response to each of the following expressions? Also, draw a ``box and pointer'' diagram for the result of each expression. Hint: It'll be a lot easier if you draw the box and pointer diagram *first* !

(let ((x (list 1 2 3 4))) (set-car! x (cdr x)) x)

(let ((x (list 1 2 3 4))) (set-car! (cdr x) (cadr x)) x)

(let ((x (list 1 2 3 4))) (set-cdr! (cddr x) (cdr x)) x)

Your name login cs61a-

Question 2 (3 points):

(a) Suppose we say

> (define baz 10) > (define s (make-serializer))

> (parallel-execute (s (lambda () (set! baz 7))) (s (lambda () (set! baz (+ baz baz)))))

What are the possible values of `baz` after this finishes?

(b) Now suppose that we change the example to use a separate serializer for each process, as follows:

> (define baz 10) > (define s (make-serializer)) > (define t (make-serializer))

> (parallel-execute (s (lambda () (set! baz 7))) (t (lambda () (set! baz (+ baz baz)))))

What are the possible values of `baz` this time?

Question 3 (4 points):

We are going to prepare a simulation of an FM car radio. To simplify the problem we'll restrict our attention to tuning, not to volume or balance or anything else a radio does. This radio features digital tuning. There are six buttons that can be preset to particular stations; for manual tuning, there are `up` and `down` buttons that move to the next higher or lower frequency. (FM frequencies are measured in megahertz and have values separated by 0.2: 88.1, 88.3, 88.5, 88.7, 88.9, 90.1, etc.) To simplify the problem further, we'll ignore the boundary problem of what to do when you're at the lowest assigned FM frequency and try to go `down` below that frequency. Just pretend you can keep going up or down forever.

Use the OOP language (`define-class` and so on).

(a) Create a `button` object class that accepts these messages:

set-freq! 93.3 sets the button's remembered frequency freq returns the remembered frequency

The initial frequency should be zero (because the buttons don't have settings initially).

This question continues on the next page.

Your name login cs61a-

Question 3 continued:

(b) Create a `radio` object class that has six buttons, numbered 0 through 5, and accepts these messages:

set-button! 3 sets button 3 to the radio's current frequency push 3 sets the radio to button 3's frequency up sets the radio to the next higher frequency down sets the radio to the next lower frequency freq returns the radio's current frequency

The radio's initial frequency should be 90.7 MHz. Points to remember: Your radio has to use six of your button objects; you needn't check for invalid argument values in the methods. **Hint:** Give your radio a list of six buttons, and use `list-ref` to get the one you want.

Question 4 (3 points):

Consider the following variation on the sieve of Eratosthenes:

(define (digit-sieve s) (cons-stream (stream-car s) (digit-sieve (stream-filter (lambda (x) (not (member? (stream-car s) x))) (stream-cdr s)))))

(define digit-stream (digit-sieve integers))

Remember, numbers are considered words, so `member?` will check for digits in a number.

(a) What are the first four elements of `digit-stream`?

(b) What is the result of `((repeated stream-cdr 10) digit-stream)`?

Your name login cs61a-

Question 5 (3 points):

You are given a binary tree, in which the nodes are represented in the form indicated by these selectors:

(define datum car) (define left-branch cadr) (define right-branch cddr)

The empty tree is represented by the empty list.

Write `traverse!`, a procedure that takes a binary tree as its argument, and rearranges the pairs to form an inorder traversal - a linear sequence of the data from the tree, in left-to-right order. (If the tree is a binary search tree, for example, then the result will be a sorted sequence.)

A binary tree with N nodes contains 2N pairs. You will string together the pairs containing the data, and discard the pairs containing the branch pointers (after you've collected the data from those branches).

Note: **Do not allocate any new pairs** in your solution. Modify the existing pairs.

Note: In this problem you are changing a structure's abstract data type, from tree to sequence. In such situations, data abstraction doesn't make much sense; just use `car`, `set-cdr!`, etc.