132

Computer Science 61A, Fall 2004 University of California, Berkeley

Exam 2A November 8, 2004 8-10PM

This is an open-book test. You have approximately two hours to complete it. You may consult any books, notes, or other paper-based inanimate objects available to you. To avoid confusion, read the problems carefully. If you find it hard to understand a problem, read it again, and do your best to answer it. If you think you have found a typographical error come to the front or the side of the room to ask about it. It would not be fair to give you extra personalized help during the test in understanding a question, and so we won't do it.

Partial credit may be given for wrong answers.

Note that **some** questions on this exam offer the option to "punt". This is a way for you to receive partial credit for recognizing that you do not know the answer. For these questions, you have two choices. You may supply an answer, which, as usual, will receive anywhere between 0% and 100% of the points depending on how correct it is. Alternatively, you may check "punt" in which case you will receive 20% of points specified in exchange for us not having to grade the question.

Your exam should contain 8 problems (numbered 0 through 7) on 10 pages. Please write your answers in the spaces provided in the test. DO NOT START UNTIL WE TELL YOU TO BEGIN.

Question	MAX POINTS	YOUR POINTS	punt
0	1		
1	15		
2	15		
3	5		
4	12		
5	5		
6	14		
7	8		
TOTAL	75		

0. [1 point] Your first name _____ Your last (family) name_____
Your Teaching Assistant's name _
The day and time that your discussion section meets _

The seat number of your seat _____.
The row number (we will help you with this, later) _____

CLEARLY PRINT YOUR CS61A LOGIN ON EVERY PAGE!!!! **11/8; 9AM**

1. Here are two programs, **filter** and **limit**, along with minor variations on them. You have already seen the first version in lecture. We point out places where programs differ from earlier ones by lines marked with ;;*.

```
(define (filter pred? L)
  (define (loop L)
    (if (null? L) '()
       (let ((hd (car L))
             (rst (cdr L)))
         (if (pred? hd)
             (cons hd (loop rst))
             (loop rst)))))
  (loop L))

;; list of all elements in L less than max
(define (limit L max)
  (filter (lambda(x)(< x max)) L))
;;.......................

(define (filter2 pred? L)
  (define (loop L)
    (if (null? L) '()
       (let ((hd (car L))
             (rst (cdr L)))
         (if (< hd max)  ;;*
             (cons hd (loop rst))
             (loop rst)))))
  (loop L))

(define (limit2 L max)
  (filter2 (lambda(x)(< x max)) L))
;;.......................

(define (filter3 max L)
  (define (loop L)
    (if (null? L) '()
       (let ((hd (car L))
             (rst (cdr L)))
         (if (< hd max)
             (cons hd (loop rst))
             (loop rst)))))
  (loop L))

(define (limit3 L max)  (filter3 max L)) ;;*
;;.......................

(define (filter4 L)  ;;*
  (define (loop L)
    (if (null? L) '()
       (let ((hd (car L))
             (rst (cdr L)))
         (if (< hd max)
             (cons hd (loop rst))
             (loop rst)))))
  (loop L))

 (define (limit4 L max)   (filter4 L)) ;;*
;;.......................
```

```
(define (filter5 pred? L) ;;*
  (define (loop L)
    (if (null? L) '()
        (if (pred? (car L)) ;;*
            (cons (car L)(loop (cdr L))) ;;*
            (loop (cdr L))))))) ;;*
  (loop L))

(define (limit5 L max)
  (filter5 (lambda(x)(< x max)) L))

;;.....................

(define (filter6 pred) ;;*
  (define (loop L)
    (if (null? L) '()
        (if (pred)  ;;*
            (cons (car L) (loop (cdr L)))
            (loop (cdr L))))))
  (loop L))

(define (limit6 L max)
  (filter6 (lambda ()(< (car L) max)))) ;;*
```

The questions consist of statements that may be **T [true],** or **F [false]** , *or need some explanation.* You must answer the question, as indicated. You should check (**F [false] describe**) if you believe a program does not return an answer but stops with an error. Then for full credit *you must describe clearly the reason for the error.* If the error has to do with the environment explain why in complete English sentences. You may draw a picture, but only to supplement, not replace, your explanation.

Some questions have another alternative (**punt**): Read the instructions on the cover page. A "punt" is worth 20%.

1A. Does limit work as described in the comment?
___Yes, (limit '(1 2 3 4 5) 4) returns (1 2 3).
___Yes, (limit '(1 2 3 4 5) 4) returns (1 2 3 4).
___Yes, (limit '(1 2 3 4 5) 4) returns (4 3 2 1).
___Yes, (limit '(1 2 3 4 5) 4) returns (4 5).
___ No, it does something else (describe) _____
___ I don't know (punt)

1B. limit6 returns the same value as limit.
___T
___F Describe
___F Punt

1C. limit5 returns the same value as limit.
___T
___F Describe
___F Punt

1D. limit4 returns the same value as limit.
___T
___F Describe
___F Punt

1E. limit3 returns the same value as limit.
___T
___F Describe
___F Punt

1F. limit2 returns the same value as limit.
___T
___F Describe
___F Punt

1G. **filter5** might take a much longer time using **eval-1** than the other filters because it calls (car L) twice and L might be very long. (T/F _)
1H. **loop** in **filter2** runs an iterative process. (T/F _)

1I. **f1** and **f2** below seem to return the same values for any L, a list of integers. Why?

```
(define (f1 L) (accumulate + 0 (filter odd? (map square L))))
(define (f2 L) (accumulate + 0 (map square (filter odd? L))))
```

2. Sometimes it's difficult to remember the order of arguments to a procedure. Consider the following contrived function:

```
(define (f a b c)
  (list a (list b) c))
```

It is designed to be called like this:

```
STk> (f 1 2 3)  ;; a=1 b=2 c=3
(1 (2) 3)
```

We'd like to create a function **g** that is just like **f** except that each argument to **g** is preceded by a symbol which we shall call a keyword. This keyword makes explicit the parameter with which the actual argument is to be associated, eliminating the need to remember the exact order of the parameters:

```
STk> (g 'a 1 'b 2 'c 3)
(1 (2) 3)
STk> (g 'b 2 'a 1 'c 3)
(1 (2) 3)
STk> (g 'c 3 'b 2 'a 1)
(1 (2) 3)
```

The goal is to write a function **make-keyword-proc** that takes two arguments. The first is a regular Scheme procedure like **f**. The second is a list of the keywords in the order that the procedure expects. **make-keyword-proc** should return a function like **g** that takes keyword arguments and calls **f** with the arguments in the order **f** expects. Continuing the example above, to create the function **g** from **f** we can use make-keyword-proc like this:

```
STk> (define g (make-keyword-proc f '(a b c)))
```

The keywords do not have to match the names of the formal parameters of f:

```
STk> (define g2 (make-keyword-proc f '(cs61a cool is)))
STk> (g2 'cs61a 1 'is 2 'cool 3) ;;note, keyword order irrelevant in call
(1 (2) 3)
```

Additionally, a call like

```
STk> (g2 'is 2 'cs61a 1 'cool 3 'notcool 4)
```

must work because `notcool` is not a keyword and doesn't affect the call to **f**. Your program should work so

```
STk> (g2 'cs61a 1)
```

will call **f** on arguments `1, #f, #f.`

2A. First write a program, call it **evenmember**, with the following specification: ___Punt (20%)

```
;; given an element x and a list L whose length is even,
;; return the sublist of L beginning with the first value eqv? to x,
;; checking only positions 0, 2, 4 ..  Examples
;;(evenmember 'b '(a x b y c z)) is  (b y c z)
;;(evenmember 'b '(a b b y c z)) is  (b y c z)
;;(evenmember 'b '(a b c d)      is ()

(define (evenmember x L)
```

2B. Next, using **evenmember**, write **make-keyword-proc**. In order to keep your program short, assume that the keyword argument list supplied is of even length, alternating keynames and values Punt (20%)

```
(define (make-keyword-proc proc keylist)
  (lambda args  ;note not  lambda(args). Uses variable number of arguments
```

)

2C. Finally, assume that we want to build keywords in to **eval-1**, and implement keyword argument functions by a new special form similar to **lambda**, but called **keyword-lambda**, say as `(define f (keyword-lambda(a b c) ….)`

How would you change **eval-1** to accommodate this? You should use one or more complete English sentences.
Describe
Punt (20%)

3. In midterm 1 we asked about a version of lisp in which there were no pairs, but just triples that had three components. One common response was to point out that we can already make triples without rewriting lisp. But now we know of a strange encoding as programs. We can define a tree-cons this way

```
(define (tree-cons label left right)
  (lambda(which)
      (cond ((= which 0) label)
            ((= which 1) left)
            ((= which 2) right)
            (else (error "illegal access to tree-cons)))))
```

3A. Show us you understand this representation by defining the accessor function `for left-tree.`

3B. Also explain why the predicate **tree-cons?** would be difficult to write, using a complete English sentence.

(or punt____): get 20% for saying "I have no idea"

4. (Option: you can punt all of question 4 _____ for 20%)

4A. Define a function **every-nth** that takes two arguments integer k and integer $n > 0$, and returns a *stream*: the infinite stream of numbers beginning at the specified integer k and whose succeeding elements are $k+n$, $k+2n$, $k+3n$, ... For example, if k=0 and n=3, the stream will begin with 0, 3, 6, 9, 12, 15, 18.

4B. Define a stream **r1** of all non-negative integer multiples of 3.

Define another stream **r2** of all non-negative integer multiples of 5.

4C. Define a function **both-streams** that given two streams s1 and s2 each of (increasing) numbers, returns a new stream s3 of those numbers that appear in both s1 and s2. For example, **(both-streams r1 r2)** would begin with 0, 15, 30, ...

4D. Are there any circumstances in which **both-streams** will not return? Explain, using examples as appropriate.

5. You are familiar with **set!** and **set-car!**. One of them is a special form and the other is not. Explain why this is the case in complete English sentences.

6. Henry wants to design a Scheme bank account, protecting his account with a password or personal identification number (PIN). *He is not as good a programmer as you are* and asks you to help him. He has written a first draft of his **new-account** program below. Be sure to notice that it is different from the one you have seen before. You should be able to help him to **rewrite it all in one new version.**
(or you can punt the whole question _____ for 20%)

6A. Enhance the program in a minimal fashion so that he can only take money out if he remembers the correct PIN.
What is a command to extract $10?

6B. Enhance the program below so that he can change the PIN.
What is a command to change the PIN to 1234?

6C. Perhaps because of a design error, anyone who has (a pointer to) Henry's account can find out the balance without knowing Henry's PIN. (Hint: compare the **deposit** and **withdraw** programs). What command would you use in the given program to find out Henry's balance?

6D. Change the program to protect Henry's privacy, but add a feature so that Henry's benefactors (perhaps his parents) can deposit money, *without knowing his PIN, or finding out his balance*. Write such a program, **give-money** so that (**give-money HA 100**) where HA is Henry's account, will deposit 100 dollars in his account.

6E. Finally, change the program so that the numerical sign on a negative monetary amount is ignored. That is, it is impossible to deposit or withdraw a negative amount of money; only the positive value is used.

```
(define (new-account balance inputPIN) ; PIN is personal identification number

  (define (withdraw amount PIN)
    (set! balance (- balance amount)) balance)

  (define (deposit amount)
    (set! balance (+ balance amount)) balance)
  (define (respond-to-msg msg)
     (cond
       ((eq? msg 'withdraw) withdraw)
        ((eq? msg 'deposit) deposit) ) )
 respond-to-msg)
```

7. In eval-1, there is no "set!". Add it. (or punt___ for 20%)
You may wish to base your answer on the following program already included in eval-1.

Check here if you have tried to do this problem previously either during a discussion section, review session, or with friends _____.

```
....
(define (lookup-variable-value var env0 env1)
   (let ((match (or (assq var env0) (assq var env1))))
       (if match (cdr match)
          (error "Undefined variable: " var))))
```

and by recalling this part of eval

```
((eq? kind 'symbol)
 (lookup-variable-value exp
                  inner-env outer-env))
```