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

Exam 1A October 4, 2004 8-10PM **SKETCH OF SOLUTIONS**

1. [19 points total] Consider these functions

```
(define (ga k)(if (= k 0) '()(cons k (ga (- k 1)))))

(define (gb k)
  (define (g1 p k)
    (if (= k 0) '()(cons p (g1 p (- k 1)))))
 (g1 k k))

(define (gc k)
  (define (g2 p k)
    (if (= k 0) '()(cons (- p 1) (g2 p (- k 1)))))
 (g2 k k))

(define (gd k)
  (define (g3 p k)
    (if (= k 0) p (g3 (cons k p) (- k 1))))
 (g3 k k))

(define (ge k)
    (define (g4 p k)
        (if (= k 0) '()(cons (ga p)(g4 p (- k 1)))))
 (g4 k k))
```

a. Choose from the following table the best response, A –H : What is returned from executing each of these expressions? (choices may be used more than once.) [10 points]

(ga 5)	(5 4 3 2 1) = E
(gb 5)	(5 5 5 5 5) = C
(gc 5)	(4 4 4 4 4) = G
(gd 5)	(1 2 3 4 5 . 5) = H
(ge 5)	((5 4 3 2 1)(5 4 3 2 1)…) = H

A	(0 1 2 3 4 5)
B	(1 2 3 4 5)
C	(5 5 5 5 5)
D	(5 4 3 2 1 0)
E	(5 4 3 2 1)
F	((-5 1) (-5 2)(-5 3)(-5 4) (-5 5))
G	(4 4 4 4 4)
H	None of the above

b. b. Draw a circle around the name of each function that is recursive:
(ga, g1, g2, g3, g4). [2 points**]**

These are all recursive programs.

c. Draw a circle around the name of each function that, when run, generates an iterative process:

(ga, gb, gc, gd, ge). [2 points] **Just gd.**

d. What is the running time of each of the functions? Choose from this table (answers may be used more than once.) [5 pts]

ga	J
gb	J
gc	J
gd	J
ge	L

I	Θ (p)
J	Θ (k)
K	Θ (p^2)
L	Θ (k^2)
M	Θ (log(k))
N	Θ (log(p))
P	None of the above

Note that if you were looking for log(n), you missed part of the concept of choosing a parameter that characterizes the size or complexity of the input, and that is allowed to grow.

Question 2. [10 points] Word switch
Your task is to finish writing a short program to translate from “netlingo” to English. For example, you know the abbreviations LOL = laughing out loud; AFAIK = as far as I know; OTOH = on the other hand.

```
(define dictionary
  '((LOL laughing out loud)
    (AFAIK as far as I know)
    (OTOH on the other hand)))

(define sample1 '(The textbook is correct AFAIK))
(define sample2 '(OTOH I never even opened it LOL))

(define (translate sent dict)
      define (translate sent dict)
      (cond ((null? sent) '())
            ((assoc (car sent) dict) ;; complete the program below
              (append (cdr (assoc (car sent) dict))  ;;* see below
                (translate (cdr sent) dict))))

              (else (cons (car sent)(translate(cdr sent) dict)))))

;; like 1 point per underline + 3 points if assembled correctly
```

Example: (translate sample2 dictionary) produces
 (on the other hand i never even opened it laughing out loud)

3. [3 points] Consider `(define (f a b c)(if (b a 0)(c a) a))`

Explain in a single English sentence what (f **x** < -) computes, if **x** has a numeric value.

If the bindings for < and – are unchanged from the defaults, then (f x < -) computes the same as (if (< x 0)(- x) x) or absolute value of x. If x is unbound or not a number, it will produce an error.

4. [3 points] threelisp
Ordered trees seem to benefit from a representation in which three items are used: (label left right) as a typical representation in lisp. A programming language designer proposes a version of lisp in which there are no pairs, but triples. That is, instead of (cons a b) there is (treecons a b c), and instead of car, cdr, there are accessor functions for the part of a tree t, (label t)(left t)(right t). Write one or more complete English sentences supporting one of these positions. (Circle which position you are supporting.)

a. This is a great idea.
b. This is a terrible idea.
c. This is not even an idea.

(a): Trees would be more compactly represented, and some accommodation can be made to store lists (say, s-expressions constituting lisp programs!) by ignoring the third item in a triple, say (cons a b) = (treecons a b nil).
So you could do it. Historical note, a system called TREET was designed by E.C. Haines, using this idea. 1965.
(b): You can already represent triples by lists of three items, so no real advantage is obtained this way, at least abstractly. By not having pairs, representing lists of one or two items, or four or more items becomes somewhat irregular. You can't use triples to represent symbolic expressions representing lisp programs which are essentially lists, usually longer than 3.
(c): Same as (b).
Some people muddled the ideas of abstraction and representation here. The proposal here was a different REPRESENTATION. A good argument to not support this representation is that the abstraction of trees does not really need it, and in particular, other kinds of trees with more than 2 sub-trees, are kind of inconvenient! Some people added irrelevant or outright wrong statements to an initially correct answer, and ended up with fewer points.

5. [8 points] (evaluate)
Here is a self-contained but not very complete evaluator program.

```
(define (evaluate exp binding)
  (cond ((number? exp) exp)
        ((symbol? exp) (cadr (assoc exp binding))) ;get val from
binding
        ((eqv? (car exp) '+) (+ (evaluate (cadr exp) binding)
                                (evaluate (caddr exp)binding)))
        ((eqv? (car exp) '*) (* (evaluate (cadr exp) binding)
                                (evaluate (caddr exp)binding)))
        (else exp)))
```

What results from executing each of the following pieces of code?

`(evaluate 4)`	**error**
`(evaluate 4 '())`	**4**
`(evaluate 'a '((a 3)))`	**3**
`(evaluate '(+ a (* b c)) '((a 1)(b 2)(c 3)))`	**7**
`(evaluate '(+ a (* b c)) '((a 1)(b 2)))`	**error**
`(evaluate '(foo 3 (+ a b)) '((foo *)(a 1)(b 2)))`	**(foo 3 (+ a b))**

6. [10 points]
For this question you will not write a program. Just draw some pictures in the space below.

a. The following elements are inserted into a binary search tree, in order. That is, element 5 is inserted first. Draw a sequence of 8 trees, each showing the resulting tree after the insertion of a new element.

5 7 9 6 10 2 3 1 *(Version B of the test had a different sequence)*
We don't draw the whole sequence, just the end product.

b. How deep is the tree? (A tree with one node is defined as depth 0).

3

c. What is the depth of the most balanced binary search tree it is possible to construct using these same elements? **3**

```
         5
      2     7
    1   3  6   9
                10
```

7. [6 points total]

In week 4 homework you were introduced to a representation for a hierarchical file system. We are going to use the same representation except instead of using (file g) for the file named g, we will use a list of length 3: (file g 100) which means that the file named g is of length 100.

The list `t1` below, is an example file directory.

```
(define t1 '(directory a
                   (file b 100)
                   (file c 200)
                   (directory c
                          (file d 10)
                          (file e 10))
                   (file f 300)))
```

a. What value does this program return, when called on this example, (YYYY t1)? [2 points]
(300 10 10 200 100)
Version B of the test added the numbers instead of consing them

b. Describe in terms of the directory structure, what r is when the line with comment `;;*` is executed, and what the value of the second argument to count2 is. [4 points]

r is a list or sub-list of files or subdirectories of a directory.
The value of the second argument is the accumulated list of file sizes *(or the sum of file sizes)* up to this time, not yet examining the sublist (cdr r) of the list of files or directories.

```
(define (YYYY r)              ; r is a directory
  ;; we define count-top, which itself has 2 subroutines
  (define (count-top fun)

    (define (count1 r sofar)
     ;; called only on (directory ...)  or (file ....)
      (if (eqv? (car r) 'directory)  (count2 (cddr r) sofar)
       ;; otherwise (car r) is 'file
        (cons (fun r) sofar)))  ;; fun is passed in to count-top

    (define(count2 r sofar)
     ;; called only on ((file ...)..) or ((directory) ...) or ()
      (if (null? r) sofar
        (count2 (cdr r) (count1 (car r) sofar)))) ;;*
    ;; this next line, inside count-top, starts the processing
    ;; of a directory
    (count1 r '()))
  ;; this next expression is inside YYYY
   (count-top caddr) )
```

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

Exam 1A October 4, 2004 8-10PM **SKETCH OF SOLUTIONS**

1. [19 points total] Consider these functions

```
(define (ga k)(if (= k 0) '()(cons k (ga (- k 1)))))

(define (gb k)
  (define (g1 p k)
    (if (= k 0) '()(cons p (g1 p (- k 1)))))
 (g1 k k))

(define (gc k)
  (define (g2 p k)
    (if (= k 0) '()(cons (- p 1) (g2 p (- k 1)))))
 (g2 k k))

(define (gd k)
  (define (g3 p k)
    (if (= k 0) p (g3 (cons k p) (- k 1))))
 (g3 k k))

(define (ge k)
    (define (g4 p k)
        (if (= k 0) '()(cons (ga p)(g4 p (- k 1)))))
 (g4 k k))
```

a. Choose from the following table the best response, A –H : What is returned from executing each of these expressions? (choices may be used more than once.) [10 points]

(ga 5)	(5 4 3 2 1) = E
(gb 5)	(5 5 5 5 5) = C
(gc 5)	(4 4 4 4 4) = G
(gd 5)	(1 2 3 4 5 . 5) = H
(ge 5)	((5 4 3 2 1)(5 4 3 2 1)…) = H

A	(0 1 2 3 4 5)
B	(1 2 3 4 5)
C	(5 5 5 5 5)
D	(5 4 3 2 1 0)
E	(5 4 3 2 1)
F	((-5 1) (-5 2)(-5 3)(-5 4) (-5 5))
G	(4 4 4 4 4)
H	None of the above

b. b. Draw a circle around the name of each function that is recursive:
(ga, g1, g2, g3, g4). [2 points**]**

These are all recursive programs.

c. Draw a circle around the name of each function that, when run, generates an iterative process:

(ga, gb, gc, gd, ge). [2 points] **Just gd.**

d. What is the running time of each of the functions? Choose from this table (answers may be used more than once.) [5 pts]

ga	J
gb	J
gc	J
gd	J
ge	L

I	Θ (p)
J	Θ (k)
K	Θ (p^2)
L	Θ (k^2)
M	Θ (log(k))
N	Θ (log(p))
P	None of the above

Note that if you were looking for log(n), you missed part of the concept of choosing a parameter that characterizes the size or complexity of the input, and that is allowed to grow.

Question 2. [10 points] Word switch
Your task is to finish writing a short program to translate from “netlingo” to English. For example, you know the abbreviations LOL = laughing out loud; AFAIK = as far as I know; OTOH = on the other hand.

```
(define dictionary
  '((LOL laughing out loud)
    (AFAIK as far as I know)
    (OTOH on the other hand)))

(define sample1 '(The textbook is correct AFAIK))
(define sample2 '(OTOH I never even opened it LOL))

(define (translate sent dict)
      define (translate sent dict)
      (cond ((null? sent) '())
            ((assoc (car sent) dict) ;; complete the program below
              (append (cdr (assoc (car sent) dict))  ;;* see below
                (translate (cdr sent) dict))))

              (else (cons (car sent)(translate(cdr sent) dict)))))

;; like 1 point per underline + 3 points if assembled correctly
```

`Example: (translate sample2 dictionary)` produces

```
 (on the other hand i never even opened it laughing out loud)
```

3. [3 points] Consider `(define (f a b c)(if (b a 0)(c a) a))`

Explain in a single English sentence what (f **x** < -) computes, if **x** has a numeric value.

If the bindings for < and – are unchanged from the defaults, then (f x < -) computes the same as (if (< x 0)(- x) x) or absolute value of x. If x is unbound or not a number, it will produce an error.

4. [3 points] threelisp
Ordered trees seem to benefit from a representation in which three items are used: (label left right) as a typical representation in lisp. A programming language designer proposes a version of lisp in which there are no pairs, but triples. That is, instead of (cons a b) there is (treecons a b c), and instead of car, cdr, there are accessor functions for the part of a tree t, (label t)(left t)(right t). Write one or more complete English sentences supporting one of these positions. (Circle which position you are supporting.)

a. This is a great idea.
b. This is a terrible idea.
c. This is not even an idea.

(a): Trees would be more compactly represented, and some accommodation can be made to store lists (say, s-expressions constituting lisp programs!) by ignoring the third item in a triple, say (cons a b) = (treecons a b nil).
So you could do it. Historical note, a system called TREET was designed by E.C. Haines, using this idea. 1965.
(b): You can already represent triples by lists of three items, so no real advantage is obtained this way, at least abstractly. By not having pairs, representing lists of one or two items, or four or more items becomes somewhat irregular. You can't use triples to represent symbolic expressions representing lisp programs which are essentially lists, usually longer than 3.
(c): Same as (b).
Some people muddled the ideas of abstraction and representation here. The proposal here was a different REPRESENTATION. A good argument to not support this representation is that the abstraction of trees does not really need it, and in particular, other kinds of trees with more than 2 sub-trees, are kind of inconvenient!   Some people added irrelevant or outright wrong statements to an initially correct answer, and ended up with fewer points.

5. [8 points] (evaluate)
Here is a self-contained but not very complete evaluator program.

```
(define (evaluate exp binding)
  (cond ((number? exp) exp)
        ((symbol? exp) (cadr (assoc exp binding))) ;get val from
binding
        ((eqv? (car exp) '+) (+ (evaluate (cadr exp) binding)
                                (evaluate (caddr exp)binding)))
        ((eqv? (car exp) '*) (* (evaluate (cadr exp) binding)
                                (evaluate (caddr exp)binding)))
        (else exp)))
```

What results from executing each of the following pieces of code?

`(evaluate 4)`	**error**
`(evaluate 4 '())`	**4**
`(evaluate 'a '((a 3)))`	**3**

`(evaluate '(+ a (* b c))`
`          '((a 1)(b 2)(c 3)))`
	7

`(evaluate '(+ a (* b c))`
`          '((a 1)(b 2)))`
	error

`(evaluate '(foo 3 (+ a b))`
`          '((foo *)(a 1)(b 2)))`
	(foo 3 (+ a b))

6. [10 points]
For this question you will not write a program. Just draw some pictures in the space below.

a. The following elements are inserted into a binary search tree, in order. That is, element 5 is inserted first. Draw a sequence of 8 trees, each showing the resulting tree after the insertion of a new element.

5 7 9 6 10 2 3 1 *(Version B of the test had a different sequence)*
We don't draw the whole sequence, just the end product.

b. How deep is the tree? (A tree with one node is defined as depth 0).

3
c. What is the depth of the most balanced binary search tree it is possible to construct using these same elements? **3**

```
        5
     2     7
   1  3  6   9
              10
```

7. [6 points total]

In week 4 homework you were introduced to a representation for a hierarchical file system. We are going to use the same representation except instead of using (file g) for the file named g, we will use a list of length 3: (file g 100) which means that the file named g is of length 100.

The list `t1` below, is an example file directory.

```
(define t1 '(directory a
                   (file b 100)
                   (file c 200)
                   (directory c
                          (file d 10)
                          (file e 10))
                   (file f 300)))
```

a. What value does this program return, when called on this example, (YYYY t1)? [2 points]

(300 10 10 200 100)

Version B of the test added the numbers instead of consing them

b. Describe in terms of the directory structure, what r is when the line with comment `;;*` is executed, and what the value of the second argument to count2 is. [4 points]

r is a list or sub-list of files or subdirectories of a directory.
The value of the second argument is the accumulated list of file sizes *(or the sum of file sizes)* up to this time, not yet examining the sublist (cdr r) of the list of files or directories.

```
(define (YYYY r)              ; r is a directory
  ;; we define count-top, which itself has 2 subroutines
  (define (count-top fun)

    (define (count1 r sofar)
     ;; called only on (directory ...)  or (file ....)
      (if (eqv? (car r) 'directory)  (count2 (cddr r) sofar)
       ;; otherwise (car r) is 'file
        (cons (fun r) sofar)))  ;; fun is passed in to count-top

    (define(count2 r sofar)
     ;; called only on ((file ...)..) or ((directory) ...) or ()
      (if (null? r) sofar
        (count2 (cdr r) (count1 (car r) sofar)))) ;;*
    ;; this next line, inside count-top, starts the processing
    ;; of a directory
    (count1 r '()))
  ;; this next expression is inside YYYY
   (count-top caddr) )
```