Midterm Examination Solutions, Cmput 325

1. Understanding Lisp code.

1.1 Consider the following Lisp definition:

```
(defun b (L)
       (If (null L)
           T
           (not (b (cdr L)))
       )
)
```

Show the result of evaluating each expression below.

```
(b '(a))
```

answer: NIL

```
(b '(1 2 3 4))
```

answer: T

1.2 Consider

```
(defun f (L)
       (if (null L)
           nil
           (cons (cons (caar L) (cadar L)) (f (cdr L)))
       )
)
```

Note that

```
(caar ...)  is a shorthand for (car (car ...))

(cadar ...) for  (car (cdr (car  ...)))
```

What will be returned after executing the following expressions?

```
(f '((a b)))
```

answer: ((a . b))

```
(f '((1 a) (2 b) (3 c)))
```

```
answer: ((1 . a) (2 . b) (3 . c))
```

1.3 Consider

```
(defun g (X Y)
    (cond ((null Y) nil)
          ((null (cdr Y)) nil)
          ((eq X (car Y)) (cons (cadr Y) (g X (cdr Y))))
          (t (g X (cdr Y)))
    )
)
```

What will be returned after executing the following expressions?

```
(g 'd '(a d f d g))
```

```
answer: (f g)
```

```
(g 'd '(q w e r))
```

```
answer: NIL
```

1.4 What will be returned when the following expression is evaluated

```
(mapcar
       '(lambda (x) (cons (cons (car x) nil) (cdr x)))
       '((a b) (c d))
)
```

```
answer: (((a) b) ((c) d))
```

2. Machine level representations.
2.1 (2 marks) Two of the following s-expressions are equivalent. Which two?

```
1.   (a  b . (c . d))
```

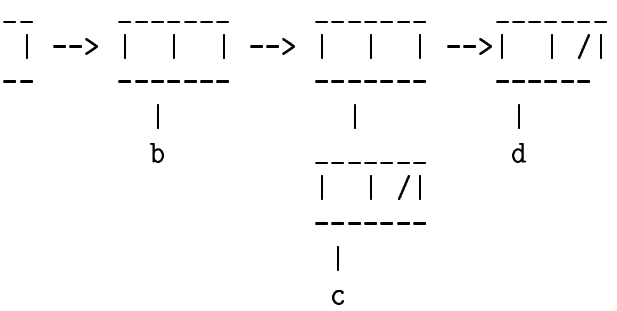
2. (a (b . c) . d)

3. (a . ((b . c) . d))

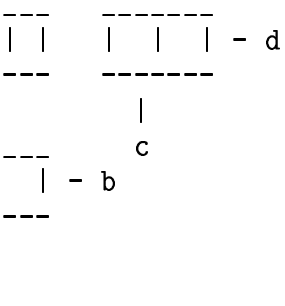
answer: 2 and 3

2.2 Show the machine level representations of the following s-expressions.

(a b (c) d)



((a . b) c . d)



3. In this problem you may use any builtin functions that have been allowed in the first two assignments.

3.1 (4 marks) For each s-expression below, write the Lisp code that constructs it.

((a b) c)

answer: (cons (cons 'a (cons 'b nil)) (cons 'c nil)

```
 (a (b . c))

answer: (cons 'a (cons (cons 'b 'c) nil))
```

3.2 (4 marks) For each s-expression below, write the Lisp code that returns the element a in it.

```
 ((b d) (a) . e)
answer:  (caadr  '((b d) (a) . e))

 (c (b (a)))
answer: (caadadr  '(c (b (a))))
```

3.3 (7 marks) Define the following function

```
(defun last (L) ....)
```

which takes a non-empty list L and returns the last element in L. For example,

```
(last '(a b (c) (d)))  ==> (d)
```

```
(defun last (L)
    (if (null (cdr L))
        (car L)
        (last (cdr L))
    )
)
```

3.4 (10 marks) Define a Lisp function

```
(defun drop (L)  ... )
```

which takes a nonempty list L of atoms and drops the nth atoms in L where n is an even number. For example

```
(drop '(a))  ==> (a)
(drop '(a b c d e))  ==> (a c e)
```

There are a number of possible ways to do this.

```
(defun drop (L)
   (if (null L)
       L
       (cons (car L) (evenDrop (cdr L)))
   )
)


(defun evenDrop (L)
   (if (null L)
       L
       (drop (cdr L))
   )
)
```

Here are some typical mistakes:

1.
```
(defun drop (L)
 (if (null L)
     L
     (cons (car L) (drop (cddr L)))
 )
)
```

It's wrong for the case

```
(drop '(a))
```

since cddr will attempt to get the cdr part of NIL (recall this is an atom). However, gcl will ignore this error and simply return NIL in this case.

2.
```
(defun drop (L)
    (if (null (cdr L))
        L
        (cons (car L) (drop (cddr L)))
    )
)
```

It's wrong for the case

```
(drop '(a b))
```

since after

```
(drop (cddr '(a b)))
```

will get (drop NIL), the case which is not defined.
Once again, gcl does return NIL for (drop NIL).

In both cases, no marks were deducted, since there was no way to separate those who use this trick intensionally (though it's not really a good programming practice from those who don't know what they were doing.

4. (12 marks) In this problem, we use the notation $\{x_1 \to v_1, ..., x_m \to v_m\}$ for context, and $[Fn, CT]$ for closure where Fn is a lambda function and CT is a context. We assume that the initial context is CT0.

For your convenience, we identify a function application by underlying its function part and argument part.

4.1 (6 marks) Consider the following function application.

```
((lambda (x) (+ x 1))  ((lambda (y) (+ y 1)) 2))
 ^^^^^^^^^^^^^^^^^^^^  ^^^^^^^^^^^^^^^^^^^^^^^^
     function                argument
```

(a) Show the result of evaluating this expression.

4

(b) Show the context when the subexpression (+ x 1) is being evaluated.

{ x->3 } U CT0

4.2 (6 marks) Consider evaluating the following lambda expression.

```
(((lambda (f) (lambda (x) (f x))) (lambda (y) y))   5)
 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^   ^
            function                                argument
```

The function part itself is an application, which is depicted further by

```
((lambda (f) (lambda (x) (f x)))   (lambda (y) y))
 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^   ^^^^^^^^^^^^^^
          function                 argument
```

(a) Show the result of evaluating this expression.

```
5
```

(b) Show the context when the subexpression (f x) is evaluated.

```
{x->5  f->[Fn, CT0]} U CT0   where Fn = (lambda (y) y)
```

5. (8 marks) Consider the following expression

```
(* (+ 3 5) 6)
```

(a) Show the compiled (SECD) code of this expression.
(LDC 6 LDC 5 LDC 3 + *)

(b) Show how the compiled code is executed on the SECD machine.

```
s e c d      where c contains the compiled code
```

```
Very similar to the examples shown in class.
```