COP 4020 Programming Languages 9-18-86

Sample Solutions to Assignment 1

Question #1: Two points for each correct answer. Partial credit, if quotes were missing, or too many parantheses, or mixup of CONS, LIST, etc. The following was expected:

```
-> (list 'a 'b 'c)
(a b c)
-> (append '(a) '(b) '(c))
(a b c)
-> (cons 'a (cons 'b (cons 'c '())))      ;; There are several ways to
(a b c)                                   ;; solve this, depending what
-> (cons 'a (cons 'b (cons 'c ())))       ;; you select for the empty
(a b c)                                   ;; list.
-> (cons 'a (cons 'b (cons 'c nil)))
(a b c)
```

Question #2: One point for each correct answer.

```
-> (length ())                            ;; LENGTH takes a list as
0                                         ;; argument and returns the
-> (length '(a b c))                      ;; number of elements on the
3                                         ;; highest level of this list.
-> (length '(a (b (c)) d))
3               - ------- -  3 elements
```

Question #3: No points on this one, just to show you that you can get system functions pretty-printed, and how a DO-loop works in LISP:

```
-> (pp length)
(def length
  (lambda ($1$)
    (cond ((and $1$ (not (dtpr $1$)))
                (error "length: non list argument: " $1$))
          (t (cond ((null $1$) 0)
                   (t (do ((11 (cdr $1$) (cdr 11))
                           (i 1 (|1+| i)))
                          ((null 11) i))))))))

;; Sometimes the pretty-printer doesn't print the structure properly, so
;; I rearranged it. The function has two main CONDitions, first it checks
;; if the argument $1$ has a value. If the value is nil, the function AND
;; returns immediately with value nil, the second condition is tested,
;; which is the catch-all case T, and we check (null $1$). Since the value
;; was nil, (null nil) returns T, so 0 is returned by length.
;; If there is a non-nil argument, that is not a list, we enforce an error
;; message (function error can be used in any user defined function).
;; If we have an argument that is a list that is not empty, we execute the
;; DO-loop. The loop has two local variables: 11 and i. The initial value
;; of 11 is (cdr $1$), since we already know that the list is not empty we
;; know that there must be at least one argument. The variable is updated
;; after each iteration with (cdr 11). The variable i is our counter.
;; Initial value is 1, since we start with the second element. i will be
;; incremented by 1 after each iteration.
;; Finally, the DO-loop terminates, if we cdr-ed through 11, and there is
;; no more element left. The value returned is the last value of i.
```

Definitions
call by text -- all applied occurences of the formal parameters are replaced by the text of the actual parameter with any embedded identifiers left to bound in the local environment.

call by name -- all applied occurences of the formal parameters are replaced by the expression for the actual parameter with any embedded identifiers bound as they were in at the point of call.

CALL BY WHAT or is it PASS BY WHAT

```
int i, m, A[4]; /* globals */

main()
{
   i = 2; m = 2; A[0] = 1; A[1] = 0; A[2] = 3; A[3] = 4;

   P();
}

void swap_set ( int x, int y )
{
   int t;

   m = 1; t = x; x = y; y = t;
}

void P ( void )
{
   int m = 3 /* call it P.m so it is not confused with the global m */

   /* before */
   swap_set ( i, A[i] )
   /* after */
}
```

And the results are:

	i	m	P.m	A[0]	A[1]	A[2]	A[3]
before	2	2	3	1	0	3	4
and after, if the parameters are passed ...							
by value	2	1	3	1	0	3	4
by reference	3	1	3	1	0	2	4
by name	3	1	3	1	0	3	2
by text	3	2	1	1	0	3	2