# KONG Exam 1 (Version 1)

DO NOT SHARE THIS DOCUMENT OUTSIDE OF THIS DISCORD SERVER.

LET ME KNOW IF YOU WANT TO ADD SOMEONE YOU KNOW.

1. In a certain language expressions are written in infix notation. Class 1 operators have **highest** precedence and class 3 operators have **lowest** precedence. The languages has binary operators and a unary prefix operator, whose precedence classes are as follows:

| | **binary operators** | **unary prefix operator** | **associativity** |
|---|---|---|---|
| **Class 1:** | # | @ | left |
| **Class 2:** | % | [none] | left |
| **Class 3:** | $ | [none] | right |

(i)**[1.5 pts.]** Here are three expressions in this language; in each case, circle the operator that is applied last:

(a1): v # u $ p
(a2): y % @ z % x
(a3): w ^ v # u $ @ (y % @ z % x)

(ii)**[2 pts.]** Draw an abstract syntax tree of **each** of the aboe expressions (a1), (a2), and (a3).

Tree of **(a1)** Tree of **(a2)** Tree of **(a3)**

(iii) Rewrite the first expression of part (i) (i.e, the expression of (a1)) in prefix notation.

**ANSWER:** _____ **[0.5 pt.]**

(iv) Rewrite the first expression of part (i) (i.e, the expression of (a2)) in prefix notation.

**ANSWER:** _____ **[0.5 pt.]**

(v) Rewrite the first expression of part (i) (i.e, the expression of (a3)) in prefix notation.

**ANSWER:** _____ **[0.5 pt.]**

2.**[4 pts.]** Suppose the Lisp variable E has been given a value as follows:

**(setf E '((-1, -2) ((90 91) 92 93 94 95 96 97 98) (9 19 29 39 49 59 69)))**

(A) What is the value of **(first (first E))**?_____

(B) What is the value of **(second (first E))**?_____

(C) What is the value of **(first (second E))**?_____

(D) What is the value of **(rest (third E))**?_____

(E) Write a Lisp expression that does not involve any numbers, but which evaluates to the following list: **(-2 -1 ((90 91) (19 29 39 49 59 69))).**

3.**[2.5 pts.]** Suppose the expressions (A)-(E) below are evaluated by Lisp immediately after evaluation of the following expressions: **(SETF X 1) (SETF Y 2) (SETF L '(1 3 5 7))**

(A) **(CONS (LIST X Y) L)** **ANSWER:**_____

(B) **(APPEND (LIST X Y) L)** **ANSWER:**_____

(C) **(LIST (LIST X Y) L)** **ANSWER:**_____

(D) **(MAPCAR #'(LAMBDA (W) (* W 7)) L)** **ANSWER:**_____

(E) **(REMOVE-IF-NOT #'(LAMBDA (W) (= W 7)) L)** **ANSWER:**_____

4.**[1 pt.]** What is the value of the following Lisp expression?

**(cons (+ 1 2 3 4) (append (list '(+ 5 6)) '(+ 7 8)))**

Circle the correct answer:

(a) **(+ 1 2 3 4 (+ 5 6) 15)** (b) **(10 (+ 5 6) + 7 8)** (c) **(10 (11 + 7 8))**

(d) **(10 + 5 6 (+ 7 8))** (e) **((+ 1 2 3 4) + 5 6 + 7 8)**

5.**[1 pt.]** Just one of the expressions (a-e) evaluates to **(1 2 3 4 5)**. Which one?

(a) **(cons '(1 2 3 4) '(5))** (b) **(list '(1 2 3 4) 5)** (c) **(list '(1 2 3 4) '(5))**

(d) **(append '(1 2 3 4) 5)** (e) **(append '(1 2 3 4) '(5))**

6.**[1 pt.]** Suppose you want to define a Lisp function SQR that returns a list of the perimeter and the area of a square, given the length of one side of the square. **(SQR 3)** should return **(12 9)**. Just one of the following is a correct way to define such function SQR.

(a) **(defun sqr (L) (append (* 4 L) (* L L)))**

(b) **(defun sqr (L) append ((* 4 L) (* L L)))**

(c) **(edfun sqr (L) append (* 4 L) (* L L))**

(d) **(defun sqr (L) (list (* 4 L) (* L L)))**

(e) **(defun sqr (L) list ((* 4 L) (* L L)))**

7.**[1 pt.]** Consider this Lisp expression: **(mapcar #'cons '(1 2 3) '((A B) (C D E F) (G)))**

What is the value of this expression? Circle the correct answer.

(a) **T** (b) **NIL** (c) **(1 2 3 (A B) (C D E F) (G)))**

(d) **((1 A B) (2 C D E F) (3 G))** (e) **((1 2 3 A B) (1 2 3 C D E F) (1 2 3 G))**

8.**[2.5 pts.]** Complete the following definition of a LISP function SCORE->GRD that takes a single argument, s, and returns the symbol A, B, C, or F according to the following scheme:

**A: s >= 90  B: 90 > s >= 80  C: 80 > s >= 70  F: s < 70**

If the argument s is not a real number then the function should return the symbol **BAD-ARG** (You may use REALP to see if argument is a real number or not). Examples:

**(SCORE->GRD 'D) => BAD-ARG  (SCORE-GRD 84.3) => B  (SCORE-GRD 68.1) => F**

**(defun score->grd(s)** ;;; Rest of the code is omitted. Try on your own.

9.**[3.0 pts.]** A Lisp function **POS** is such that if **L** is a list and **E** is an element of **L** then **(POS E L)** returns the position of the first occurrence of **E** in **L**, but if **L** is a list and **E** is not an element of **L** then **(POS E L)** returns **0**.

Examples:
**(POS 5 ()) => 0**
**(POS 6 '(2 4 6 5 1 5)) => 3**
**(POS 6 '(1 2 4 6 5 1 5)) => 4**
**(POS 7 '(7 2 4 6 5 1 5)) => 1**
**(POS 8 '(2 4 6 5 1 5)) => 0**
**(POS 7 '(1 2 3 7 7 7)) => 4**

(a) If **e** is **not** equal to **(car L)** and **(pos e (cdr L)) => 0**, what is the value of **(pos e L)**? _____

(b) If **e** is **not** equal to **(car L)** and **(pos e (cdr L)) => 8**, what is the value of **(pos e L)**? _____

(c) Complete the full definition of POS function:

**(defun pos (E L)** ;;; Rest of the code is omitted. Try on your own.

10.**[1.5 pts.]** Complete the following definition of tail-recursive Lisp function **TR-ADD** such that if **L** is any list of numbers then **(TR-ADD L 0)** returns the sum of those numbers.

Example:
**(tr-add '(2 5 2 3) 0)**
= **(tr-add '(5 2 3) 2)**
= **(tr-add '(2 3) 7)**
= **(tr-add '(3) 9)**
= **(tr-add '() 12)** = **12**

**(defun tr-add (L res)** ;;; Rest of the code is omitted. Try on your own.

11.**[2.5 pts.]** Complete the definition below of a Lisp function **REPEATED-ELTS** such that, if **L** is any list of numbers, then **(REPEATED-ELTS L)** returns a list that contains just one element from each "run" of **two or more equal** numbers in the list, and contains no other elements.
For Example:

**(REPEATED-ELTS '(1 2 3 3 9 2 2 2 1 1 3 9 3 2 2 9 8 3 3 3 3 9)) => (3 2 1 2 3)**

Here the argument of **REPEATED-ELTS** contains **5** "runs" of **two or more equal** numbers; those 5 runs have been underlined above. Thus **REPEATED-ELTS** returns a list that consists of one element from each of those 5 runs. Further examples:

Example A0: **(repeated-elts ()) => NIL**
Example A1: **(repeated-elts '(7)) => NIL**
Example A2: **(repeated-elts '(7 7)) => (7)**
Example B: **(repeated-elts '(4 9 9 8 7 7)) => (9 7)**
Example C: **(repeated-elts '(6 4 9 9 8 7 7)) => (9 7)**
Example D: **(repeated-elts '(6 6 4 9 9 8 7 7)) => (6 9 7)**
Example E: **(repeated-elts '(6 6 6 4 9 9 8 7 7)) => (6 9 7)**

```
(defun repeated-elts (L)
      (cond
            ((endp L) NIL)
            ((not (eql (car L) (cadr L))) ; e.g, L = (6 5 ...) or (7)    [see examples C and A1]
                  (repeated-elts _____________))
            ((not (eql (car L) (caddr L))) ; e.g, L = (6 6 4 ...) or (7 7) [see examples D & A2]
                  (_____________ _____________(_____________ (cddr L))))
            (t (repeated-elts _____________))))  ; e.g, L = (6 6 6 4 ....)       [see example E]
```