1. In a certain language expressions are written in infix notation. The language has binary, unary prefix, and unary postfix operators that belong to the following precedence classes:

	binary ops	prefix ops	postfix ops	associativity
1st Class:	# ~	~	[none]	right
2nd Class:	&	[none]	$	left
3rd Class:	% ^	@	[none]	right

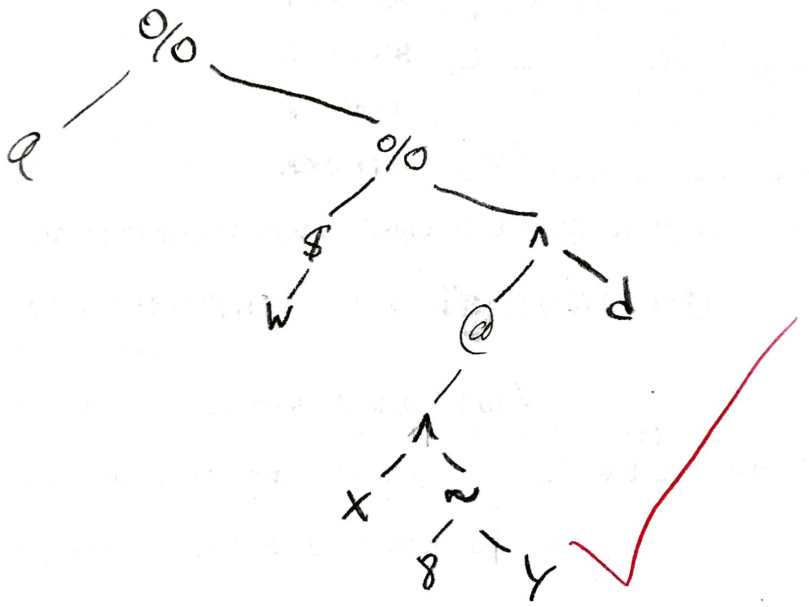
1st class operators have ***highest*** precedence and 3rd class operators have ***lowest*** precedence.

(a)**[0.5 pt.]** Circle the operator that is applied *last* when evaluating the following expression. [To help you, a subscript has been attached to each operator to indicate its precedence class and whether that precedence class is left-associative (L) or right-associative (R), but *this information can also be obtained from the above table*—e.g., @ is in the 3rd class and is right-associative, so it has a subscript of 3R below.]

$$(@_{3R})\ x\ ^{\wedge}_{3R}\ 8\ \sim_{1R}\ y$$

(b)**[0.5 pt.]** Circle the operator that is applied *last* when evaluating the following expression: $a\ (\%_{3R})\ w\ \$_{2L}\ \%_{3R}\ z\ ^{\wedge}_{3R}\ d$

(c)**[2 pts.]** Draw the abstract syntax tree of the following expression: $a\ \%_{3R}\ w\ \$_{2L}\ \%_{3R}\ (@_{3R}\ x\ ^{\wedge}_{3R}\ 8\ \sim_{1R}\ y)\ ^{\wedge}_{3R}\ d$



(d)**[1 pt.]** Rewrite the expression of part (c) in ***prefix*** notation.

%a%$w^@^x~8yd

(e)**[1 pt.]** Rewrite the expression of part (c) in ***postfix*** notation.

awSx8y~^@d^%%

2.**[1 pt.]** Suppose Rare Lisp is a language that is the same as Common Lisp except in that, whereas arguments of function calls in Common Lisp are evaluated in left-to-right order, arguments of function calls in Rare Lisp may be evaluated in any order. (In this respect Rare Lisp would be like Scheme and C++, whereas Common Lisp is like Java.) What happens when the expression

```
(+  (setf x 2)  x  (setf x 4))
```

is evaluated by a Rare Lisp interpreter? Circle the correct answer:

(a) If the arguments of + are evaluated in left-to-right order then the value 8 will be returned; if the arguments are evaluated in right-to-left order then the value 10 will be returned.

(b) If the arguments of + are evaluated in left-to-right order then the value 10 will be returned; if the arguments are evaluated in right-to-left order then the value 8 will be returned.

(c) An evaluation error will occur regardless of the order in which the arguments of + are evaluated.

(d) The value 8 will be returned regardless of the order in which the arguments of + are evaluated.

(e) The value 10 will be returned regardless of the order in which the arguments of + are evaluated.

3. Suppose the expressions (a) – (e) below are evaluated by Lisp immediately *after* evaluation of the following three SETF expressions: (SETF X 1) (SETF Y 2) (SETF L '(1 3 5 7))
Write down the value of each of (a) – (e). [Be careful to write parentheses where they should be and nowhere else! You will receive ***no credit*** if the right answer is (Z) and you write Z, or if the right answer is Z and you write (Z).]

-½

(a) `(CONS (LIST X Y) L)` **ANSWER:** ((1 2) (1 3 5 7)) **[0.5 pt]**

(b) `(APPEND (LIST X Y) L)` **ANSWER:** (1 2 1 3 5 7) **[0.5 pt]**

(c) `(LIST (LIST X Y) L)` **ANSWER:** ((1 2) (1 3 5 7)) **[0.5 pt]**

(d) `(MAPCAR #'(LAMBDA (W) (* W 7)) L)` **ANSWER:** (7 21 35 49) **[0.5 pt]**

(e) `(MAPCAR #'(LAMBDA (I) (EQUAL I 5)) L)` **ANSWER:** (NIL NIL T NIL) **[0.5 pt]**

4.**[1 pt.]** What is the value of the Lisp expression (mapcar #'cons '(1 2 3) '((A B) (C D E F) (G))) ?
Circle the 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))

5.**[4 pts.]** Suppose the Lisp variable E has been given a value as follows:

```
(setf E '(((2 Q) 9 19 29 39 49 59 69) (90 91 92 93 94 95 96 97)) )
```

Write a Lisp expression ***which does not involve any numbers***, but which evaluates to the list

```
((2 9 19 29 39 49 59 69) (92 93 94 95 96 97))
```

Your expression may contain the variable E and any Lisp function calls.

-1½

(list (cons (caaar E) (cadr (car E))) (caddr E))

(second (first E)) is NOT what you want. (rest (first E)) is correct.

(caddr E) = (fourth E) NOT what you want!

23

-5

6.**[2 pts.]** Complete the following definition of a Lisp function SAFE-AVG that takes 2 arguments and returns the *average* (i.e., the mean) of those 2 arguments if they are numbers. But if one or both of the arguments is not a number, then the function returns the symbol BAD-ARGS. Examples:

(SAFE-AVG 2.0 6.4) => 4.2 (SAFE-AVG 3 7) => 5

(SAFE-AVG '(23.1) 47.3) => BAD-ARGS (SAFE-AVG 'ONE 'TWO) => BAD-ARGS

Hint: You may want to use the built-in predicate function NUMBERP.

(defun safe-avg (m n)

```
(cond((and (numberp m) (numberp n))
(/ (+ m n) 2))
(t 'BAD-ARGS)))
```

-1/4

7.**[3 pts.]** Complete the following definition of a recursive Lisp function SPLIT-NUMS such that if N *is a non-negative integer* then (SPLIT-NUMS N) returns a list of two lists: The first of the two lists consists of the ***odd*** integers between 0 and N in descending order, and the second list consists of the ***even*** integers between 0 and N in descending order. Examples:

(SPLIT-NUMS 0) => (NIL (0))
(SPLIT-NUMS 7) => ((7 5 3 1) (6 4 2 0))
(SPLIT-NUMS 8) => ((7 5 3 1) (8 6 4 2 0))
(SPLIT-NUMS 9) => ((9 7 5 3 1) (8 6 4 2 0))

(5 3 1) (6 4 2 0)
6 => ((6 4 2 0) (5 3 1))
7 => ((7 5 3 1) (6 4 2 0))

```
(defun split-nums (n)
  (if (zerop n)
      '(nil (0))
      (let ((X (split-nums (- n 1))))
        (if (evenp n)
            (list (car x)  (cons n (cadr x)))
            (list (cons n (car x))  (cadr x))))))
```

8.**[2 pts.]** Define a Common Lisp function `least` such that if *k* is a real number and *x* is a list of real numbers then `(least k x)` returns *k* if every element of *x* if is ≥ *k*, but returns the least element of *x* if some element of *x* is < *k*; in other words, `(least k x)` returns the least element of (cons *k x*).

Hint: A Scheme analog of this function is defined on p. 23 / p. 418 of the course reader / Sethi, in Exercise 10.2. But you must write a Common Lisp function. For example, `else` has no predefined meaning in Common Lisp, and so the COND clause `(else ...)` in the definition given in Exercise 10.2 would be incorrect.

```
(defun least (k x)
  (cond((and (numberp k) (endp x))
        ((< (car x) k) (least (car x) (cdr x)))
        (t (least k (cdr x))))))
```

k

-1/2

Base case is wrong!

23 3/4

-3/4

9. **[2.5 pts.]** Complete the following definition of a Common Lisp function `REMOVE-ADJ-DUPL` such that if *l* is a list of atoms then `(REMOVE-ADJ-DUPL l)` is a list obtained from *l* by removing all but one member of each sequence of adjacent duplicates in *l*.

Examples:
A. `(REMOVE-ADJ-DUPL NIL) => NIL`
B. `(REMOVE-ADJ-DUPL '(K)) => (K)` => (A D C A B)
C. `(REMOVE-ADJ-DUPL '(P A A D D D D C C A B B)) => (P A D C A B)`
D. `(REMOVE-ADJ-DUPL '(P P A A D D D D C C A B B)) => (P A D C A B)`
E. `(REMOVE-ADJ-DUPL '(Q P A A D D D D C C A B B)) => (Q P A D C A B)`

```
(defun remove-adj-dupl (L)
   (if (endp (cdr L))
       L ; Hint: see Examples A and B above
       (let ((X (remove-adj-dupl (cdr L))))
          (if (equal (car L) (cadr L))
              X ; Hint: see Example D
              (cons (car L) X) )))) ; Hint: see Example E
```

(L) calls the function L. Here L is not a function, so (L) is illegal! -1/2

10. **[1 pt.]** Write a definition of the function `REMOVE-ADJ-DUPL` of the previous question ***without*** using LET or LET*.

```
(defun remove-adj-dupl (L)
   (if (endp (cdr L))
       L
       (if (equal (car L) (cadr L))
           (remove-adj-dupl (cdr L))
           (cons (car L) (remove-adj-dupl (cdr L))))))
```

11. Suppose the expressions (a) and (b) below are evaluated by Lisp immediately ***after*** evaluation of the following SETF expression: (SETF CAR #'CADR) Write down the values of (a) and (b).

(a) `(FUNCALL CAR '(A B C D))` **ANSWER:** ~~((B C D))~~ B **[0.5 pt]**

(b) `(FUNCALL #'CAR '(A B C D))` **ANSWER:** ~~((NIL) (NIL) (NIL) (NIL))~~ A **[0.5 pt]**

-1

-1/2