Question #1

COMPRESS

```
(defun compress (ls)
  (compress-helper ls nil nil))

(defun compress-helper (todo latest done)
(cond
  ((null todo) ;----------> nothing to do = just output done + latest
    (append done latest))
  ((null latest) ;----------> at the beginning or after change = remember the firs element
    (compress-helper (cdr todo) (list (car todo) 1) done))
  ((equal (car todo) (car latest)) ;----------> the the next element is the same as the latest = add to count
    (compress-helper (cdr todo) (list (car todo) (1+ (cadr latest))) done))
  (t ;----------> otherwise = next element is different so add to result and reset latest
    (compress-helper (cdr todo) (list (car todo) 1) (append done latest)))
))
```

COMPRESS2

```
(defun compress2 (ls)
  (compress2-helper2 ls nil nil))

(defun compress2-helper2 (todo latest done)
(cond
  ((null todo) ;------> nothing to do = just output done + latest
    (append done (compress2-helper3 latest)))
  ((null latest) ; --------> at the beginning or after change = remember the firs element
    (compress2-helper2 (cdr todo) (list (car todo) 1) done))
  ((equal (car todo) (car latest)) ;----------> the the next element is the same as the latest = add to count
    (compress2-helper2 (cdr todo) (list (car todo) (1+ (cadr latest))) done))
  (t ;----------> otherwise = next element is different so add to result and reset latest
    (compress2-helper2 (cdr todo) (list (car todo) 1) (append done (compress2-helper3 latest))))
))

(defun compress2-helper3 (ls)
(cond
  ((or (null ls) (atom (car ls))) ls)
  (t (list (compress2 (car ls)) (cadr ls)))
))
```

DECOMPRESS

```
(defun decompress (ls)
(if (null ls)
  nil
  (append (decompress-expand (car ls) (cadr ls)) (decompress (cddr ls)))
))

(defun decompress-expand (elem n)
(if (zerop n)
  nil
  (cons elem (decompress-expand elem (1- n)))
))
```

DECOMPRESS2

```
(defun decompress2 (ls)
(if (null ls)
  nil
  (append (decompress-expand (decompress2-helper (car ls)) (cadr ls)) (decompress2 (cddr ls)))
))

(defun decompress2-helper (ls)
(if (atom ls)
  ls
  (decompress2 ls)
))
```

Question #2

```
(defun listpp (ls)
(cond
  ((null ls) t)
  ((atom ls) nil)
  ((and (or (atom (car ls)) (listpp (car ls))) (listpp (cdr ls))) ls)
))
```

Question #3

(machine-version)

➔ PC/686

(list (setq answer ‘left-last) (setq answer ‘middle-last) (setq answer ‘right-last))

➔ (LEFT-LAST MIDDLE-LAST RIGHT-LAST)

answer

➔ RIGHT-LAST

Therefore it is left to right.

Question #4

```
(defun inner-product2 (ls inner-func outer-func)
  (reduce outer-func (apply 'mapcar inner-func ls)))
```

For 1 level:

```
(inner-product2 '((1 2 3)(4 5 6)) '* '+)
```
➔ 32
```
(inner-product2 '((1 2 3)(4 5 6)(7 8 9)) '* '+)
```
➔ 270

For up to 2 levels:

```
(defun *2 (e1 e2)
(if (atom e1)
  (* e1 e2)
  (inner-product2 (list e1 e2) '* '+)
))

 (inner-product2 ' (((1 2 3) 2 3) ((6 7 8) 7 8)) '*2 '+)
```

For any number of levels:

```
(defun *n (e1 e2)
(if (atom e1)
  (* e1 e2)
  (inner-product2 (list e1 e2) '*n '+)
))

(inner-product2  '(((1 2 3) 2 3) ((6 7 8) 7 8)) '*n '+)
```
➔ 82
```
(inner-product2  '(((1 2 3) (2 3 4) 3) ((6 7 8) (7 8 9) 8)) '*n '+)
```
➔ 142
```
(inner-product2  '(((1 (2 1 9) 3) (2 3 4) 3) ((6 (7 1 8) 8) (7 8 9) 8)) '*n '+)
```
➔ 215