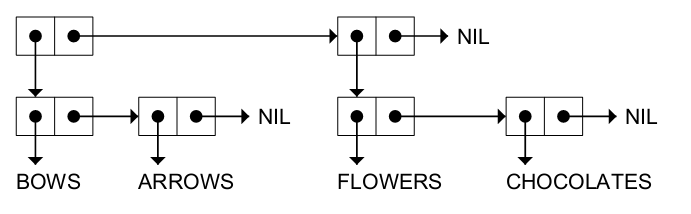
***1. Write the parenthesis notation for each cons cell structure***



((BOWS ARROWS)(FLOWERS CHOCOLATES))

****

(A (B C) D)



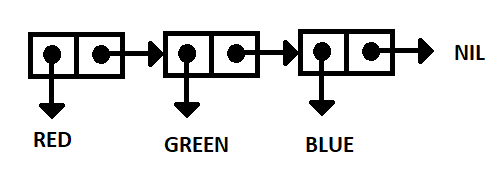
(A B (C D))



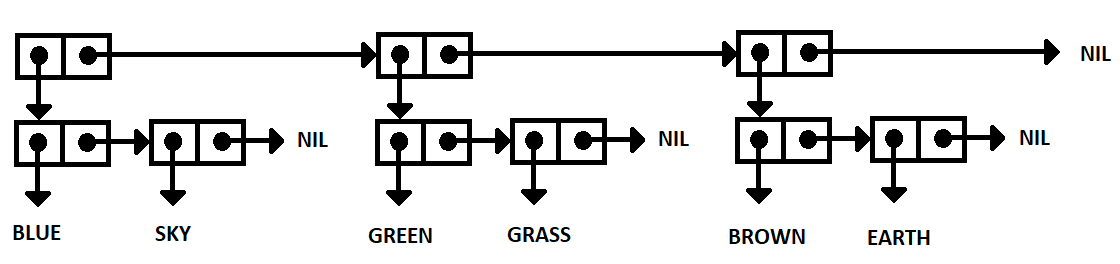
(((PHONE HOME)))

***2. Draw the internal structure for the following lists***

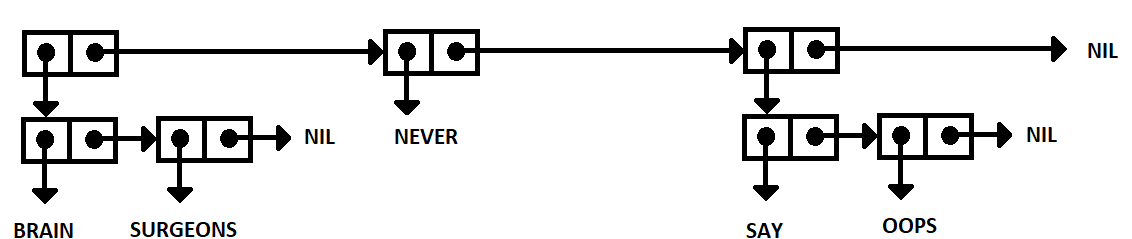
(RED GREEN BLUE)



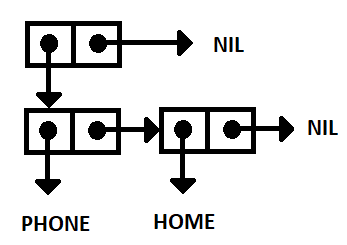
((BLUE SKY) (GREEN GRASS) (BROWN EARTH))



((BRAIN SURGEONS) NEVER (SAY OOPS))



((PHONE HOME))



***3. Consider the following list, and answer the following questions.***

**((BLUE CUBE) (RED PYRAMID))**

|  |  |
| --- | --- |
| CAR | (BLUE CUBE) |
| CDR | ((RED PYRAMID)) |
| CADR | (RED PYRAMID) |
| CDADR | (PYRAMID) |
| CADADR | PYRAMID |

***4. Number of elements***

|  |  |
| --- | --- |
| (OPEN THE POD BAY DOORS HAL) | 6 |
| ((OPEN) (THE POD BAY DOORS) HAL) | 3 |
| ((1 2 3) (4 5 6) (7 8 9) (10 11 12)) | 4 |
| ((ONE) FOR ALL (AND (TWO (FOR ME)))) | 4 |
| ((Q SPADES)(7 HEARTS)(6 CLUBS)(5 DIAMONDS)(2 DIAMONDS)) | 5 |

***5. Evaluate***

|  |  |  |
| --- | --- | --- |
| (list 'cons t nil) | | (CONS T NIL) |
| (eval (list 'cons t nil)) | | (T) |
| (eval (eval (list 'cons t nil))) | | T |
| (apply #'cons '(t nil)) | | (T) |
| (eval nil) | | NIL |
| (list 'eval nil) | | (EVAL NIL) |
| (eval (list 'eval nil)) | | NIL |
| (setf line '(roses are red)) | (ROSES ARE RED) | | |
| (reverse line) | (RED ARE ROSES) | | |
| (first (last line)) | RED | | |
| (nth 1 line) | ARE | | |
| (reverse (reverse line)) | (ROSES ARE RED) | | |
| (append line (list (first line))) | (ROSES ARE RED ROSES) | | |
| (append (last line) line) | (RED ROSES ARE RED) | | |
| (list (first line) (last line)) | (ROSES (RED)) | | |
| (cons (last line) line) | ((RED) ROSES ARE RED) | | |
| (remove 'are line) | (ROSES RED) | | |
| (append line '(violets are blue)) | (ROSES ARE RED VIOLETS ARE BLUE) | | |
| (defun add1 (n) (+ 1 n))  (mapcar #'add1 '(1 3 5 7 9)) | (2 4 6 8 10) | | |
| (mapcar #'zerop '(2 0 3 4 0 -5 -6)) | (NIL T NIL NIL T NIL NIL) | | |
| (mapcar #'null '(0 1 T NIL () T)) | (NIL NIL NIL T T NIL) | | |

***7. More evaluations***

|  |  |
| --- | --- |
| (setf x1 (list 'a 'b 'c))  (setf x2 (list 'a 'b 'c))  (setf z x1) | (A B C)  (A B C)  (A B C) |
| (equal x1 x2) | T |
| (eq x1 x2) | NIL |
| (eq z x1) | T |
| (eq z '(a b c)) | NIL |
| (equal z '(a b c)) | T |
| (eql 'foo 'foo) | T |
| (eql 3 3.0) | NIL |
| (= 3 3.0) | T |
| (car NIL) | NIL |
| (cdr NIL) | NIL |

***8. Differences between EQ, EQL, EQUAL, EQUALP, and = in Lisp***

|  |  |
| --- | --- |
| = | Compares numbers only and checks for equal value |
| eq | Check for same object and memory address (same object) |
| eql | Numbers of same type and value are eql and arguments that are eq are eql |
| equal | Numbers and characters are compared with eql. Arguments that are structurally the same are equal, as well, such as strings and lists. If the printed representations are the same, they are equal. Does not work for hashes or structures. |
| equalp | Similar to equal but case insensitive and type insensitive. Also, equalp can correctly compare identical arrays, hashes and structures. |