**Lab 7**

Problem 1

* Fetch

(defun **fetch** (key assoc\_list)

    (if (assoc key assoc\_list)

        (cadr (assoc key assoc\_list))

        '?

    )

)

* Test Cases:

(defvar bob '(

    (temperature 100)

    (pressure (120 60))

    (pulse 72)

))

(print

    (fetch 'temperature bob)

)

*; 100*

(print

    (fetch 'pressure bob)

)

*; (120 60)*

(print

    (fetch 'complaints bob)

)

*; ?*

Problem 2

* List Keys:

(defun **list\_keys** (assoc\_list)

    (mapcar

        (lambda (kv\_pair) (car kv\_pair))

        assoc\_list

    )

)

* Test Cases:

(defvar bob '(

    (temperature 100)

    (pressure (120))

    (pulse 72)

))

(print

    (list\_keys bob)

)

*; (TEMPERATURE PRESSURE PULSE)*

(defvar dan nil)

(print

    (list\_keys dan)

)

*; NIL*

[for the next 3 problems]

Helper Function: Make Person

(defun **make\_person** (name father mother)

    (let (

        (person (gensym "PERSON"))

    )

        (setf (get person 'father) father)

        (setf (get person 'mother) mother)

        (setf (get person 'name) name)

        person

    )

)

Variables

*; GGG Grandparent*

(defvar jon (make\_person "Jon" nil nil))

*; Great Great Grandparents*

(defvar john (make\_person "John" jon nil))

(defvar emi  (make\_person "Emi"  nil nil))

*; Great Grandparents*

(defvar john2 (make\_person "John II" john nil))

(defvar sara  (make\_person "Sara"    nil  nil))

(defvar rich  (make\_person "John II" nil  nil))

(defvar cara  (make\_person "Cara"    nil  emi))

*; Grand Parents*

(defvar jonh3 (make\_person "John III" john2 sara))

(defvar ali   (make\_person "Ali"      rich  cara))

*;Parents*

(defvar dad (make\_person "Dan" jonh3 ali))

*; Person*

(defvar person (make\_person "Gion" dad nil))

Problem 3

* Grandfather

(defun **grandfather** (person)

    (let\* (

        (father (get person 'father))

        (grandfather (get father 'father))

    )

        grandfather

    )

)

* Test Cases:

(print

    (get (grandfather person) 'name)

)

*; "John III"*

(print

    (get (grandfather rich) 'name)

)

*; NIL*

(print

    (get (grandfather john2) 'name)

)

*; "John"*

Problem 4

* Adam

(defun **adam** (person)

    (let (

        (father (get person 'father))

    )

        (if (null father)

            person

            (adam father)

        )

    )

)

* Test Cases:

(print

    (get (adam person) 'name)

)

*; "Jon"*

(print

    (get (adam dad) 'name)

)

*; "Jon"*

(print

    (get (adam rich) 'name)

)

*; "John II"*

(print

    (get (adam jon) 'name)

)

*; "Jon"*

Problem 5

* Ancestors:

(defun **ancestors** (person)

    (if (null person)

        nil

        (let (

            (father (get person 'father))

            (mother (get person 'mother))

        )

            (append (list person) (ancestors father) (ancestors mother))

        )

    )

)

* Print Names:

(defun **print\_names** (people)

    (cond

        ((not (null people))

            (print (get (car people) 'name))

            (print\_names (cdr people))

        )

    )

)

* Test Cases:

(print\_names

    (ancestors person)

)

*; "Gion"*

*; "Dan"*

*; "John III"*

*; "John II"*

*; "John"*

*; "Jon"*

*; "Sara"*

*; "Ali"*

*; "Rich"*

*; "Cara"*

*; "Emi"*

(print\_names

    (ancestors ali)

)

*; "Ali"*

*; "Rich"*

*; "Cara"*

*; "Emi"*

(print\_names

    (ancestors jon)

)

*; "Jon"*

Problem 6