Lisp Notes:

In Listener:

(let ((\*Standard-Input\* (open “filename” :direction :input)))

(family))

Cons- adds to list

Put in file to test files

Use read to get data from file, bunch of optional arguments

(read &optional (stream t) (eof-error-p nil) (eof-value nil) ( recursive? Nil))

In editor:

Say this: (read \*Standard-Input\* nil nil)

Lists are basic data structure in Lisp

Every expression returns a value. IF statements will return values.

Format: different output function. And formats them according to string derectives

For string pass nil

Ctrl + shift + a: see args list.

Any argument after &Rest or &Key is treated differently

Setf -> assignment function

(list ……) list in Lisp

For project:

(E name1 name2 name3)

Makes it easier to read expressions

Read function -> always reads from standard in

Lisp is case insensitive for symbol input

Look at first symbol at line

To get access to first character (first …) remember to make a var for it. Same with second, third fourth, fifth and so on… when no more it will return NIL

Defun -> create a function with this

;;;; -\*- Mode: Lisp; -\*- Top level comment needed for Lisp fucntionalities

Creates a record with one parameter ‘name’ parents and children

(defstruct person

(name nil)

(parents nil)

(children nil)

(Make-person … ) makes an instant of person struct

Keyword arguments: give name of the argument and then value

Finding stuff in a list:

Member function

(member 3 b ) “is the number 3 in list b”

Returns value and everything after,

Workaround setf found (member 2 b)

(if found

(first found)

Nil in lisp also represent false

:test #’equal -> does a pointer, value and number comparison instead of

(remove-duplicates list :test #’equal)

(dolist (i a)

(print

Esc-x then type Find Unbalanced Parentheses: jumps to spot where Lisp thinks there is a missing parentheses or one too many

(Format t “~a~%” Yes) – prints yes with newline

Starting CodeBase from Klassner:

(defstruct person

  (parents nil)) ;; add more slots as needed.

(defun storeperson (symbol-name struct family-tree)

  "Students need to write this! This should enter the person structure in STRUCT into the hashtable in FAMILY-TREE with the key in SYMBOL-NAME."

  family-tree)

(defun personstruct (symbol-name family-tree)

  "Returns a structure of type person corresponding to the key SYMBOL-NAME in the hashtable FAMILY-TREE. If there is no one in the tree with the name in SYMBOL-NAME, returns NIL."

  (gethash symbol-name family-tree nil))

(defun ancestors (p tree)

  "Returns a list of symbol-names of all the ancestors of P in TREE. Does not remove any duplicated names!"

   (let ((parent1 (personstruct (first (person-parents p)) tree))

         (parent2 (personstruct (second (person-parents p)) tree)))

     (when parent1

         (append (list (person-name parent1) (person-name parent2))

                 (ancestors parent1 tree)

                 (ancestors parent2 tree)))))

(defun family ()

  "This is the top-level function for the whole Lisp program."

  (let ((tree (make-hash-table :size 1000 :test #'equal)))

; body of program goes in here!

    ))