SRA

# The Little LISPer

by

Daniel P. Friedman

WILLIAM GEAR: *The Little LISPer* is a nonprogrammed text for the nonprogrammer or programmer who likes the feature that books used to have--one can curl up (near the refrigerator) and read it from front to back for a pleasurable introduction to LISP and its ideas. Evaluate it--you'll quote it.

MARK ELSON: Dan Friedman's LISP primer...provides by far the finest introduction to LISP that I have seen. Its excellence emanates along two different dimensions:

(1) The question-answer technique is beautifully exploited...Friedman blends development and redundancy, which the student needs for recall and logical guesswork, very effectively. Delicious juicy carrots seem to dangle at every step.

(2) The book has style, but that of course is not enough. The order of development of topics is what allows the style to be effective. Large languages can be tutorially developed from many different directions; LISP evolves so much power from so few basic features that a very delicate chain of development is necessary to bring comprehension and appreciation of that power. *The LISPer* achieves this admirably.

In short, the book is excellent in both coverage and style, a self-sufficient introduction to LISP for any logical human being from 8 to 80.

HAROLD STONE: *The Little LISPer* by Daniel Friedman...is both informative and enjoyable. It is an unusual book that definitely has a place in computer science education and should...find wide acceptance as a supplement to a programming languages course or a course on LISP. It does not serve as a reference manual...On the other hand, it has characteristics that no other text on LISP has. It explains the essential features of the language in a manner that is easily digested, and it is done at just the right level for reaching students.

## INDEX

# The Little LISPer

by

Daniel P. Friedman

Indiana University
Bloomington, Indiana

with

Phillip S. Blackerby
Robert J. King
Abraham Goldberg
David L. Walrath

## DEDICATION

To Patrick Suppes for the hours of pleasure his book, Introduction to Logic, has made possible.

## ACKNOWLEDGMENTS

Many thanks to John McCarthy, Mike Greenawalt, John Howard, Terry Pratt, David Musser, William Gear, Mark Elson, Harold Stone, Jonathan Slocum, Juny Armus, Bob Wesson, and Marylin Sealy for their extremely helpful comments about The Little LISPer. Grateful appreciation to Ben Shneiderman and Steve Mitchell for believing in this unorthodox approach to LISP. Thank you to Bill Cohagan who taught me LISP and to Ann and Frances Patterson for their careful typing and proofreading of this manuscript. I am greatly indebted to Patrick Mahaffey for his futuristic ideas of teaching LISP to public affairs graduate students and to John Gronouski, Dean of the Lyndon Baines Johnson School of Public Affairs, who was persuaded to try the experiment. I want to thank the four public affairs graduate students, Phillip S. Blackerby, Robert J. King, David L. Walrath, and Abraham Goldberg who not only were students in the "quickie" course but also persevered with me an additional week for the completion of the first draft. I especially want to thank Abraham Goldberg who contributed his editing talents and novice perspective for that week. Above all, I want to thank my wife Mary for her understanding, encouragement, and concern for my well-being during the completion of this manuscript.



Printed in the United States of America

Library of Congress Catalog Card Number: 73-91284

ISBN: 0-574-19165-8

TABLE OF CONTENTS

# PREFACE

The fundamental structure of the LISP programming language was derived from the abstract notions of lambda calculus and recursive function theory by John McCarthy. His goal was to produce a programming language with a powerful notation for defining and transforming functions. Instead of operating on numeric quantities, LISP was designed to manipulate abstract symbols, called atoms, and combinations of symbols, called lists. The expressive power of the language was recognized by a small number of researchers who were primarily concerned with difficult symbolic manipulation problems in artificial intelligence. The unorthodox nature of LISP contributed to the development of a narrow cult of LISP enthusiasts among the artificial intelligentsia. Although compilers for LISP were available for a wide variety of computers, early computer scientists were mainly concerned with the number-crunching aspect of machines. More recently symbolic manipulation problems have risen in importance and the increased complexity of the problem solving tasks assigned to the computer has prompted widespread interest in LISP.

Simultaneously, the increased number of students doing advanced work in computer science has focused academic interest on LISP. Courses on programming languages (such as I2 in the ACM Curriculum 68) have become the regular diet of undergraduate and graduate computer science majors. In these courses the goal was to teach the concepts of LISP in an organized and appealing manner. Unfortunately, the available texts required a great deal of tedious finger exercises before the meat of the course was accessible. Dan Friedman's <u>Little LISPer</u> provides a thoroughly palatable introduction to LISP with the emphasis on the elusive, but profound, concepts. The reader is continually challenged and is highly motivated to read on until the controlled confusion is resolved. By encouraging discovery, the author ensures that the student's interest is always at its peak. Finally, by providing instant feedback, the student's mastery of the material is guaranteed.

Ben Shneiderman
Department of Computer Science
State University of New York at Stony Brook

# INTRODUCTION

At an undergraduate curriculum design meeting two or three years ago, a professor of numerical analysis, Dr. Richard Bartels, expressed this opinion: "A student with an undergraduate degree in Computer Science who has not learned LISP is culturally deprived." Currently LISP is employed in a major proportion of all artificial intelligence research: computational linguistics, robotics, pattern recognition, generalized problem solving, theorem proving, game playing, algebraic manipulation, etc. While artificial intelligence comprises only one area of computer science, it is nevertheless a major subfield. Moreover, there is hardly any area in computer science which has been unaffected by LISP.

The Little LISPer is a programmed text based on lecture notes from a two-week "quickie" introduction to LISP for students with no previous programming experience and an admitted dislike for anything quantitative; many were preparing for careers in public affairs. The purpose of the course, and therefore of this book, is to teach the student how to think "recursively."

"Writing programs recursively in LISP is pattern recognition." Our attempt is to verify this statement within the covers of this book. Since our only concern is recursive programming, our treatment is limited to the why's and wherefore's of a few LISP functions, specifically, CAR, CDR, CONS, EQ, ATOM, NULL, NUMERP, ZEROP, ADD1, SUB1, AND, OR, NOT, and COND. The Little LISPer is not a complete book on LISP. In fact, the formal statement of the definition of these functions can be given on two pages. Hence our promise to you the reader is two pages worth of definitions and a book's worth of intuition and technique.

## GUIDELINES FOR THE READER

You should not rush through this book. Read carefully, because valuable hints are scattered throughout the text. Do not read the book in less than three sittings unless you are already familiar with LISP but are not a "LISPer." Read systematically. If you do not fully understand one chapter, you will understand the next one even less. The problems are arranged according to their difficulty; it will be difficult to solve later ones before you have solved those previous.

DO NOT HESITATE TO GUESS. This book is based on intuition, and yours is as good as anyone's. We expect you to guess. Occasionally we make you guess before giving the answer, either because we do not expect you to know an answer at a particular stage, or because we want you, in case you guessed wrongly, to slow down and look back for what you missed earlier.

No formal definitions are given in this book. We believe that you can form your own definitions and will thus remember them and understand them better than if we had written each one for you. But be sure you know and understand the Principles and Commandments thoroughly before passing them by. The key to learning LISP is "pattern recognition." The Commandments simply point out the patterns that you will have seen already. Early in the book, some concepts are narrowed for simplicity; later, they are expanded and qualified. You should also know that, while everything in the book is true LISP, LISP itself is more general and incorporates more than we could intelligibly cover in an introductory text. After you have mastered this book, you can read and understand the more advanced and complete books written on the subject.*

We do not tell you in this book how to get access to LISP on your local computer system or even suggest that you have a computer handy while you read it. If, however, you want to interact with a computer while you read, get in touch with the systems personnel at your local computation center. They will be happy to show you the procedures that are specific to their LISP. In spite of local variations, however, LISP is LISP and is basically the same throughout the world.

Food appears in many of our examples for two reasons. First, food is easier to visualize than abstract symbols. (This book is not good to read while dieting.) We hope the choice of food will help you understand the examples and concepts we use. Second, we want to provide you with a little distraction. We know how frustrating the subject matter can be, and a little distraction will help you keep your sanity.

DO NOT GIVE UP EASILY. LISP operates on the "lightning principle." The concepts of LISP will strike you suddenly when you are almost ready to give up. Once you have been struck, everything will fall into place. Be sure you understand chapters 2 and 8. Once you fully understand the concepts of "Do it, do it again, and again, and again..." and " 'Help' is on the way" or "Welcome to the hammock," you understand most of LISP.

So you are ready to start. Good luck! We hope your brain will enjoy the exercises waiting for it on the following pages.

Bon appétit!

---

*See references at the end of the book.

## CHAPTER 1: TOYS

| | | |
|---|---|---|
| 1.1) | Is it true that this is an atom?<br>ATOM | Yes,<br>because ATOM is a string of characters beginning with an A. |
| 1.2) | Is it true that this is an atom?<br>TURKEY | Yes,<br>because TURKEY is a string of characters beginning with a letter. |
| 1.3) | Is it true that this is an atom?<br>1492 | No,<br>since 1492 is a string of characters beginning with a digit.<br>Note: Numbers are not atoms yet. |
| 1.4) | Is it true that this is an atom?<br>3TURKEYS | No,<br>since 3TURKEYS is a string of characters beginning with a digit. |
| 1.5) | Is it true that this is an atom?<br>L | Yes,<br>because L is a string of --one-- characters beginning with a letter. |
| 1.6) | Is it true that this is a list?<br>(ATOM) | Yes,<br>because (ATOM) is an atom enclosed by parentheses. |
| 1.7) | Is it true that this is a list?<br>(ATOM TURKEY OR) | Yes,<br>because it is a collection of atoms enclosed by parentheses. |
| 1.8) | Is it true that this is a list?<br>(ATOM TURKEY) OR | No,<br>since this is only two S-expressions not enclosed by parentheses.<br>The first one is a list containing two atoms, and the second one is an atom. |
| 1.9) | Is it true that this is a list?<br>((ATOM TURKEY) OR) | Yes,<br>because the two S-expressions are now enclosed by parentheses. |
| 1.10) | Is it true that this is an S-expression?<br>XYZ | Yes,<br>because all atoms are S-expressions. |
| 1.11) | Is it true that this is an S-expression?<br>(X Y Z) | Yes,<br>because it is a list. |
| 1.12) | Is it true that this is an S-expression?<br>((X Y) Z) | Yes,<br>because all lists are S-expressions. |
| 1.13) | Is it true that this is a list?<br>(HOW ARE YOU DOING SO FAR) | Yes,<br>because it is a collection of S-expressions enclosed by parentheses. |
| 1.14) | How many S-expressions are in the list<br>(HOW ARE YOU DOING SO FAR)<br>and what are they? | 6<br>HOW, ARE, YOU, DOING, SO, and FAR. |
| 1.15) | Is it true that this is a list?<br>(((HOW) ARE)((YOU)(DOING SO)) FAR) | Yes,<br>because it is a collection of S-expressions, enclosed by parentheses. |
| 1.16) | How many S-expressions are in the list<br>(((HOW) ARE)((YOU)(DOING SO)) FAR)<br>and what are they? | 3<br>((HOW) ARE), ((YOU)(DOING SO)), and FAR. |
| 1.17) | Is it true that this is a list?<br>( ) | Yes,<br>because it is a collection of zero S-expressions enclosed by parentheses.<br>This special S-expression is called the null list. |

| | | |
|---|---|---|
| 1.18) | Is it true that this is a list?<br>( ( ) ( ) ( ) ( ) ) | Yes,<br>because it is a collection of S-expressions enclosed by parentheses. |
| 1.19) | What is the CAR of L, where L is the argument<br>(A B C) | A,<br>because A is the first atom of this non-null list. |
| 1.20) | What is the CAR of L, where L is the argument<br>((A B C) X Y Z) | (A B C)<br>because (A B C) is the first S-expression of this non-null list. |
| 1.21) | What is the CAR of L, where L is the argument<br>HOTDOG | No answer,<br>since you cannot ask for the CAR of an atom. |
| 1.22) | What is the CAR of L, where L is the argument<br>( ) | No answer,<br>since you cannot ask for the CAR of the null list. |

PRINCIPLE No. 1

> CAR is only defined for
> non-null lists.

| | | |
|---|---|---|
| 1.23) | What is the CAR of L, where L is the argument<br>(((HOTDOGS))(AND)(PICKLE) RELISH) | ((HOTDOGS)) --Read as: "The list of the list composed of the atom HOTDOGS."<br>because ((HOTDOGS)) is the first S-expression of this non-null list. |
| 1.24) | What is (CAR L), where L is the argument<br>(((HOTDOGS))(AND)(PICKLE) RELISH) | ((HOTDOGS)) |
| 1.25) | What is (CAR (CAR L)), where L is the argument<br>(((HOTDOGS))(AND)) | (HOTDOGS) |
| 1.26) | What is the CDR of L, where L is the argument<br>(A B C)<br>Note: "CDR" is pronounced "couder". | (B C),<br>because (B C) is the list L, without (CAR L). |
| 1.27) | What is the CDR of L, where L is the argument<br>((A B C) X Y Z) | (X Y Z). |
| 1.28) | What is (CDR L), where L is the argument<br>((X) T R) | (T R),<br>since (CDR L) is just another way to ask for "The CDR of the list L". |
| 1.29) | What is (CDR A), where A is the argument<br>HOTDOGS | No answer,<br>since you cannot ask for the CDR of an atom. |
| 1.30) | What is (CDR L), where L is the argument<br>( ) | No answer,<br>since you cannot ask for the CDR of the null list. |

PRINCIPLE No. 2

> CDR is only defined for non-null lists;
> the CDR of any list is always another
> list.

| | | |
|---|---|---|
| 1.31) | What is (CAR (CDR L)), where L is the argument<br>((B)(X Y)((C))) | (X Y),<br>because ((X Y)((C))) is (CDR L), and (X Y) is the CAR of (CDR L). |

| | | |
|---|---|---|
| 1.32) | What is (CDR (CDR L)), where L is the argument<br>((B)(X Y)((C))) | (((C))),<br>because ((X Y)((C))) is (CDR L), and (((C))) is the CDR of (CDR L). |
| 1.33) | What is (CDR (CAR L)), where L is the argument<br>(A (B (C)) D) | No answer,<br>since (CAR L) is an atom, and CDR will not take an atom for an argument; see PRINCIPLE No. 2. |
| 1.34) | What does (CAR L) take as an argument? | (CAR L) takes any non-null list as its argument, L. |
| 1.35) | What does (CDR L) take as an argument? | (CDR L) takes any non-null list as its argument, L. |
| 1.36) | What is the <u>CONS</u> of the atom A and the list L, where<br>A is the argument PEANUT, and<br>L is the argument (BUTTER AND JELLY)<br>This can alternatively be asked (CONS A L),<br>Read: "CONS the atom A onto the list L." | (PEANUT BUTTER AND JELLY)<br>because CONS sticks an atom onto the front of a list. |
| 1.37) | What is the CONS of S and L, where<br>S is (MAYONNAISE AND), and<br>L is (PEANUT BUTTER AND JELLY) | ((MAYONNAISE AND) PEANUT BUTTER AND JELLY),<br>because CONS sticks any S-expression onto the front of the list. |
| 1.38) | What is (CONS S L), where<br>S is ((HELP) THIS), and<br>L is (IS VERY ((HARD) TO LEARN)) | (((HELP) THIS) IS VERY ((HARD) TO LEARN)). |
| 1.39) | What does CONS take as its arguments? | (CONS S L) takes two arguments:<br>the first one, S, is any S-expression;<br>the second one, L, is any list. |
| 1.40) | What is (CONS S L), where<br>S is (A B (C)), and<br>L is ( ) | ((A B (C))),<br>since ( ) is a list. |
| 1.41) | What is (CONS S L), where<br>S is A, and<br>L is ( ) | (A) |
| 1.42) | What is (CONS S L), where<br>S is (A B (C)), and<br>L is B | †No answer,<br>since the second argument, L, <u>must</u> be a list. |
| 1.43) | What is (CONS S L), where<br>S is A, and<br>L is B | †No answer,<br>why? |

PRINCIPLE No. 3

<u>The second argument of (CONS S L) must be a list, and the result must also be a list.</u>

| | | |
|---|---|---|
| 1.44) | What is (CONS S (CAR L)), where<br>S is A, and<br>L is ((B) C C) | (A B),<br>why? |
| 1.45) | What is (CONS S (CDR L)), where<br>S is A, and<br>L is ((B) C D) | (A C D),<br>why? |

† In actual LISP systems (CONS S A) where A is any atom is well defined. Our intention is to prevent you from thinking about CONSing something onto an atom.

1.46) Is it true that the list, L, is the null list, where
L is ( )?
This question is alternatively read as:
(NULL L).

Yes,
because it is the list composed of zero S-expressions.

---

1.47) Is (NULL L) true, or false, where
L is the argument (A B C)

False,
because it is a non-null list.

---

1.48) Is (NULL L) true, or false, where
L is A

†No answer,
because you cannot ask (NULL L) of an atom.

---

PRINCIPLE No. 4

| (NULL L) is only defined for lists. |
|---|

1.49) Is it true, or false, that S is an atom, where
S is the argument HARRY

True,
because HARRY is a string of characters beginning with a letter.

---

1.50) Is (ATOM S) true, or false, where
S is HARRY

True,
since (ATOM S) is just another way to ask: "Is it true, or false, that S is an atom?"

---

1.51) Is (ATOM S) true, or false, where
S is (HARRY HAD A HEAP OF APPLES)

False,
since the argument, S, is a list.

---

†In actual LISP systems (NULL S) where S is any S-expression is well defined. The intention is to prevent you from thinking about (NULL A) where A is an atom.

---

1.52) How many arguments does ATOM take?
What are they?

ATOM takes one argument, which is any S-expression.

---

1.53) Is (ATOM (CAR L)) true, or false, where
L is (HARRY HAD A HEAP OF APPLES)

True,
because (CAR L) is HARRY, and HARRY is an atom.

---

1.54) Is (ATOM (CDR L)) true or false, where
L is (HARRY HAD A HEAP OF APPLES)

False,
see PRINCIPLE No. 2.

---

1.55) Is (ATOM (CAR (CDR L))) true, or false, where
L is (SWING LOW SWEET CHERRY)

True,
because (CDR L) is (LOW SWEET CHERRY), and
(CAR (CDR L)) is LOW, which is an atom.

---

1.56) Is (ATOM (CAR (CDR L))) true, or false, where
L is (SWING (LOW SWEET) CHERRY)

False,
because (CDR L) is ((LOW SWEET) CHERRY), and
(CAR (CDR L)) is (LOW SWEET), which is a list.

---

1.57) True or false: A1 and A2 are the same atom, where
A1 is HARRY, and
A2 is HARRY

True,
because both A1 and A2 are HARRY.

---

1.58) Is (EQ A1 A2) true, or false, where
A1 is the argument HARRY, and
A2 is the argument HARRY

True,
because (EQ A1 A2) is just another way to ask:
"Are A1 and A2 the same atom?"

---

1.59) Is (EQ A1 A2) true, or false, where
A1 is MARGARINE, and
A2 is BUTTER

False,
since the arguments A1 and A2 are different atoms.

---

1.60) How many arguments does EQ take, and what are they?

EQ takes two arguments, both of which must be atoms.

---

1.61) Is (EQ A L) true, or false, where
A is STRAWBERRY, and
L is (STRAWBERRY)

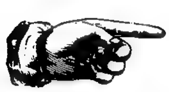
No answer,
since neither argument of EQ can be a list.

---

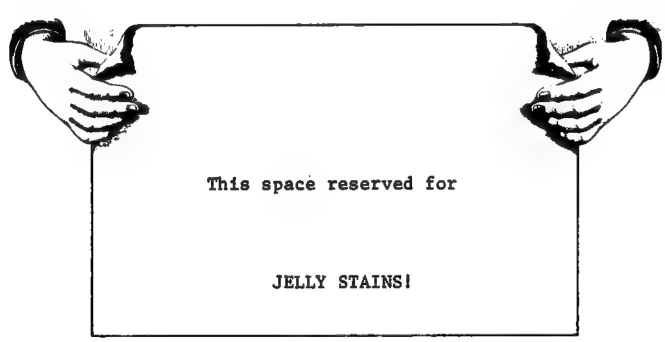
PRINCIPLE No. 5

> (EQ A1 A2) takes two arguments. Both of the arguments must be atoms, which begin with letters.

| | | |
|---|---|---|
| 1.62) | Is (EQ (CAR L) A) true, or false, where<br>L is (MARY HAD A LITTLE LAMB CHOP), and<br>A is MARY | True,<br>because (CAR L) is the atom MARY, and the argument A is also the atom MARY. |
| 1.63) | Is (EQ (CDR L) A) true, or false, where<br>L is (SOURED MILK), and<br>A is MILK | No answer,<br>See PRINCIPLES Nos. 2 and 5. |
| 1.64) | Is (EQ (CAR L)(CAR (CDR L))) true, or false, where<br>L is (BEANS BEANS ARE GOOD FOR YOUR HEART) | True.<br>This compares the first and second atoms in the list. |



Now go make yourself a PEANUT BUTTER AND JELLY SANDWICH.



2.1) True, or false: (ISLAT L), where L is (JACK SPRAT COULD EAT NO CHICKEN FAT)

True, because each S-expression in L is an atom.

---

2.2) True, or false: (ISLAT L), where L is ((JACK) SPRAT COULD EAT NO CHICKEN FAT)

False, since (CAR L) is a list.

---

2.3) True, or false: (ISLAT L), where L is (JACK (SPRAT COULD) EAT NO CHICKEN FAT)

False, since one of the S-expressions in L is a list.

---

2.4) True, or false: (ISLAT L), where L is ( )

True, because ( ) contains no lists, and because it does not contain any lists, it must be a LAT.

---

2.5) True, or false: a LAT is a list of atoms.

True. Note: Every LAT is a list, but not every list is a LAT.

---

2.6) Write the function (ISLAT L), using some, but not necessarily all, of the following functions: CAR, CDR, CONS, NULL, ATOM, and EQ.

We did not expect you to know this, but we wanted you to realize that you are still missing some necessary relationships. Go on to the next question. Good luck.

---

2.7) This is the function (ISLAT L), defined with the help of the functions, NULL, CAR, CDR, and ATOM:

```
(ISLAT (LAMBDA (L)
    (COND
    ((NULL L) T)
    ((ATOM (CAR L))(ISLAT (CDR L)))
    (T F)
                  ) ))
```

What is the value of (ISLAT L), where L is the argument (BACON AND EGGS)?

True. The function (ISLAT L), where L is (BACON AND EGGS) has the value T --true-- because L is a LAT.

---

2.8) How do you arrive at the answer T for the function (ISLAT L)?

We don't expect you to know this one, either. The answer is arrived at through asking the questions of (ISLAT L). HINT: Write down the function (ISLAT L) and refer to it for the next group of questions.

---

2.9) What is the first question asked by (ISLAT L)?

(NULL L)
Note: "(COND" is only a necessary stem that you will have to learn to live with. Similarly, "(LAMBDA" is a necessary part of the scenery, but for our purposes, it is as useful as a screen door on a submarine sandwich.

---

2.10) What is the meaning of the line ((NULL L) T), where L is (BACON AND EGGS)

(NULL L) asks if the argument L is the null list. If it is, then the value of the function is T. If it is not, then we ask the next question. In this case, L is not the null list, so we ask the next question.

---

2.11) What is the next question?

(ATOM (CAR L))

---

2.12) What is the meaning of the line ((ATOM (CAR L))(ISLAT (CDR L))), where L is (BACON AND EGGS)

(ATOM (CAR L)) asks if the first S-expression of the list, L, is an atom. If (CAR L) is an atom, then we want to know if the rest of L is also composed only of atoms. If (CAR L) is not an atom, then we ask the next question. In this case, (CAR L) is an atom, so the value of the function is (ISLAT (CDR L)).

---

2.13) What is the meaning of (ISLAT (CDR L))?

(ISLAT (CDR L)) finds out if the rest of the list, L, is composed only of atoms, by referring us back to the original function, but now with a new argument.

---

| | | |
|---|---|---|
| 2.14) | Now, what is the argument, L, for ISLAT? | Now the argument, L, is (CDR L), which is (AND EGGS). |
| 2.15) | What is the next question? | (NULL L). |
| 2.16) | What is the meaning of the line<br>((NULL L) T)<br>where<br>L is now (AND EGGS) | (NULL L) asks if the argument, L, is the null list. If it is, then the value of the function is T. If it is not, then we ask the next question. In this case, L is not the null list, so we ask the next question. |
| 2.17) | What is the next question that must be asked? | (ATOM (CAR L)) |
| 2.18) | What is the meaning of the line<br>((ATOM (CAR L))(ISLAT (CDR L)))<br>where<br>L is (AND EGGS)? | (ATOM (CAR L)) asks if (CAR L) is an atom. If it is an atom, then the value of the function is (ISLAT (CDR L)). If it is not an atom, then we ask the next question. In this case, (CAR L) is an atom, so we want to find out if the rest of the list, L, is composed only of atoms. |
| 2.19) | What is the meaning of<br>(ISLAT (CDR L)) | (ISLAT (CDR L)) finds out if the rest of L is composed only of atoms, by referring us back again to the original function, (ISLAT L), but this time, with the argument (CDR L), which is (EGGS). |
| 2.20) | What is the next question? | (NULL L) |
| 2.21) | What is the meaning of the line<br>((NULL L) T)<br>where<br>L is now (EGGS) | (NULL L) asks if the argument, L, is the null list. If it is, the value of the function is T -- true. If it is not, then move to the next question. In this case, L is not null, so we ask the next question. |
| 2.22) | What is the next question that must be asked? | (ATOM (CAR L)) |
| 2.23) | What is the meaning of the line<br>((ATOM (CAR L))(ISLAT (CDR L)))<br>where<br>L is now (EGGS) | (ATOM (CAR L)) asks if (CAR L) is an atom. If it is, then the value of the function is (ISLAT (CDR L)). If (CAR L) is not an atom, then ask the next question. In this case, (CAR L) is an atom, so once again we look at (ISLAT (CDR L)). |
| 2.24) | What is the meaning of (ISLAT (CDR L))? | (ISLAT (CDR L)) finds out if the rest of the list, L, is composed only of atoms, by referring us back to the original function, (ISLAT L), with L replaced by (CDR L). |
| 2.25) | Now, what is the argument for ISLAT? | ( ) |
| 2.26) | What is the meaning of the line<br>((NULL L) T)<br>where<br>L is now ( ) | (NULL L) asks if the argument, L, is the null list. If it is, then the value of the function is T. If it is not the null list, then we ask the next question. In this case, ( ) <u>is</u> the null list. <u>Therefore, the value of the function (ISLAT L), where L is (BACON AND EGGS), is T, or true.</u> |
| 2.27) | Do you remember the problem about<br>(ISLAT L)? | Probably not. The function (ISLAT L) has a value of T if the list, L, is a list of atoms, where L is (BACON AND EGGS). |
| 2.28) | Can you describe what the function ISLAT does, in your own words? | Here are our words: "ISLAT looks at each S-expression, in turn, and asks if each S-expression is an atom, until it runs out of S-expressions. When it runs out without encountering a list, the function's value is T. As soon as it finds a list, the function value is F --false. To see how we could arrive at a value of "false", consider the next few questions." |

| | | |
|---|---|---|
| 2.29) | This is the function (ISLAT L), again:<br>`(ISLAT (LAMBDA (L)`<br>`    (COND`<br>`    ((NULL L) T)`<br>`    ((ATOM (CAR L))(ISLAT (CDR L)))`<br>`    (T F)`<br>`        ) ))`<br>What is the value of (ISLAT L), where<br>L is now (BACON (AND EGGS))? | False,<br>since the list L contains an S-expression that is a list. |
| 2.30) | What is the first question to be asked? | (NULL L) |
| 2.31) | What is the meaning of the line<br>((NULL L) T)<br>where<br>L is (BACON (AND EGGS)) | (NULL L) asks if L is the null list.<br>If it is, the value of the function is T. If L is not null, then move to the next question.<br>In this case, it is not null, so we ask the next question. |
| 2.32) | What is the next question: | (ATOM (CAR L)) |
| 2.33) | What is the meaning of the line<br>((ATOM (CAR L))(ISLAT (CDR L)))<br>where<br>L is (BACON (AND EGGS)) | (ATOM (CAR L)) asks if (CAR L) is an atom.<br>If it is, the value of the function is<br>(ISLAT (CDR L)).<br>If it is not, we ask the next question. In this case, (CAR L) is an atom, so we want to check if the rest of the list, L, is composed only of atoms. |
| 2.34) | What is the meaning of<br>(ISLAT (CDR L))? | (ISLAT (CDR L)) checks to see if the rest of the list, L, is composed only of atoms, by referring us back to (ISLAT L) with L replaced by (CDR L). |
| 2.35) | What is the meaning of the line<br>((NULL L) T)<br>where<br>L is now ((AND EGGS)) | (NULL L) asks if L is the null list.<br>If it is null, the value of the function is T.<br>If it is not null, we ask the next question.<br>In this case, L is not null, so move to the next question. |
| 2.36) | What is the next question? | (ATOM (CAR L)) |
| 2.37) | What is the meaning of the line<br>((ATOM (CAR L))(ISLAT (CDR L)))<br>where<br>L is now ((AND EGGS)) | (ATOM (CAR L)) asks if (CAR L) is an atom.<br>If it is, then the value of the function is<br>(ISLAT (CDR L)).<br>If it is not, then we move to the next question.<br>In this case, (CAR L) is not an atom, so we ask the next question. |
| 2.38) | What is the next question? | T |
| 2.39) | What is the meaning of the question, T? | T asks if T is true. |
| 2.40) | Is T true? | Yes,<br>because the question T is <u>always</u> true! |
| 2.41) | T | T |
| 2.42) | Why is T the last question? | Because we do not need to ask any more questions. |
| 2.43) | What is the meaning of the line<br>(T F) | T asks if T is true.<br>If T is true --<u>as it always is</u>-- then the value of the function is <u>F</u> -- false. |

2.44) What is the meaning of the line
```
        )   ))
```
These are just the closing parentheses which match "(COND", "(LAMBDA", and "(ISLAT" at the beginning of the description of the function. We call these "aggravation parentheses", and they are always put at the end of a function.

---

2.45) Can you describe how we arrived at the value F for
    (ISLAT L)
where
    L is (BACON (AND EGGS))?

Here is one way to say it:
"(ISLAT L) looks at each S-expression in its argument, to see if it is an atom. If it runs out of S-expressions before it finds a list, the value of (ISLAT L) is T. If it finds a list, as it did in the example (BACON (AND EGGS)), the value of (ISLAT L) is F,"

---

2.46) Is it true, or false, that A is a member of LAT,
where
    A is the argument TEA, and
    LAT is the argument (COFFEE TEA OR MILK)

True,
because one of the atoms of the LAT
    (COFFEE TEA OR MILK)
is the same as the atom A, TEA.

---

2.47) Is (MEMBER A LAT) true, or false, where
    A is POACHED, and
    LAT is (FRIED EGGS AND SCRAMBLED EGGS)

False,
since A is not one of the atoms of the LAT.

---

2.48) This is the function (MEMBER A LAT):

```
(MEMBER (LAMBDA (A LAT)
    (COND
    ((NULL LAT) F)
    ((EQ (CAR LAT) A) T)
    (T (MEMBER A (CDR LAT)))
             )  ))
```

What is the value of (MEMBER A LAT), where
    A is MEAT, and
    LAT is (MASHED POTATOES AND MEAT GRAVY)

True;
because the atom MEAT is one of the atoms of the LAT, (MASHED POTATOES AND MEAT GRAVY).

---

2.49) How do you arrive at the value T for the above function?

The value is arrived at by asking the questions about (MEMBER A LAT). HINT: Write down the function (MEMBER A LAT), and refer to it while you work on the next group of questions.

---

2.50) What is the first question asked by
    (MEMBER A LAT)

(NULL LAT)
Note: This is also the first question asked by (ISLAT L).

---

COMMANDMENT No. 1

Thou shalt always ask NULL as the first question in describing any function.

2.51) What is the meaning of the line
    ((NULL LAT) F)
where
    LAT is (MASHED POTATOES AND MEAT GRAVY)

(NULL LAT) asks if the LAT is the null list. If it is, then the value of the function is F, false, since the atom MEAT was not found in the LAT. If it is not, then we ask the next question. In this case, it is not null, so we ask the next question.

---

2.52) What is the next question?

(EQ (CAR LAT) A)

---

2.53) What is the meaning of the line
    ((EQ (CAR LAT) A) T)
where
    A is MEAT, and
    LAT is (MASHED POTATOES AND MEAT GRAVY)

(EQ (CAR LAT) A) asks if the CAR of LAT is the same as the atom A.
If it is, then the value of the function is T.
If it is not, then we ask the next question. In this case, it is not, so we ask the next question.

---

| | | |
|---|---|---|
| 2.54) | What is the next question? | T |
| 2.55) | Is T <u>really</u> a question? | Yes,<br>T is a question whose answer is always true. |
| 2.56) | Give an example of a question to which the answer is contained in the question. | Here's ours:<br>"What color is an orange?" |
| 2.57) | What is the meaning of the line<br>(T (MEMBER A (CDR LAT))) | Now that we know that the LAT is not null, and we know that the CAR of LAT is not the same atom as A, we want to find out if the atom A is somewhere in the rest of the LAT. The function does this by referring us back to the original function, with the argument LAT replaced by (CDR LAT). |
| 2.58) | What are the arguments for MEMBER now? | A is MEAT, and<br>LAT is now (CDR LAT) -- (POTATOES AND MEAT GRAVY). |
| 2.59) | What is the next question? | (NULL LAT).<br>Remember COMMANDMENT No. 1? |
| 2.60) | Is (NULL LAT) true, or false, where<br>LAT is (POTATOES AND MEAT GRAVY) | F -- false. |
| 2.61) | What do we do now? | Ask the next question. |
| 2.62) | What is the next question? | (EQ (CAR LAT) A). |
| 2.63) | What is (EQ (CAR LAT) A), where<br>A is MEAT, and<br>LAT is (POTATOES AND MEAT GRAVY) | F -- false. |
| 2.64) | What do we do now? | Ask the next question. |
| 2.65) | What is the next question? | T |
| 2.66) | What is T? | T -- true. |
| 2.67) | What is the meaning of the line<br>(T (MEMBER A (CDR LAT))) ? | (MEMBER A (CDR LAT)) finds out if A is a member of the CDR of the LAT, by referring us back to the original function. |
| 2.68) | What are the arguments of MEMBER now? | A is MEAT, and<br>LAT is (AND MEAT GRAVY). |
| 2.69) | What is the next question? | (NULL LAT) |
| 2.70) | What do we do now? | Ask the next question,<br>since (NULL LAT) is false. |
| 2.71) | What is the next question? | (EQ (CAR LAT) A) |
| 2.72) | What do we do now? | Ask the next question, |
| 2.73) | What is the next question? | T |
| 2.74) | What is the value of the line<br>(T (MEMBER A (CDR LAT))) | (MEMBER A (CDR LAT)) |
| 2.75) | What do we do now? | <u>Recurse</u> -- refer to the original function, with new arguments. |
| 2.76) | What are the new arguments? | A is MEAT, and<br>LAT is (MEAT GRAVY) |

| | | |
|---|---|---|
| 2.77) | What is the next question? | (NULL LAT) |
| 2.78) | What do we do now? | Ask the next question, since (NULL LAT) is false. |
| 2.79) | What is the next question? | (EQ (CAR LAT) A) |
| 2.80) | What is the value of the line<br>((EQ (CAR LAT) A) T) | T,<br>because (CAR LAT), which is MEAT, and A, which is MEAT, are the same atom. |
| 2.81) | What is the value of the function<br>(MEMBER A LAT)<br>where<br>A is MEAT, and<br>LAT is (MEAT GRAVY) | T,<br>because we have now found that MEAT is a <u>member</u> of (MEAT GRAVY). |
| 2.82) | What is the value of the function<br>(MEMBER A LAT)<br>where<br>A is MEAT, and<br>LAT is (AND MEAT GRAVY) | T,<br>because MEAT is also a member of the LAT<br>(AND MEAT GRAVY). |
| 2.83) | What is the value of the function<br>(MEMBER A LAT)<br>where<br>A is MEAT, and<br>LAT is (POTATOES AND MEAT GRAVY) | T,<br>because MEAT is also a member of the LAT<br>(POTATOES AND MEAT GRAVY). |
| 2.84) | What is the value of the function<br>(MEMBER A LAT)<br>where<br>A is MEAT, and<br>LAT is (MASHED POTATOES AND MEAT GRAVY) | T,<br>because MEAT is also a member of the LAT<br>(MASHED POTATOES AND MEAT GRAVY).<br>Of course, you noticed that this is our original LAT. |
| 2.85) | Just to make sure you have it right, let's run through it again quickly:<br>What is the value of<br>`(MEMBER (LAMBDA (A LAT)`<br>`(COND`<br>`((NULL LAT) F)`<br>`((EQ (CAR LAT) A) T)`<br>`(T (MEMBER A (CDR LAT)))`<br>`) ))`<br>where<br>A is MEAT, and<br>LAT is (MASHED POTATOES AND MEAT GRAVY) | T.<br>HINT: Write down the function MEMBER and its arguments and refer to them as you go through the next bunch of questions. |
| 2.86) | (NULL LAT) | No,<br>move to the next line. |
| 2.87) | (EQ (CAR LAT) A) | No,<br>move to the next line. |
| 2.88) | T | Yes,<br>recurse with A and (CDR LAT), where<br>A is MEAT and<br>(CDR LAT) is (POTATOES AND MEAT GRAVY). |
| 2.89) | (NULL LAT) | No,<br>move to the next line. |
| 2.90) | (EQ (CAR LAT) A) | No,<br>move to the next line. |
| 2.91) | T | Yes,<br>recurse with A and (CDR LAT), where<br>A is MEAT, and<br>(CDR LAT) is (AND MEAT GRAVY) |

| | | |
|---|---|---|
| 2.92) | (NULL LAT) | No,<br>move to the next line. |
| 2.93) | (EQ (CAR LAT) A) | No,<br>move to the next line. |
| 2.94) | T | Yes,<br>recurse with A and (CDR LAT), where<br>A is MEAT, and<br>(CDR LAT) is (MEAT GRAVY) |
| 2.95) | (NULL LAT) | No,<br>move to the next line. |
| 2.96) | (EQ (CAR LAT) A) | Yes,<br>the value of this function is T. |
| 2.97) | What is the value of (MEMBER A LAT), where<br>A is MEAT, and<br>LAT is (MEAT GRAVY) | T |
| 2.98) | What is the value of (MEMBER A LAT), where<br>A is MEAT, and<br>LAT is (AND MEAT GRAVY) | T |
| 2.99) | What is the value of (MEMBER A LAT), where<br>A is MEAT, and<br>LAT is (POTATOES AND MEAT GRAVY) | T |
| 2.100) | What is the value of (MEMBER A LAT), where<br>A is MEAT, and<br>LAT is (MASHED POTATOES AND MEAT GRAVY) | T<br>(T sometimes appears as *T*). |
| 2.101) | What is the value of (MEMBER A LAT), where<br>A is LIVER, and<br>LAT is (BAGELS AND LOX) | F<br>(F sometimes appears as NIL) |
| 2.102) | (NULL LAT) | No,<br>move to the next line. |
| 2.103) | (EQ (CAR LAT) A) | No,<br>move to the next line. |
| 2.104) | T | Yes,<br>recurse with A and (CDR LAT), where<br>A is LIVER, and<br>(CDR LAT) is (AND LOX) |
| 2.105) | (NULL LAT) | No,<br>move to the next line. |
| 2.106) | (EQ (CAR LAT) A) | No,<br>move to the next line. |
| 2.107) | T | Yes,<br>recurse with A and (CDR LAT), where<br>A is LIVER, and<br>(CDR LAT) is (LOX) |
| 2.108) | (NULL LAT) | No,<br>move to the next line. |
| 2.109) | (EQ (CAR LAT) A) | No,<br>move to the next line. |
| 2.110) | T | Yes,<br>recurse with A and (CDR LAT), where<br>A is LIVER, and<br>(CDR LAT) is ( ) |

| | | |
|---|---|---|
| 2.111) | (NULL LAT) | Yes. |
| 2.112) | What is the value of (MEMBER A LAT), where<br>A is LIVER, and<br>LAT is ( ) | F |
| 2.113) | What is the value of (MEMBER A LAT), where<br>A is LIVER, and<br>LAT is (LOX) | F |
| 2.114) | What is the value of (MEMBER A LAT), where<br>A is LIVER, and<br>LAT is (AND LOX) | F |
| 2.115) | What is the value of (MEMBER A LAT), where<br>A is LIVER, and<br>LAT is (BAGELS AND LOX) | F |

| | | |
|---|---|---|
| 3.1) | What is (REMBER A LAT), where<br>A is MINT, and<br>LAT is (LAMB CHOPS AND MINT JELLY) | (LAMB CHOPS AND JELLY)<br>REMBER stands for REMove the memBER. |
| 3.2) | (REMBER A LAT), where<br>A is MINT, and<br>LAT is (LAMB CHOPS AND MINT FLAVORED MINT JELLY) | (LAMB CHOPS AND FLAVORED MINT JELLY) |
| 3.3) | (REMBER A LAT), where<br>A is TOAST, and<br>LAT is (BACON LETTUCE AND TOMATO) | (BACON LETTUCE AND TOMATO) |
| 3.4) | (REMBER A LAT), where<br>A is CUP, and<br>LAT is (COFFEE CUP TEA CUP AND HICK CUP) | (COFFEE TEA CUP AND HICK CUP) |
| 3.5) | What does (REMBER A LAT) do? | It takes an atom and a LAT as its arguments, and makes a new LAT with the first occurrence of the atom in the old LAT removed. |
| 3.6) | What steps will we use to do this? | First, we will compare A with (CAR LAT). We will want to build a list, from left to right. |
| 3.7) | How do we ask if A is the same as (CAR LAT)? | (EQ (CAR LAT) A) |
| 3.8) | What would be the value of (REMBER A LAT) if A is the same as (CAR LAT)? | (CDR LAT) |
| 3.9) | What do we do if A is not the same as (CAR LAT)? | We will want to keep (CAR LAT), but also find out if A is somewhere in the rest of the LAT. |
| 3.10) | How do we find out if A is somewhere in the rest of LAT? | (REMBER A (CDR LAT)) |

3.11) Let us now use the ideas developed so far, to write the function REMBER:

```
(REMBER (LAMBDA (A LAT)
    (COND
    ((EQ (CAR LAT) A)(CDR LAT))
    (T                (REMBER A (CDR LAT)))
              ) ))
```

What is missing from here?

Obviously, we have forgotten COMMANDMENT No. 1! Now, rewrite (REMBER A LAT).

3.12) Now, we think that this is the function REMBER:

```
(REMBER (LAMBDA (A LAT)
    (COND
    ((NULL LAT)( ))
    ((EQ (CAR LAT) A)(CDR LAT))
    (T                (REMBER A (CDR LAT)))
            ) ))
```

What is the value of (REMBER A LAT), where
A is BACON, and
LAT is (BACON LETTUCE AND TOMATO)

(LETTUCE AND TOMATO)
HINT: Write down the function REMBER and its arguments, and refer to them as you go through the next sequence of questions.

| | | |
|---|---|---|
| 3.13) | Now, let's see if this function works. What is the first question? | (NULL LAT) |
| 3.14) | What do we do now? | Move to the next line and ask the next question. |

| | | |
|---|---|---|
| 3.15) | (EQ (CAR LAT) A) | Yes,<br>so the value of the function is (CDR LAT).<br>In this case, it is the list<br>(LETTUCE AND TOMATO) |
| 3.16) | Is this the correct value of the function? | Yes,<br>because the above list is the original list without the atom BACON. |
| 3.17) | But did we really use a good example? | Who knows?<br>But the proof of the pudding is in the eating, so let's try another example. |
| 3.18) | What does (REMBER A LAT) do? | It takes an atom and a LAT as its arguments, and makes a new LAT with the first occurrence of the atom in the old LAT removed. |
| 3.19) | What steps will we use to do this? | First, we will compare each atom of the LAT with the atom A.<br>Second, we want to build a list from left to right. |
| 3.20) | What is the value of (REMBER A LAT), where<br>A is AND, and<br>LAT is (BACON LETTUCE AND TOMATO) | (BACON LETTUCE TOMATO) |
| 3.21) | Let us see if this function works.<br>What is the first question asked by REMBER? | (NULL LAT) |
| 3.22) | What do we do now? | Move to the next line, and ask the next question. |
| 3.23) | (EQ (CAR LAT) A) | No,<br>so move to the next line. |
| 3.24) | What is the meaning of the line<br>(T (REMBER A (CDR LAT))) | T asks if T is true -- as it always is -- and the rest of the line says to recurse with A and (CDR LAT), where<br>A is AND, and<br>(CDR LAT) is (LETTUCE AND TOMATO) |
| 3.25) | (NULL LAT) | No,<br>so move to the next line. |
| 3.26) | (EQ (CAR LAT) A) | No,<br>so move to the next line. |
| 3.27) | What is the meaning of the line<br>(T (REMBER A (CDR LAT))) | Recurse, where<br>A is AND, and<br>(CDR LAT) is (AND TOMATO) |
| 3.28) | (NULL LAT) | No,<br>so move to the next line, and ask the next question. |
| 3.29) | (EQ (CAR LAT) A) | Yes. |
| 3.30) | What is the value of the function<br>(REMBER A LAT) | (CDR LAT) -- (TOMATO). |
| 3.31) | Is this correct? | No,<br>since (TOMATO) is not the list<br>(BACON LETTUCE AND TOMATO)<br>with only A --AND-- removed. |
| 3.32) | What did we do wrong? | We dropped AND, but we also lost all the atoms preceeding AND. |
| 3.33) | How can we keep from losing the atoms<br>BACON and LETTUCE? | We use CONS "The Magnificent".<br>Remember CONS, from Chapter One? |

COMMANDMENT No. 2

**Thou shalt use CONS to build lists.**

| | | |
|---|---|---|
| 3.34) | Let's just see what happens when we use CONS:<br>`(REMBER (LAMBDA (A LAT)`<br>`    (COND`<br>`    ((NULL LAT)( ))`<br>`    ((EQ (CAR LAT) A)(CDR LAT))`<br>`    (T (CONS (CAR LAT)(REMBER A (CDR LAT))))`<br>`              ) ))`<br>Now, what is the value of (REMBER A LAT), where<br>A is AND, and<br>LAT is (BACON LETTUCE AND TOMATO) | (BACON LETTUCE TOMATO)<br>Make a copy of this function with CONS and the arguments A and LAT, so you can refer to it for the following questions. |
| 3.35) | What is the first question? | (NULL LAT) |
| 3.36) | What do we do now? | Move to the next line of the function and ask the next question. |
| 3.37) | (EQ (CAR LAT) A) | No,<br>so move to the next line. |
| 3.38) | What is the meaning of the line<br>(T (CONS (CAR LAT)(REMBER A (CDR LAT))))<br>where<br>A is AND, and<br>LAT is (BACON LETTUCE AND TOMATO) | CONS (CAR LAT) --BACON-- onto the value of (REMBER A (CDR LAT)). But since we don't know the value of (REMBER A (CDR LAT)) yet, we will have to find this value before we can CONS (CAR LAT) onto it. |

Draw a picture of "CONS The Magnificent" on this page.

| | | |
|---|---|---|
| 3.39) | What is the meaning of (REMBER A (CDR LAT)) | This refers us back to the original function, with LAT replaced by (CDR LAT) -- (LETTUCE AND TOMATO). |
| 3.40) | (NULL LAT) | No,<br>so move to the next line. |
| 3.41) | (EQ (CAR LAT) A) | No,<br>so move to the next line. |
| 3.42) | What is the meaning of the line<br>(T (CONS (CAR LAT)(REMBER A (CDR LAT)))) | CONS (CAR LAT) --LETTUCE-- onto the value of (REMBER A (CDR LAT)).<br>But since we don't know that value, we will have to find that value before we can CONS (CAR LAT) onto our list. |
| 3.43) | What is the meaning of (REMBER A (CDR LAT))? | This refers us back to the original function with LAT replaced by (CDR LAT) -- (AND TOMATO). |
| 3.44) | (NULL LAT) | No,<br>so move to the next line. |
| 3.45) | (EQ (CAR LAT) A) | Yes. |
| 3.46) | What is the value of the line<br>((EQ (CAR LAT) A)(CDR LAT)) | (CDR LAT) -- (TOMATO). |
| 3.47) | Are we finished? | Certainly not!<br>So far we know what (REMBER A LAT) is when LAT is (AND TOMATO), but we don't yet know what it is when LAT is (LETTUCE AND TOMATO) or (BACON LETTUCE AND TOMATO). |
| 3.48) | We now have a value for (REMBER A (CDR LAT)), where<br>A is AND, and<br>(CDR LAT) is (AND TOMATO)<br>This value is (TOMATO). This is not the final value, so what must we do? | Recall that we wanted to CONS LETTUCE onto the value of (REMBER A (CDR LAT)), where A was AND, and (CDR LAT) was (AND TOMATO). Now that we have this value, which is (TOMATO), we can CONS LETTUCE onto this value. Reread the last six steps. |
| 3.49) | What is the result when we CONS LETTUCE onto (TOMATO) | (LETTUCE TOMATO) |
| 3.50) | What does (LETTUCE TOMATO) represent? | It represents the value of (REMBER A (CDR LAT)). when<br>A was AND, and<br>(CDR LAT) was (AND TOMATO) |
| 3.51) | Are we finished yet? | Not quite.<br>So far we know what (REMBER A LAT) is when<br>LAT is (LETTUCE AND TOMATO),<br>but we don't yet know what it is when<br>LAT is (BACON LETTUCE AND TOMATO) |
| 3.52) | We now have a value for (REMBER A (CDR LAT)) when<br>A was AND, and<br>(CDR LAT) was (LETTUCE AND TOMATO).<br>This value was (LETTUCE TOMATO). This is not the final value, so what must we do again? | Recall that, at one time, we wanted to CONS BACON onto the value of (REMBER A (CDR LAT)), when<br>A was AND, and<br>(CDR LAT) was (LETTUCE AND TOMATO)<br>Now that we have this value, which is<br>(LETTUCE TOMATO),<br>we can CONS BACON onto this value. |
| 3.53) | What is the result when we CONS BACON onto (LETTUCE TOMATO) | (BACON LETTUCE TOMATO) |

| | | |
|---|---|---|
| 3.54) | What does (BACON LETTUCE TOMATO) represent? | It represents the value of<br>    (CONS (CAR LAT)(REMBER A (CDR LAT))),<br>when<br>    A was AND, and<br>    (REMBER A (CDR LAT)) was (LETTUCE TOMATO) |
| 3.55) | Are we finished yet? | Yes. |
| 3.56) | Can you put in your own words how we arrived at the final value<br>    (BACON LETTUCE TOMATO) | In our words:<br>"REMBER checked each atom of the LAT, one at a time, to see if it was the same as the atom AND. If the CAR was not the same as the atom, we saved it to be CONS'd to the final value later. When REMBER found the atom AND, it dropped it, and CONS'd the previous atoms onto the rest of the LAT, in reverse order." |
| 3.57) | What is the value of the function<br>`(REMBER (LAMBDA (A LAT)`<br>`    (COND`<br>`    ((NULL LAT)( ))`<br>`    ((EQ (CAR LAT) A)(CDR LAT))`<br>`    (T (CONS (CAR LAT)(REMBER A (CDR LAT))))`<br>`              ) ))`<br>where<br>    A is AND, and<br>    LAT is (BACON LETTUCE AND TOMATO) | (BACON LETTUCE TOMATO)<br>HINT: Write down the function REMBER and its arguments and refer to them as you go through the next sequence of questions. |
| 3.58) | (NULL LAT) | No. |
| 3.59) | (EQ (CAR LAT) A) | No. |
| 3.60) | T | T,<br>so the value is<br>    (CONS (CAR LAT)(REMBER A (CDR LAT))) |
| 3.61) | What is the meaning of<br>    (CONS (CAR LAT)(REMBER A (CDR LAT))) | This says to refer back to the original function REMBER, but with the argument LAT replaced by (CDR LAT), and that <u>after</u> we arrived at a value for (REMBER A (CDR LAT)) we will CONS (CAR LAT) --BACON-- onto it. |
| 3.62) | (NULL LAT) | No. |
| 3.63) | (EQ (CAR LAT) A) | No. |
| 3.64) | T | T,<br>so the value is<br>    (CONS (CAR LAT)(REMBER A (CDR LAT))) |
| 3.65) | What is the meaning of<br>    (CONS (CAR LAT)(REMBER A (CDR LAT))) | This says we recurse back to the original function REMBER, with the argument LAT replaced by (CDR LAT), and that <u>after</u> we arrive at a value for (REMBER A (CDR LAT)), we will CONS (CAR LAT) --LETTUCE-- onto it. |
| 3.66) | (NULL LAT) | No. |
| 3.67) | (EQ (CAR LAT) A) | Yes. |
| 3.68) | What is the value of the line | (CDR LAT) -- (TOMATO) |
| 3.69) | Now what? | CONS (CAR LAT) --LETTUCE-- onto (TOMATO) --see answer #3.60 forming (LETTUCE TOMATO). |

3.70) Now what?

CONS (CAR LAT) --BACON-- onto (LETTUCE TOMATO) -- see answer #3.64 forming (BACON LETTUCE TOMATO).

---

3.71) Now that we have completed REMBER, try this example:
(REMBER A LAT), where
A is SAUCE, and
LAT is (SOY SAUCE AND TOMATO SAUCE)

(REMBER A LAT) is (SOY AND TOMATO SAUCE)

---

3.72) What is (FIRSTS L), where
L is
((APPLE PEACH PUMPKIN)(PLUM PEAR CHERRY)
(GRAPE RAISIN PEA)(BEAN CARROT EGGPLANT))

(APPLE PLUM GRAPE BEAN)

---

3.73) What is (FIRSTS L), where
L is ((A B)(C D)(E F))

(A C E)

---

3.74) What is (FIRSTS L), where
L is ( )

( )

---

3.75) What is (FIRSTS L), where
L is ((FIVE PLUMS)(FOUR)(ELEVEN GREEN ORANGES))

(FIVE FOUR ELEVEN)

---

3.76) In your own words, what does (FIRSTS L) do?

We tried the following:
"FIRSTS takes one argument, a list, which must either be a null list, or contain one or more non-null lists. It builds another list composed of the first S-expression of each internal list."

---

3.77) See if you can write the functions FIRSTS.
Remember the COMMANDMENTS!

Believe it or not, you can probably write the following:

```
(FIRSTS (LAMBDA (L)
   (COND
   ((NULL L)     )
   (T (CONS             (FIRSTS (CDR L))))
             )  ))
```

---

3.78) Why (FIRSTS (LAMBDA (L)?

Because we always state the function, then "(LAMBDA", then the arguments of the function.

---

3.79) Why (COND ?

Because it is a necessary part of the function, and must always be used.

---

3.80) Why ((NULL L)     ) ?

COMMANDMENT No. 1.

---

3.81) Why (T ?

Because the last line always begins with (T. The reason for this is clarified as more examples are considered.

---

3.82) Why (CONS ?

Because we are building a list --COMMANDMENT No. 2.

---

3.83) Why (FIRSTS (CDR L)) ?

Because we can only look at one S-expression at a time. In order to do this, we must recurse.

---

3.84) Why )  )) ?

Because these are the matching parentheses for (COND and the first line, and they always appear at the end of a function definition.

---

3.85) Keeping in mind the definition of (FIRSTS L), what is a typical element of the value of (FIRSTS L), where
L is ((A B)(C D)(E F))

A

---

| | | |
|---|---|---|
| 3.86) | What is another typical element? | C, or E. |
| 3.87) | Suppose there was a function (SECONDS L). What would be a typical element of the value of (SECONDS L), where<br>    L is ((A B)(C D)(E F)) | B, or D, or F. |
| 3.88) | How do we describe a typical element for<br>    (FIRSTS L) | By taking the CAR of (CAR L) -- (CAR (CAR L)). See Chapter 1. |
| 3.89) | As we find a typical element of (FIRSTS L), what do we do with it? | We CONS it onto the recursion -- (FIRSTS (CDR L)). |

COMMANDMENT No. 3

> Thou shalt always realize when building a list, thou need only describe the first typical element, and then CONS it onto the natural recursion.
>
> NOTE: You have just read THE most important statement in this book. Please read it again.

| | | |
|---|---|---|
| 3.90) | From COMMANDMENT No. 3, we can now fill in more of the function (FIRSTS L). What does the last line look like now? | (T (CONS (CAR (CAR L))(FIRSTS (CDR L))))<br>TYPICAL ELEMENT NATURAL RECURSION |
| 3.91) | What does this function do?<br>(FIRSTS (LAMBDA (L)<br>    (COND<br>    ((NULL L)     )<br>    (T (CONS (CAR (CAR L))(FIRSTS (CDR L))))<br>            )  ))<br>where<br>    L is ((A B)(C D)(E F)). | Nothing yet.<br>We are still missing one important ingredient in our recipe. The line ((NULL L)     ) needs a value for the case where L is the null list. We can, however, proceed without it for now. |
| 3.92) | (NULL L), where<br>    L is ((A B)(C D)(E F)) | No,<br>so move to the next line. |
| 3.93) | What is the meaning of the line<br>    (T (CONS (CAR (CAR L))(FIRSTS (CDR L)))) | It saves (CAR (CAR L)) to CONS it onto (FIRSTS (CDR L)), when it finds that value. To find (FIRSTS (CDR L)), we recurse to the original function, (FIRSTS L), with the new argument (CDR L). |
| 3.94) | (NULL L), where<br>    L is ((C D)(E F)) | No,<br>so move to the next line. |
| 3.95) | What is the meaning of the line<br>    (T (CONS (CAR (CAR L))(FIRSTS (CDR L)))) | Save (CAR (CAR L)), and recurse with (FIRSTS (CDR L)). |
| 3.96) | (NULL L), where<br>    L is ((E F)) | No,<br>so move to the next line. |
| 3.97) | What is the meaning of the line<br>    (T (CONS (CAR (CAR L))(FIRSTS (CDR L)))) | Save (CAR (CAR L)), and recurse with (FIRSTS (CDR L)). |
| 3.98) | (NULL L) | Yes. |

3.99) Now, what is the value of the line
((NULL L) )

There is no value; something is missing.

---

3.100) What do we need to CONS atoms onto?

A list.
Remember PRINCIPLE No. 3 -- see Chapter 1.

---

3.101) What value can we give the function in the case that (NULL L) is true, for the purpose of CONSing?

Since the final value must be a list, we cannot use T or F as our value, so how about ( )?

---

3.102) With ( ) as a value, we now have three CONS steps to go back and pick up.

I. We need to: 1. CONS E onto ( ).
2. CONS C onto the value of 1.
3. CONS A onto the value of 2.

or, alternatively,

II. We need to: 1. CONS A onto the value of 2.
2. CONS C onto the value of 3.
3. CONS E onto ( ).

or, alternatively,

III. We need to: CONS A onto the CONS of C onto the CONS of E onto ( ).

In any case, what is the final value of (FIRSTS L) ?

(A C E).

---

3.103) With which of the three alternatives are you most comfortable?

Correct!
Now you use that one.

---

3.104) (INSERTR OLD NEW LAT), where
OLD is FUDGE
NEW is TOPPING, and
LAT is (ICE CREAM WITH FUDGE FOR DESSERT)

(ICE CREAM WITH FUDGE TOPPING FOR DESSERT)

---

3.105) (INSERTR OLD NEW LAT), where
OLD is AND
NEW is JALAPENO, and
LAT is (TACOS TAMALES AND SALSA)

(TACOS TAMALES AND JALAPENO SALSA)

---

3.106) (INSERTR OLD NEW LAT), where
OLD is D
NEW is E, and
LAT is (A B C D F G D H)

(A B C D E F G D H)

---

3.107) In your own words, what does
(INSERTR OLD NEW LAT)
do?

In our own words:
"It needs three arguments: the atoms OLD and NEW, and a LAT. INSERTR builds a LAT with NEW inserted to the right of the first occurrence of OLD."

---

3.108) See if you can write the function
(INSERTR OLD NEW LAT).

```
(INSERTR (LAMBDA (OLD NEW LAT)
    (COND
    ((NULL LAT)( ))
    ((EQ (CAR LAT) OLD)              (CDR LAT))
    (T (CONS (CAR LAT)(INSERTR OLD NEW (CDR LAT))))
                       ) ))
```

If you were unable to write this much, you should review the PRINCIPLES and COMMANDMENTS.

---

3.109) What is the value of the INSERTR we just wrote, where
OLD is FUDGE
NEW is TOPPING, and
LAT is (ICE CREAM WITH FUDGE FOR DESSERT)

(ICE CREAM WITH FOR DESSERT)

---

3.110) Notice that so far, this is the same as REMBER; but for (INSERTR OLD NEW LAT), what do we do, where (EQ (CAR LAT) OLD) is true?

When (CAR LAT) is the same as OLD, we want to insert NEW to the right.

---

3.111) How is this done?

Let's try CONS NEW onto (CDR LAT).

---

3.112) Now we have

```
(INSERTR (LAMBDA (OLD NEW LAT)
    (COND
    ((NULL LAT)( ))
    ((EQ (CAR LAT) OLD)       (CONS NEW (CDR LAT)))
    (T (CONS (CAR LAT)(INSERTR OLD NEW (CDR LAT))))
              )  ))
```

What is (INSERTR OLD NEW LAT), where
    OLD is FUDGE
    NEW is TOPPING, and
    LAT is (ICE CREAM WITH FUDGE FOR DESSERT)

(ICE CREAM WITH TOPPING FOR DESSERT)

---

3.113) Is this the list we wanted?

No,
we have only replaced FUDGE with TOPPING.

---

3.114) What still needs to be done?

Somehow we need to include the atom which is the same as OLD before the atom NEW.

---

3.115) How can we include OLD before NEW?

Try CONSing OLD onto
    (CONS NEW (CDR LAT))

---

3.116) Now you should be able to write the rest of the function
    (INSERTR OLD NEW LAT)
Do it.

```
(INSERTR (LAMBDA (OLD NEW LAT)
    (COND
    ((NULL LAT)( ))
    ((EQ (CAR LAT) OLD)(CONS OLD (CONS NEW (CDR LAT))))
    (T (CONS (CAR LAT)(INSERTR OLD NEW (CDR LAT))))
                          )  ))
```

Notice that, where OLD is FUDGE, NEW is TOPPING, and LAT is (ICE CREAM WITH FUDGE FOR DESSERT), the value of the function is
    (ICE CREAM WITH FUDGE TOPPING FOR DESSERT).
If you got this right, have one.

---

3.117) Now try (INSERTL OLD NEW LAT).
HINT: INSERTL inserts the atom NEW to the left of the first occurrence of the atom OLD.

This much is trivial -- right?

```
(INSERTL (LAMBDA (OLD NEW LAT)
    (COND
    ((NULL LAT)( ))
    ((EQ (CAR LAT) OLD)(CONS NEW (CONS (CAR LAT)
                                       (CDR LAT))))
    (T (CONS (CAR LAT)(INSERTL OLD NEW (CDR LAT))))
                   )  ))
```

---

3.118) Did you think of a different way to do that?

For example, this line could have been:
    ((EQ (CAR LAT) OLD)(CONS NEW (CONS OLD (CDR LAT))))
or even better, like this:
    ((EQ (CAR LAT) OLD)(CONS NEW LAT)),
since (CONS OLD (CDR LAT)) is the same as LAT.

---

3.119) Now try to write (SUBST OLD NEW LAT).
HINT: (SUBST OLD NEW LAT) replaces the first occurrence of OLD in the LAT with the atom NEW.
For example, where
    OLD is FUDGE
    NEW is TOPPING, and
    LAT is (ICE CREAM WITH FUDGE FOR DESSERT),
the value is
    (ICE CREAM WITH TOPPING FOR DESSERT)
Now you have the idea.

Obviously:

```
(SUBST (LAMBDA (OLD NEW LAT)
    (COND
    ((NULL LAT)( ))
    ((EQ (CAR LAT) OLD)(CONS NEW (CDR LAT)))
    (T (CONS (CAR LAT)(SUBST OLD NEW (CDR LAT))))
                  )  ))
```

Notice that this is the same as our second attempt at INSERTR.

---

Now go CONS a piece of cake into your mouth.

| | | |
|---|---|---|
| 4.1) | Is 14 an atom? | Yes, it is, because we are expanding the value of ATOM to include all numbers. |
| 4.2) | Is (ATOM N) true or false where N is 14 | T, because 14 is an atom. This extends the definition of what an atom can be. |
| 4.3) | Is -3 a number? | No, since negative values are not considered here. |
| 4.4) | Is 3.1415 a number? | No, since only whole numbers are considered here. |
| 4.5) | Are -3 and 3.1415 numbers available in LISP | Yes, however, only whole numbers are considered here. |
| 4.6) | What is (ADD1 N), where N is 67 | 68 |
| 4.7) | What is (SUB1 N), where N is 5 | 4 |
| 4.8) | Is (ZEROP N) true, or false, where N is 0 | T. |
| 4.9) | Is (ZEROP N) true, or false, where N is 1492 | F. |
| 4.10) | Can you write a function (GREATERP N1 N2), using (ZEROP N), (ADD1 N), and (SUB1 N)? (GREATERP N1 N2), where N1 is 12, and N2 is 11, has the value T: where N1 is 6, and N2 is 45, has the value F. | How about: `(GREATERP (LAMBDA (N1 N2) (COND ((ZEROP N2) T) ((ZEROP N1) F) (T (GREATERP (SUB1 N1)(SUB1 N2))) ) ))` |
| 4.11) | Did we just violate COMMANDMENT No. 1? | Yes, however, we can treat ZEROP like NULL since ZEROP tests if a number is empty and NULL tests if a list is empty. |
| 4.12) | Is the way we wrote (GREATERP N1 N2) correct? | No; try it for the case where N1 and N2 are the same number. Let N1 and N2 be 3. |
| 4.13) | (ZEROP N2), where N1 is 3, and N2 is 3 | No, so move to the next question. |
| 4.14) | (ZEROP N1), where N1 is 3, and N2 is 3 | No, so move to the next question. |
| 4.15) | What is the meaning of the line (T (GREATERP (SUB1 N1)(SUB1 N2))) | Recurse, but with both the arguments reduced by one. |
| 4.16) | (ZEROP N2), where N1 is 2, and N2 is 2 | No, so move to the next question. |

4.17) (ZEROP N1), where
    N1 is 2, and
    N2 is 2

No,
so move to the next question.

---

4.18) What is the meaning of the line
    (T (GREATERP (SUB1 N1)(SUB1 N2)))

Recurse,
but with both arguments closer to zero by one.

---

4.19) (ZEROP N2), where
    N1 is 1, and
    N2 is 1

No,
so move to the next question.

---

4.20) (ZEROP N1), where
    N1 is 1, and
    N2 is 1

No,
so move to the next question.

---

4.21) What is the meaning of the line
    (T (GREATERP (SUB1 N1)(SUB1 N2)))

Recurse,
but with both arguments closer to zero by one.

---

4.22) (ZEROP N2), where
    N1 is 0, and
    N2 is 0

Yes,
so the value of (GREATERP N1 N2) is T.

---

4.23) Is this correct?

No,
because 3 is not greater than 3.

---

4.24) What can we do to the function (GREATERP N1 N2)
to take care of this subtle mistake?

Switch the ZEROP statements, that is:

```
(GREATERP (LAMBDA (N1 N2)
    (COND
    ((ZEROP N1) F)
    ((ZEROP N2) T)
    (T (GREATERP (SUB1 N1)(SUB1 N2)))
         ) ))
```

---

4.25) True or false: N1 and N2 are the same numeric
atoms where
    N1 is 1492 and
    N2 is 1492

True,
because both N1 and N2 are the numeric atom 1492.

---

4.26) Is (EQ N1 N2) true or false, where
    N1 is the argument 1492 and
    N2 is the argument 1492

No answer,
because EQ is only defined for literal atoms.
Althouth the definition of atom has been extended,
the function EQ has not been similarly extended.

---

4.27) Is (EQN N1 N2) true or false, where
    N1 is the argument 1492 and
    N2 is the argument 1492

True,
because (EQN N1 N2) is just another way to ask
if N1 and N2 are the same numeric atom.

---

4.28) Can you write the function (EQN N1 N2), using
(ZEROP N), (ADD1 N) and (SUB1 N)?
(EQN N1 N2), where
    N1 is 12 and
    N2 is 12 has the value T;
where
    N1 is 13 and
    N2 is 12 has the value F.

```
(EQN (LAMBDA (N1 N2)
    (COND
    ((ZEROP N2)(ZEROP N1))
    ((ZEROP N1) F)
    (T (EQN (SUB1 N1)(SUB1 N2)))
              ) ))
```

---

4.29) What is somewhat unusual about the line
    ((ZEROP N2)(ZEROP N1))

Here,
we ask if N2 is 0; if it is, then the value
of EQN is True if N1 is 0 and False if N1 is
not 0. In other words, we answered a question
with a question.

---

4.30) What is (DIFFERENCE N1 N2), where
    N1 is 8, and
    N2 is 3

5

---

4.31) What is (DIFFERENCE N1 N2), where
    N1 is 17, and
    N2 is 9

8

---

4.32) Try to write (DIFFERENCE N1 N2)
HINT: Use (SUB1 N).

How about this:

```
(DIFFERENCE (LAMBDA (N1 N2)
    (COND
    ((ZEROP N2) N1)
    (T (DIFFERENCE (SUB1 N1)(SUB1 N2)))
            ) ))
```

---

4.33) Can you describe in your own words how
(DIFFERENCE N1 N2)
does what it does?
Where
N1 is 8, and
N2 is 5

It takes two numbers as arguments, and reduces both of them by one, until the smaller --N2-- hits zero. When N2 hits zero, the value of the function is the other number --3.

---

4.34) What is (DIFFERENCE N1 N2), where
N1 is 3, and
N2 is 8

No answer,
because (SUB1 N1), where N1 is 0 is not a number.
That is, -1 is not a number, here.

---

4.35) What would be a good value for
(DIFFERENCE N1 N2),
where
N1 is 3, and
N2 is 8

There are a few alternatives:
1. 0, since it is the smallest number, or
2. 5, This is the alternative we choose, because it is the difference between N1 and N2.

---

4.36) How would you change DIFFERENCE to take care of the case where N2 is greater than N1?

By including another ZEROP line for M.

---

4.37) Rewrite (DIFFERENCE N1 N2) to take care of the case where N2 is greater than N1.

```
(DIFFERENCE (LAMBDA (N1 N2)
    (COND
    ((ZEROP N2) N1)
    ((ZEROP N1) N2)
    (T (DIFFERENCE (SUB1 N1)(SUB1 N2)))
            ) ))
```

---

4.38) What is (PLUS N1 N2), where
N1 is 46, and
N2 is 12

58

---

4.39) Try to write (PLUS N1 N2).
HINT: It uses ZEROP, ADD1, and SUB1.

```
(PLUS (LAMBDA (N1 N2)
    (COND
    ((ZEROP N1) N2)
    (T (PLUS (SUB1 N1)(ADD1 N2)))
            ) ))
```

Wasn't that easy?

---

4.40) Try to follow (PLUS N1 N2), where
N1 is 2, and
N2 is 5

Your answer should be 7.

---

4.41) Is this a VEC?
(1 2 8 X 4 3)

No,
it is just a list of atoms. Note: VEC stands for vector.

---

4.42) Is this a VEC?
(2 11 3 79 47 6)

Yes,
because it is a list of numbers.

---

4.43) Is this a VEC?
(3 (7 4) 13 9)

No,
because it is not a list of numbers -- (7 4) is not a number.

---

4.44) Is this a VEC?
( )

Yes,
it is a list of --zero-- numbers. This special case is the null VEC.

---

4.45) What is (ADDVEC VEC), where
VEC is (3 5 2 7)

17

---

4.46) What is (ADDVEC VEC), where
VEC is (15 6 7 12 3)

43

---

| | | |
|---|---|---|
| 4.47) | What does (ADDVEC VEC) do? | It builds a number by totaling all the numbers in the VEC. |
| 4.48) | What is the natural way to build numbers, just as CONS is the natural way to build lists? | By using PLUS. |
| 4.49) | When building lists with CONS, the value of the terminal condition is ( ). What should be the value of the terminal condition when building numbers? | 0 |
| 4.50) | What is the natural terminal condition for lists? | (NULL L) |
| 4.51) | What is the natural terminal condition for VECs? | (NULL VEC), because a VEC is also a list. |
| 4.52) | When you want to build a number from a list of numbers, what should the terminal condition line look like? | ((NULL VEC) 0), just as ((NULL L)( )) is the terminal condition for lists. |
| 4.53) | What is the terminal line of (ADDVEC VEC) | ((NULL VEC) 0) |
| 4.54) | What does CONS do? | CONS builds lists. |
| 4.55) | What does (ADDVEC VEC) do? | ADDVEC builds a number by totaling all the numbers in a VEC. |
| 4.56) | What does (ADDVEC VEC) use to build a number? | It uses PLUS, because PLUS builds numbers! |
| 4.57) | What will be the last line in the function:<br>(ADDVEC (LAMBDA (VEC)<br>(COND<br>((NULL VEC) 0)<br>) )) | (T (PLUS (CAR VEC)(ADDVEC (CDR VEC))))<br>Notice the similarity between this line, and the last line of the function (REMBER A LAT):<br>(T (CONS (CAR LAT)(REMBER A (CDR LAT)))) |
| 4.58) | What is (TIMES N1 N2), where N1 is 5, and N2 is 3 | 15 |
| 4.59) | What is (TIMES N1 N2), where N1 is 13, and N2 is 4 | 52 |
| 4.60) | What does (TIMES N1 N2) do? | It builds up a number by adding N1 up N2 times. |
| 4.61) | What is the terminal condition for (TIMES N1 N2) | ((ZEROP N2) 0) |
| 4.62) | Since (ZEROP N2) is the terminal condition, N2 must eventually be reduced to zero. What function is used to do this? | (SUB1 N2) |
| 4.63) | Try to write the function (TIMES N1 N2) | (TIMES (LAMBDA (N1 N2)<br>(COND<br>((ZEROP N2) 0)<br>(T (PLUS N1 (TIMES N1 (SUB1 N2))))<br>) )) |

| | | |
|---|---|---|
| 4.64) | What is (TIMES N1 N2), where<br>N1 is 12, and<br>N2 is 3 | 36,<br>but let's follow through the function one time to see how we get this value. |
| 4.65) | (ZEROP N2) | No. |
| 4.66) | What is the meaning of the line<br>(T (PLUS N1 (TIMES N1 (SUB1 N2)))) | It adds N1 --12---to the natural recursion:<br>(TIMES N1 (SUB1 N2)) |
| 4.67) | What are the new arguments of<br>(TIMES N1 N2) | N1 is 12, and<br>N2 is 2. |
| 4.68) | (ZEROP N2) | No. |
| 4.69) | What is the meaning of the line<br>(T (PLUS N1 (TIMES N1 (SUB1 N2)))) | It adds N1 --12-- to (TIMES N1 (SUB1 N2)) |
| 4.70) | What are the new arguments of<br>(TIMES N1 N2) | N1 is 12, and<br>N2 is 1. |
| 4.71) | (ZEROP N2) | No. |
| 4.72) | What is the meaning of the line<br>(T (PLUS N1 (TIMES N1 (SUB1 N2)))) | It adds N1 --12-- to (TIMES N1 (SUB1 N2)) |
| 4.73) | What is the value of the line<br>((ZEROP N2) 0) | 0,<br>because (ZEROP N2) is now true. |
| 4.74) | Are we finished now? | No. |
| 4.75) | Why not? | Because we still have three PLUS's to pick up. |
| 4.76) | What is the value of the function? | PLUS 12 to PLUS 12 to PLUS 12 to 0 --36.<br>Notice that N1 has been PLUS'd N2 times. |
| 4.77) | What is (QUOTIENT N1 N2), where<br>N1 is 12, and<br>N2 is 4 | 3 |
| 4.78) | What is (REMAINDER N1 N2), where<br>N1 is 12, and<br>N2 is 4 | 0 |
| 4.79) | What is (QUOTIENT N1 N2), where<br>N1 is 7, and<br>N2 is 3 | 2 |
| 4.80) | What is (REMAINDER N1 N2), where<br>N1 is 7, and<br>N2 is 3 | 1 |
| 4.81) | What is (QUOTIENT N1 N2), where<br>N1 is 0, and<br>N2 is 4 | 0 |
| 4.82) | What is (REMAINDER N1 N2), where<br>N1 is 0, and<br>N2 is 4 | 0 |
| 4.83) | What is (QUOTIENT N1 N2), where<br>N1 is 3, and<br>N2 is 7 | 0 |
| 4.84) | What is (REMAINDER N1 N2), where<br>N1 is 3, and<br>N2 is 7 | 3 |
| 4.85) | Try to write the function<br>(QUOTIENT N1 N2) | How about |

```
(QUOTIENT (LAMBDA (N1 N2)
    (COND
    ((ZEROP N1) 0)
    (T (ADD1 (QUOTIENT (DIFFERENCE N1 N2) N2)))
              )  ))
```

| | | |
|---|---|---|
| 4.86) | Does this work, where<br>N1 is 5, and<br>N2 is 2 | Let's see. |
| 4.87) | (ZEROP N1) | No,<br>so move to the next line. |
| 4.88) | What is the meaning of the line<br>(T (ADD1 (QUOTIENT (DIFFERENCE N1 N2) N2))) | We are adding one to the natural recursion:<br>(QUOTIENT (DIFFERENCE N1 N2) N2) |
| 4.89) | What are the arguments of (QUOTIENT N1 N2), after we recursed once? | N1 is 3 --(DIFFERENCE N1 N2)--, and<br>N2 is 2. |
| 4.90) | (ZEROP N1) | No,<br>so move to the next line. |
| 4.91) | (T (ADD1 (QUOTIENT (DIFFERENCE N1 N2) N2))) | This again adds one to the natural recursion. |
| 4.92) | What are the arguments of (QUOTIENT N1 N2), now? | N1 is 1, and<br>N2 is 2. |
| 4.93) | (ZEROP N1) | No,<br>so move to the next line. |
| 4.94) | (T (ADD1 (QUOTIENT (DIFFERENCE N1 N2) N2))) | This again adds one to the recursion. |
| 4.95) | What are the arguments of (QUOTIENT N1 N2), now? | N1 is 1 --(DIFFERENCE N1 N2)--, and<br>N2 is 2. |
| 4.96) | Is this what we wanted? | No.<br>Obviously, this will never reach the terminal condition. |
| 4.97) | Do we need a different terminal condition? | Yes. |
| 4.98) | When do we want to stop adding one to the value? | When N2 is greater than N1. |
| 4.99) | So what is the correct terminal condition? | (GREATERP N2 N1) |
| 4.100) | What should be the value given when the terminal condition is true? | 0,<br>because 0 will not affect ADD1. |

COMMANDMENT No. 4

<ins>When building a value with ADD1 or PLUS, always use 0 for the value of the terminating line; when building a value with TIMES, always use 1 for the value of the terminating line, because 0 and 1 do not affect the final value of the function. When building a value with CONS, always consider first ( ) for the value of the terminating line</ins>.

When using ADD1, the terminal line value is 0.
When using PLUS, the terminal line value is 0.
When using TIMES, the terminal line value is 1.
When using CONS, the terminal line value is usually ( ).

| | | |
|---|---|---|
| 4.101) | What is (VECPLUS VEC1 VEC2), where<br>VEC1 is (3 6 9 11 4), and<br>VEC2 is (8 5 2 0 7) | (11 11 11 11 11) |

| | | |
|---|---|---|
| 4.102) | What is (VECPLUS VEC1 VEC2), where<br>VEC1 is (2 3), and<br>VEC2 is (4 6) | (6 9) |
| 4.103) | What does (VECPLUS VEC1 VEC2) do? | It adds the first number of VEC1 to the first number in VEC2, then it adds the second number in VEC1 to the second number in VEC2, and so on, for the VECs of the same length. |
| 4.104) | Can you write (VECPLUS VEC1 VEC2) ? | `(VECPLUS (LAMBDA (VEC1 VEC2)`<br>`(COND`<br>`((NULL VEC1)( ))`<br>`(T (CONS (PLUS (CAR VEC1)(CAR VEC2))(VECPLUS`<br>`(CDR VEC1)(CDR VEC2))))`<br>`) ))` |
| 4.105) | What are the arguments of PLUS in the last line? | (CAR VEC1) and (CAR VEC2) |
| 4.106) | What are the arguments of CONS in the last line? | (PLUS (CAR VEC1)(CAR VEC2)), and<br>(VECPLUS (CDR VEC1)(CDR VEC2)). |
| 4.107) | What is (VECPLUS VEC1 VEC2), where<br>VEC1 is (3 7), and<br>VEC2 is (4 6) | (7 13),<br>but let's see just how it works. |
| 4.108) | (NULL VEC1) | No. |
| 4.109) | (T (CONS (PLUS (CAR VEC1)(CAR VEC2))(VECPLUS<br>(CDR VEC1)(CDR VEC2)))) | CONS 7 onto the natural recursion:<br>(VECPLUS (CDR VEC1)(CDR VEC2)) |
| 4.110) | Why does the natural recursion include the CDR of both arguments? | Because the typical element of the final value uses the CAR of both VECs, so now we are ready to consider the rest of both VECs. |
| 4.111) | (NULL VEC1), where<br>VEC1 is now (7), and<br>VEC2 is now (6) | No. |
| 4.112) | (T (CONS (PLUS (CAR VEC1)(CAR VEC2))(VECPLUS<br>(CDR VEC1)(CDR VEC2)))) | CONS 13 onto the natural recursion. |
| 4.113) | (NULL VEC1) | Yes. |
| 4.114) | What is the value of the line? | ( ) |
| 4.115) | What is the value of the function? | (7 13),<br>by CONSing 7 onto the CONS of 13 onto ( ). |
| 4.116) | What problem arises when we want to do<br>(VECPLUS VEC1 VEC2), where<br>VEC1 is (3 7), and<br>VEC2 is (4 6 8 1)<br>HINT: Try going through the function with these arguments. | When VEC1 eventually gets to be ( ), we quit, but that means the final value will be (7 13), which is wrong. The final value should be (7 13 8 1) |
| 4.117) | What trivial change can you make in the terminal condition line to get the correct final value? | Change ((NULL VEC1)( )) to<br>((NULL VEC1) VEC2) |
| 4.118) | What is (VECPLUS VEC1 VEC2), where<br>VEC1 is (3 7 8 1), and<br>VEC2 is (4 6) | No answer,<br>since VEC2 will become null before VEC1.<br>See PRINCIPLES No. 1 and No. 2. |
| 4.119) | What do we need to include in our function<br>(VECPLUS VEC1 VEC2) | Another terminal condition. |
| 4.120) | What is the other terminal line? | ((NULL VEC2) VEC1) |

4.121) So now that we have expanded our function definition so that VECPLUS works for any two VECs, see if you can rewrite it.

```
(VECPLUS (LAMBDA (VEC1 VEC2)
    (COND
    ((NULL VEC1) VEC2)
    ((NULL VEC2) VEC1)
    (T (CONS (PLUS (CAR VEC1)(CAR VEC2))(VECPLUS
              (CDR VEC1)(CDR VEC2))))
                ) ))
```

---

4.122) Does the order of the two terminal conditions matter?

No.

---

4.123) What is (MAXVEC VEC), where
VEC is (4 8 2 13 12)

13

---

4.124) What is (MAXVEC VEC), where
VEC is (468 942 8 27 43)

942

---

4.125) What is (MAXVEC VEC), where
VEC is (4)

4,
get the idea?

---

4.126) Can you write the function (MAXVEC VEC), which has as its final value the greatest number in the VEC?

```
(MAXVEC (LAMBDA (VEC)
    (COND
    ((NULL (CDR VEC))(CAR VEC))
    ((GREATERP (CAR VEC)(MAXVEC (CDR VEC)))(CAR VEC))
    (T (MAXVEC (CDR VEC)))
         ) ))
```

This is the way we would write the function, but it also could have been written as follows:

```
(MAXVEC (LAMBDA (VEC)
    (COND
    ((NULL (CDR VEC))(CAR VEC))
    ((GREATERP (CAR VEC)(CAR (CDR VEC)))(MAXVEC
        (CONS (CAR VEC)(CDR (CDR VEC)))))
    (T (MAXVEC (CDR VEC)))
         ) ))
```

Try to follow through both ways until you are satisfied that you understand both of them. Use whichever makes more sense to you.

---

4.127) In the first (MAXVEC VEC) that we wrote, what is the meaning of the line
((GREATERP (CAR VEC)(MAXVEC (CDR VEC)))(CAR VEC))

((GREATERP (CAR VEC)(MAXVEC (CDR VEC)))(CAR VEC)) asks if the first number in the VEC is greater than the biggest number in the rest of the VEC. If it is, then the first number is the one we want.

---

4.128) What is the meaning of the line
(T (MAXVEC (CDR VEC)))

Since we know by now that the first number is not the greatest, then we throw it away, and start all over again.

---

4.129) In the second (MAXVEC VEC) that we wrote, what is the meaning of the line
((GREATERP (CAR VEC)(CAR (CDR VEC)))(MAXVEC
(CONS (CAR VEC)(CDR (CDR VEC)))))

It means we ask if the first number is greater than the second number. If it is, we recurse after throwing away the second number.

---

4.130) How did we throw away the second number?

(CONS (CAR VEC)(CDR (CDR VEC)))

---

4.131) What would happen if we tried to do this where
VEC is (6)

No answer.
See PRINCIPLE No. 2.

---

4.132) How did we keep this from ever happening?

The terminal condition was
(NULL (CDR VEC))
instead of
(NULL VEC)

---

4.133) Go through (MAXVEC VEC) both ways, where
VEC is (4 9 13 2 13 12)
What is the answer?

13

---

4.134) In the second (MAXVEC VEC), which 13 is the final value?

(4 9 13 2 <u>13</u> 12)

---

4.135) In the first (MAXVEC VEC), which 13 is the final value?

(4 9 13 2 <u>13</u> 12)

---

4.136) Now write (REMAINDER N1 N2)

```
(REMAINDER (LAMBDA (N1 N2)
    (COND
    ((GREATERP N2 N1) N1)
    (T (REMAINDER (DIFFERENCE N1 N2) N2))
            )  ))
```

---

4.137) What is (LESSP N1 N2), where
N1 is 4, and
N2 is 6

T

---

4.138) (LESSP N1 N2), where
N1 is 8, and
N2 is 3

F

---

4.139) (LESSP N1 N2), where
N1 is 6, and
N2 is 6

F

---

4.140) Now try to write (LESSP N1 N2)

```
(LESSP (LAMBDA (N1 N2)
    (COND
    ((ZEROP N2) F)
    ((ZEROP N1) T)
    (T (LESSP (SUB1 N1)(SUB1 N2)))
          )  ))
```

---

4.141) (EXPT N1 N2), where
N1 is 1, and
N2 is 1

1

---

4.142) (EXPT N1 N2), where
N1 is 2, and
N2 is 3

8

---

4.143) (EXPT N1 N2), where
N1 is 5, and
N2 is 3

125

---

4.144) Now write (EXPT N1 N2)
HINT: See COMMANDMENT No. 4.

```
(EXPT (LAMBDA (N1 N2)
    (COND
    ((ZEROP N2) 1)
    (T (TIMES N1 (EXPT N1 (SUB1 N2))))
            )  ))
```

---

4.145) What is the value of (LENGTH LAT), where
LAT is (HOTDOGS WITH MUSTARD SAUERKRAUT
AND PICKLES)

6

---

4.146) What is (LENGTH LAT), where
LAT is (HAM AND CHEESE ON RYE)

5

---

4.147) Now try to write (LENGTH LAT)

```
(LENGTH (LAMBDA (LAT)
    (COND
    ((NULL LAT) 0)
    (T (ADD1 (LENGTH (CDR LAT))))
          )  ))
```

---

4.148) What is (PICK N LAT), where
N is 4, and
LAT is (LASAGNA SPAGHETTI RAVIOLI MACARONI MEATBALL)

MACARONI

---

4.149) What is (PICK N LAT), where
N is 0, and
LAT is ( )

No answer.

---

4.150) Try to write (PICK N LAT)

```
(PICK (LAMBDA (N LAT)
    (COND
    ((ZEROP (SUB1 N))(CAR LAT))
    (T (PICK (SUB1 N)(CDR LAT)))
           )  ))
```

---

Wouldn't a HAM AND CHEESE ON RYE be good right now?

Don't forget the mustard.

4.151) What is (REMPICK N LAT), where
N is 3, and
LAT is (HOTDOGS WITH HOT MUSTARD)

(HOTDOGS WITH MUSTARD)

---

4.152) What is (REMPICK N LAT), where
N is 0, and
LAT is ( )

No answer.

---

4.153) Now try to write (REMPICK N LAT)

```
(REMPICK (LAMBDA (N LAT)
    (COND
    ((ZEROP (SUB1 N))(CDR LAT))
    (T (CONS (CAR LAT)(REMPICK (SUB1 N)(CDR LAT))))
               )  ))
```

---

4.154) Is (NUMBERP AT) true, or false, where
AT is TOMATO

F.

---

4.155) Is (NUMBERP AT) true, or false, where
AT is 76

T.

---

4.156) Can you write (NUMBERP A) which is True if A is a numeric atom and False if A is a literal atom?

No,
this like ADD1, SUB1, ZEROP, CAR, CDR, CONS, NULL
EQ and ATOM is a primitive (built-in) function.

---

4.157) Now using (NUMBERP AT), write a function (MAKALAT L), which gives a LAT as a final value, with all the numbers removed.
For example, where
L is (5 PEARS 6 PRUNES 9 DATES)
the final value is (PEARS PRUNES DATES).

```
(MAKALAT (LAMBDA (L)
    (COND
    ((NULL L)( ))
    ((NUMBERP (CAR L))(MAKALAT (CDR L)))
    (T (CONS (CAR L)(MAKALAT (CDR L))))
                 )  ))
```

---

4.158) Now write (MAKAVEC L), which gives a VEC as a final value.

```
(MAKAVEC (LAMBDA (L)
    (COND
    ((NULL L)( ))
    ((NUMBERP (CAR L))(CONS (CAR L)(MAKAVEC (CDR L))))
    (T (MAKAVEC (CDR L)))
          )  ))
```

---

## CHAPTER 5: THE MULTICHAPTER CHAPTER

5.1) Write the function
(MEMBER A LAT)

```
(MEMBER (LAMBDA (A LAT)
    (COND
    ((NULL LAT) F)
    ((EQ (CAR LAT) A) T)
    (T (MEMBER A (CDR LAT)))
          ) ))
```

---

5.2) Do you recall, or can you see now, what MEMBER does.

(MEMBER A LAT) checks each atom of the LAT to see if each is the same as the atom A. When it finds the first occurrence of A, it stops and gives a value of T, since A is a member of the LAT.

---

5.3) Write the function
(REMBER A LAT)

```
(REMBER (LAMBDA (A LAT)
    (COND
    ((NULL LAT)( ))
    ((EQ (CAR LAT) A)(CDR LAT))
    (T (CONS (CAR LAT)(REMBER A (CDR LAT))))
                 ) ))
```

---

5.4) Do you recall, or can you see now, what REMBER does?

(REMBER A LAT) looks at each atom of the LAT to see if it is equal to the atom A. If it is not, REMBER saves the atom and proceeds. When it finds the first occurrence of A, it stops and gives the value (CDR LAT), or the rest of the LAT, so that the final value is the same as the original LAT, but with only the first occurrence of A missing.

---

5.5) Can you write a function called (MULTIREMBER A LAT), which gives as its final value the LAT with <u>all</u> occurrences of A removed?
HINT: What do we want as the value when
(EQ (CAR LAT) A) is true?
Consider the example where
A is CUP, and
LAT is (COFFEE CUP TEA CUP AND HICK CUP)

```
(MULTIREMBER (LAMBDA (A LAT)
    (COND
    ((NULL LAT)( ))
    ((EQ (CAR LAT) A)(MULTIREMBER A (CDR LAT)))
    (T (CONS (CAR LAT)(MULTIREMBER A (CDR LAT))))
                ) ))
```

Notice that after the first occurrence of A, we now recurse with (MULTIREMBER A (CDR LAT)), in order to remove the other possible occurrences. The value for the function is (COFFEE TEA AND HICK).

---

5.6) Can you see how MULTIREMBER does what it does?

Possibly not,
so we will go through the steps necessary to arrive at the value (COFFEE TEA AND HICK).

---

5.7) (NULL LAT)

No,
so move to the next line.

---

5.8) (EQ (CAR LAT) A)

No,
so move to the next line.

---

5.9) What is the meaning of the whole line
(T (CONS (CAR LAT)(MULTIREMBER A (CDR LAT))))

Save (CAR LAT) to be CONS'd onto the value of (MULTIREMBER A (CDR LAT)) later. Now find (MULTIREMBER A (CDR LAT)).

---

5.10) (NULL LAT)

No, so move to the next line.

---

5.11) (EQ (CAR LAT) A)

Yes,
so forget (CAR LAT), and find (MULTIREMBER A (CDR LAT)).

---

5.12) (NULL LAT)

No,
so move to the next line.

---

5.13) (EQ (CAR LAT) A)

No,
so move to the next line.

---

| | | |
|---|---|---|
| 5.14) | What is the meaning of the line<br>(T (CONS (CAR LAT)(MULTIREMBER A (CDR LAT)))) | Save (CAR LAT) -TEA- to be CONS'd onto the value of (MULTIREMBER A (CDR LAT)) later. Now find (MULTIREMBER A (CDR LAT)). |
| 5.15) | (NULL LAT) | No,<br>so move to the next line. |
| 5.16) | (EQ (CAR LAT) A) | Yes,<br>so forget (CAR LAT), and find (MULTIREMBER A (CDR LAT)). |
| 5.17) | (NULL LAT) | No,<br>so move to the next line. |
| 5.18) | (EQ (CAR LAT) A) | No,<br>so move to the next line. |
| 5.19) | What is the meaning of the line<br>(T (CONS (CAR LAT)(MULTIREMBER A (CDR LAT)))) | Save (CAR LAT) --AND-- to be CONS'd onto the value of (MULTIREMBER A (CDR LAT)) later. Now find (MULTIREMBER A (CDR LAT)). |
| 5.20) | (NULL LAT) | No,<br>so move to the next line. |
| 5.21) | (EQ (CAR LAT) A) | No,<br>so move to the next line. |
| 5.22) | What is the meaning of the line<br>(T (CONS (CAR LAT)(MULTIREMBER A (CDR LAT)))) | Save (CAR LAT) --HICK-- to be CONS'd onto the value of (MULTIREMBER A (CDR LAT)) later. Now find (MULTIREMBER A (CDR LAT)). |
| 5.23) | (NULL LAT) | No,<br>so move to the next line. |
| 5.24) | (EQ (CAR LAT) A) | Yes,<br>so forget (CAR LAT), and find (MULTIREMBER A (CDR LAT)). |
| 5.25) | (NULL LAT) | Yes,<br>so we have a value of ( ). |
| 5.26) | Are we finished? | No,<br>since we still have several saved up CONS operations to pick up. |
| 5.27) | What do we do next? | CONS the most recent (CAR LAT) we have --HICK-- onto ( ). |
| 5.28) | What do we do next? | CONS AND onto (HICK). |
| 5.29) | What do we do next? | CONS TEA onto (AND HICK). |
| 5.30) | What do we do next? | CONS COFFEE onto (TEA AND HICK). |
| 5.31) | Are we finished now? | Yes. |

5.32) Now can you write a function
(MULTIINSERTR OLD NEW LAT)
Try it.

```
(MULTIINSERTR (LAMBDA (OLD NEW LAT)
    (COND
    ((NULL LAT)( ))
    ((EQ (CAR LAT) OLD)(CONS (CAR LAT)(CONS NEW
       (MULTIINSERTR OLD NEW (CDR LAT)))))
    (T (CONS (CAR LAT)(MULTIINSERTR OLD NEW (CDR LAT))))
                   ) ))
```

Note: It would also be correct to use OLD in place of the underlined (CAR LAT).

5.33) Now try to write the function (MULTIINSERTL OLD NEW LAT)

```
(MULTIINSERTL (LAMBDA (OLD NEW LAT)
    (COND
    ((NULL LAT)( ))
    ((EQ (CAR LAT) OLD)(CONS NEW (CONS (CAR LAT)
        (MULTIINSERTL OLD NEW (CDR LAT)))))
    (T (CONS (CAR LAT)(MULTIINSERTL OLD NEW (CDR LAT))))
          )  ))
```

Note: It would also be correct to use OLD in place of the underlined (CAR LAT).

---

5.34) Is this function defined correctly?

```
(MULTIINSERTL (LAMBDA (OLD NEW LAT)
    (COND
    ((NULL LAT)( ))
    ((EQ (CAR LAT) OLD)(CONS NEW (MULTIINSERTL
        OLD NEW LAT)))
    (T (CONS (CAR LAT)(MULTIINSERTL OLD NEW (CDR LAT))))
          )  ))
```

Not quite.
To find out why, go through the function, where
OLD is FISH
NEW is FRIED, and
LAT is (FISH AND CHIPS OR FISH AND FRIES)

---

5.35) Was the terminal condition ever reached?

No,
because you never get past the first occurrence of OLD.

---

**COMMANDMENT No. 5**

Thou shalt endeavor to change at least one argument while recursing. The changing argument should be the one tested in the termination condition(s) and it should be changed to be closer to termination, e.g.,

CDR with NULL and
SUB1 with ZEROP

but not

CONS with NULL
ADD1 with ZEROP

---

5.36) Now write (MULTISUBST OLD NEW LAT)

```
(MULTISUBST (LAMBDA (OLD NEW LAT)
    (COND
    ((NULL LAT)( ))
    ((EQ (CAR LAT) OLD)(CONS NEW (MULTISUBST
        OLD NEW (CDR LAT))))
    (T (CONS (CAR LAT)(MULTISUBST OLD NEW (CDR LAT))))
                )  ))
```

---

5.37) Now write a function (OCCUR A LAT), which counts the number of times the atom A occurs in LAT.

```
(OCCUR (LAMBDA (A LAT)
    (COND
    ((NULL LAT) 0)
    ((EQ (CAR LAT) A)(ADD1 (OCCUR A (CDR LAT))))
    (T (OCCUR A (CDR LAT)))
          )  ))
```

---

5.38) Write the function (ONEP N) where the function is True if N is 1

```
(ONEP (LAMBDA (N)
    (COND
    ((ZEROP N) F)
    (T (ZEROP (SUB1 N)))
                )  ))
```

or

```
(ONEP (LAMBDA (N)
    (COND
    (T (EQN N 1))
          )  ))
```

---

5.39) The second definition of ONEP is called a "one-liner." Is there a simple way to write "one-liners?" Try to guess what changes occur in the simplification of ONEP.

By removing

```
    (COND
    (T          )
```

we get

```
(ONEP (LAMBDA (N)
    (EQN N 1)
          ))
```

---

5.40) Now write a function
    (REMPICK N LAT), using (ONEP N)
which removes the Nth atom of the Lat.
For example, where
    N is 3, and
    LAT is (LEMON MERINGUE SALTY PIE),
the function has the value
    (LEMON MERINGUE PIE).

```
(REMPICK (LAMBDA (N LAT)
    (COND
    ((ONEP N)(CDR LAT))
    (T (CONS (CAR LAT)(REMPICK (SUB1 N)(CDR LAT))))
          ) ))
```

---

5.41) Is REMPICK a "multi-" function?

No.

---

| | | |
|---|---|---|
| 6.1) | True or false:<br>(NOT (ATOM S)), where<br>S is (HUNGARIAN GOULASH) | T. |
| 6.2) | (NOT (ATOM S)), where<br>S is ATOM | F. |
| 6.3) | (NOT (ATOM S)), where<br>S is (TURKISH ((COFFEE) AND) BAKLAVA) | T.<br>Get the idea? |
| 6.4) | What is (LEFTMOSTAT L), where<br>L is ((HOT)(TUNA (AND)) CHEESE) | HOT |
| 6.5) | (ISLAT L), where<br>L is ((HOT)(TUNA (AND)) CHEESE) | F. |
| 6.6) | Is (CAR L) an atom, where<br>L is ((HOT)(TUNA (AND)) CHEESE) | No. |
| 6.7) | What is (LEFTMOSTAT L), where<br>L is (((HAMBURGER) FRENCH)(FRIES (AND A) COKE)) | HAMBURGER |
| 6.8) | What is (LEFTMOSTAT L), where<br>L is (((4) FOUR) 17 (SEVENTEEN)) | 4 |

6.9) Now see if you can write the function definition for (LEFTMOSTAT L)

```
(LEFTMOSTAT (LAMBDA (L)
    (COND
    ((NULL L)( ))
    ((NOT (ATOM (CAR L)))(LEFTMOSTAT (CAR L)))
    (T (CAR L))
        ) ))
```

Use one of our previous examples as arguments, and try to follow through this to see how it works. Notice that now we are recursing down the CAR of the list, instead of the CDR of the list.

6.10) What is (REMBER* A L), where
A is CUP, and
L is ((COFFEE) CUP ((TEA) CUP)(AND (HICK)) CUP)
Note: "REMBER*" is pronounced "REMBER-STAR".

((COFFEE)((TEA))(AND (HICK)))

6.11) What is (REMBER* A L), where
A is SAUCE, and
L is (((TOMATO SAUCE))((BEAN) SAUCE)(AND ((FLYING)) SAUCE))

(((TOMATO))((BEAN))(AND ((FLYING))))

6.12) Now write (REMBER* A L)
Note: "*" means "OH MY GAWD".

```
(REMBER* (LAMBDA (A L)
    (COND
    ((NULL L)( ))
    ((NOT (ATOM (CAR L)))(CONS (REMBER* A (CAR L))
        (REMBER* A (CDR L))))
    ((EQ (CAR L) A)(REMBER* A (CDR L)))
    (T (CONS (CAR L)(REMBER* A (CDR L))))
        ) ))
```

6.13) (INSERTR* OLD NEW L), where
OLD is CHUCK
NEW is ROAST, and
L is (HOW (MUCH WOOD) COULD ((A WOOD CHUCK) (CHUCK)(IF (A)(WOOD CHUCK) COULD ((CHUCK) WOOD))))

(HOW (MUCH WOOD) COULD ((A WOOD CHUCK ROAST) (CHUCK ROAST)(IF (A) (WOOD CHUCK ROAST) COULD ((CHUCK ROAST) WOOD))))

6.14) Now write a function (INSERTR* OLD NEW L), which inserts the atom NEW to the right of OLD, regardless of where OLD occurs.

```
(INSERTR* (LAMBDA (OLD NEW L)
    (COND
    ((NULL L)( ))
    ((NOT (ATOM (CAR L)))(CONS (INSERTR* OLD NEW
        (CAR L))(INSERTR* OLD NEW (CDR L))))
    ((EQ (CAR L) OLD)(CONS OLD (CONS NEW (INSERTR*
        OLD NEW (CDR L)))))
    (T (CONS (CAR L)(INSERTR* OLD NEW (CDR L))))
        ) ))
```

6.15) How are (INSERTR* OLD NEW L), and (LEFTMOSTAT L) similar?

Both functions recurse with the CAR of their list arguments.

---

6.16) How does (REMBER* A L) differ from (MULTIREMBER A LAT) ?

REMBER* recurses with the CAR as well as with the CDR. It does not recurse with the CAR, however, until it finds out that the CAR is not an atom.

---

6.17) How are (INSERTR* OLD NEW L), and (REMBER* A L) similar?

They both recur with the CAR as well as with the CDR, whenever the CAR is a list.

---

6.18) How are <u>all</u> *-functions --pronounced "star-functions"-- similar?

They <u>all</u> recurse with the CAR as well as with the CDR, whenever the CAR is a list.

---

COMMANDMENT No. 6

> <u>Thou shalt always recurse with the CAR as well as with the CDR when writing *-functions.</u>

6.19) (OCCURSOMETHING A L), where
A is BANANA, and
L is ((BANANA)(SPLIT ((((BANANA ICE)))(CREAM (BANANA)) SHERBERT))(BANANA)(BREAD) (BANANA BRANDY))

5

---

6.20) What is a better function name for (OCCURSOMETHING A L)

(OCCUR* A L)

---

6.21) Write (OCCUR* A L)

```
(OCCUR* (LAMBDA (A L)
    (COND
    ((NULL L) 0)
    ((NOT (ATOM (CAR L)))(PLUS (OCCUR* A (CAR L))
        (OCCUR* A (CDR L))))
    ((EQ (CAR L) A)(ADD1 (OCCUR* A (CDR L))))
    (T (OCCUR* A (CDR L)))
       ) ))
```

---

6.22) (SUBST* OLD NEW L), **where**
OLD is BANANA
NEW is ORANGE, and
L is ((BANANA)(SPLIT ((((BANANA ICE)))(CREAM (BANANA)) SHERBERT))(BANANA)(BREAD) (BANANA BRANDY))

((ORANGE)(SPLIT ((((ORANGE ICE)))(CREAM (ORANGE)) SHERBERT))(ORANGE)(BREAD) (ORANGE BRANDY))

---

6.23) Write (SUBST* OLD NEW L)

```
(SUBST* (LAMBDA (OLD NEW L)
    (COND
    ((NULL L)( ))
    ((NOT (ATOM (CAR L)))(CONS (SUBST* OLD NEW (CAR L))
        (SUBST* OLD NEW (CDR L))))
    ((EQ (CAR L) OLD)(CONS NEW (SUBST* OLD NEW
        (CDR L))))
    (T (CONS (CAR L)(SUBST* OLD NEW (CDR L))))
        ) ))
```

---

6.24) (INSERTL* OLD NEW L), where
OLD is FRIED
NEW is FRENCH
L is (FRIED ((POTATOES ((FRIED))(FISH))((FRIED))) EGG)

(FRENCH FRIED ((POTATOES ((FRENCH FRIED))(FISH)) ((FRENCH FRIED))) EGG)

---

6.25) Write (INSERTL* OLD NEW L)

```
(INSERTL* (LAMBDA (OLD NEW L)
    (COND
    ((NULL L)( ))
    ((NOT (ATOM (CAR L)))(CONS (INSERTL* OLD NEW
        (CAR L))(INSERTL* OLD NEW (CDR L))))
    ((EQ (CAR L) OLD)(CONS NEW (CONS OLD (INSERTL*
        OLD NEW (CDR L)))))
    (T (CONS (CAR L)(INSERTL* OLD NEW (CDR L))))
            ) ))
```

---

6.26) (AND (ATOM (CAR L))(EQ (CAR L) X)), where
X is PIZZA, and
L is (MOZARELLA PIZZA)

F

---

6.27) Why is it false?

Since AND asks (ATOM (CAR L)), and it is, so then it asks (EQ (CAR L) X), and it is not; hence AND has the value F.

---

6.28) What is (AND (ATOM (CAR L))(EQ (CAR L) X)), where
X is PIZZA, AND
L is ((MOZARELLA MUSHROOM) PIZZA)

F

---

6.29) Why is it false?

Since AND asks (ATOM (CAR L)), and it is not; so AND has the value F.

---

6.30) Give an example for X and L, where AND is true.

How about, where
X is PIZZA, and
L is (PIZZA (TASTES GOOD))

---

6.31) Can you put in your own words what the function AND does?

The function AND is either true or false. AND asks questions one at a time until it finds an argument the value of which is false. Then AND stops, making its value false. If it cannot find one false argument, then the value of AND is true.

---

6.32) (OR (NOT (ATOM (CAR L)))(EQ (CAR L) X)), where
X is CURRIED, and
L is (CURRIED (SHRIMP WITH RICE))

T.

---

6.33) Why is it true?

(NOT (ATOM (CAR L))) is F, so we ask (EQ (CAR L) X), which is T, so the value of OR is true.

---

6.34) (OR (NOT (ATOM (CAR L)))(EQ (CAR L) X)), where
X is CURRIED, and
L is ((CURRIED) SHRIMP WITH RICE)

T.

---

6.35) Why is it true?

(NOT (ATOM (CAR L))) is T, so the value of OR is true.

---

6.36) (OR (NOT (ATOM (CAR L)))(EQ (CAR L) X)), where
X is CURRIED and
L is (SHRIMP WITH CURRIED RICE)

OR asks (NOT (ATOM (CAR L))), which is false, so OR asks (EQ (CAR L) X), which is also false, so the value of OR is false.

---

6.37) Can you put in your own words what the function OR does?

The function OR is either true or false. OR asks questions one at a time until it finds an argument the value of which is true. Then OR stops, making its value true. If it cannot find one true argument, then the value of OR is false.

---

6.38) True or false: It is possible that one of the arguments of AND and OR is not considered?

T,†
because AND stops if the first argument has the value F, and OR stops if the first argument has the value T.

---

6.39) (MEMBER* A L), where
A is GRAPE, and
L is ((GRAPEFRUIT (RAISINS))(WINE (AND) GRAPES))

F,
since the atom GRAPE does not appear in the list L.

---

†COND _is_ a function; it also has this property.

6.40) (MEMBER* A L), where
A is CHIPS, and
L is ((POTATO)(CHIPS ((WITH) FISH)(CHIPS)))

T,
because the atom CHIPS appears in the list L.

---

6.41) Write (MEMBER* A L).

```
(MEMBER* (LAMBDA (A L)
    (COND
    ((NULL L) F)
    ((NOT (ATOM (CAR L)))(OR (MEMBER* A (CAR L))
        (MEMBER* A (CDR L))))
    ((EQ (CAR L) A) T)
    (T (MEMBER* A (CDR L)))
        ) ))
```

---

6.42) What is (MEMBER* A L), where
A is CHIPS, and
L is ((POTATO)(CHIPS ((WITH) FISH)(CHIPS)))

T.

---

6.43) Which CHIPS did it find?

((POTATO)(<u>CHIPS</u> ((WITH) FISH)(CHIPS)))

---

6.44) Try to combine the last two lines of (MEMBER* A L) into one line. Rewrite the function using this line.

```
(MEMBER2* (LAMBDA (A L)
    (COND
    ((NULL L) F)
    ((NOT (ATOM (CAR L))) (OR (MEMBER2* A (CAR L))
        (MEMBER2* A (CDR L))))
    (T (OR (EQ (CAR L) A)(MEMBER2* A (CDR L))))
        ) ))
```

---

6.45) Try to write (MEMBER2* A L) without using <u>NOT</u>.

```
(MEMBER3* (LAMBDA (A L)
    (COND
    ((NULL L) F)
    ((ATOM (CAR L))(OR (EQ (CAR L) A)
        (MEMBER3* A (CDR L))))
    (T (OR (MEMBER3* A (CAR L))(MEMBER3* A (CDR L))))
        ) ))
```

---

6.46) Now, can you break up the last line of (MEMBER3* A L) to form two lines.
HINT: See (MEMBER* A L)

```
(MEMBER4* (LAMBDA (A L)
    (COND
    ((NULL L) F)
    ((ATOM (CAR L))(OR (EQ (CAR L) A)
        (MEMBER4* A (CDR L))))
    ((MEMBER4* A (CAR L)) T)
    (T (MEMBER4* A (CDR L)))
        ) ))
```

---

6.47) What is unusual about (MEMBER4* A L)

We are recursing in the question side of a line.

---

6.48) Have we ever seen recursion on the left side of a line before (MEMBER4* A L)

Yes,
see (MAXVEC VEC).

---

6.49) Why does (EQ N1 N2), where
N1 is 15, and
N2 is 12
have no answer?

Since EQ is only defined for atoms beginning with letters.

---

6.50) Why does (EQN A1 A2) where
A1 is HARRY and
A2 is HARRY
have no answer?

Since EQN is only defined for numeric atoms.

---

6.51) Try to write a function
(EQAN A1 A2)
which is True if A1 and A2 are the same atom.

```
(EQAN (LAMBDA (A1 A2)
    (COND
    ((AND (NUMBERP A1)(NUMBERP A2))(EQN A1 A2))
    ((OR (NUMBERP A1)(NUMBERP A2)) F)
    (T (EQ A1 A2))
        ) ))
```

---

6.52) (EQUAL S1 S2), where
S1 is BANANAS, and
S2 is (BANANAS)

F.

---

6.53) (EQUAL S1 S2), where
S1 is (STRAWBERRY ICE CREAM), and
S2 is (STRAWBERRY ICE CREAM)

T.

---

6.54) (EQUAL S1 S2), where
S1 is (STRAWBERRY ICE CREAM), and
S2 is (STRAWBERRY CREAM ICE)

F.

---

6.55) (EQUAL S1 S2), where
S1 is (BANANA ((SPLIT))), and
S2 is ((BANANA)(SPLIT))

F.

---

6.56) (EQUAL S1 S2), where
S1 is (BEEF ((SAUSAGE))(AND (SODA))), and
S2 is (BEEF ((SALAMI))(AND (SODA)))

F.

---

6.57) (EQUAL S1 S2), where
S1 is (5 APPLE PIES), and
S2 is (5 APPLE PIES)

T.

---

6.58) Can you write in your own words what (EQUAL S1 S2) does?

EQUAL determines if the S-expression S1 is exactly the same as the S-expression S2.

---

6.59) Using EQAN, write (EQUAL S1 S2)

```
(EQUAL (LAMBDA (S1 S2)
    (COND
    ((AND (NOT (ATOM S1))(NOT (ATOM S2)))
        (AND (EQUAL (CAR S1)(CAR S2))
        (EQUAL (CDR S1)(CDR S2))))
    ((AND (ATOM S1)(ATOM S2))(EQAN S1 S2))
    (T F)
            ) ))
```

---

6.60) (EQUAL S1 S2), where
S1 is ( ), and
S2 is ( )

Your response should be no answer. However, (ATOM S) where S is ( ) is True! Hence, (EQ S1 S2) and (EQUAL S1 S2) where S1 is ( ) and S2 is ( ) are also True. The null list ( ) is also referred to as the atom NIL.†

---

6.61) Is EQUAL a "star-" function?

Yes.

---

6.62) How would REMBER change if we replaced EQ by EQUAL?

```
(REMBER (LAMBDA (S L)
    (COND
    ((NULL L)( ))
    ((EQUAL (CAR L) S)(CDR L))
    (T (CONS (CAR L)(REMBER S (CDR L))))
          ) ))
```

Now REMBER removes the first S-expression S in the list L, instead of the first atom in the list of atoms L.

---

6.63) Is REMBER a "star-" function?

No.

---

6.64) Why not?

Because REMBER only recurses with the (CDR L).

---

6.65) Can we assume that all functions which were written using EQ and EQN can be generalized by replacing EQ and EQN by the function EQUAL.

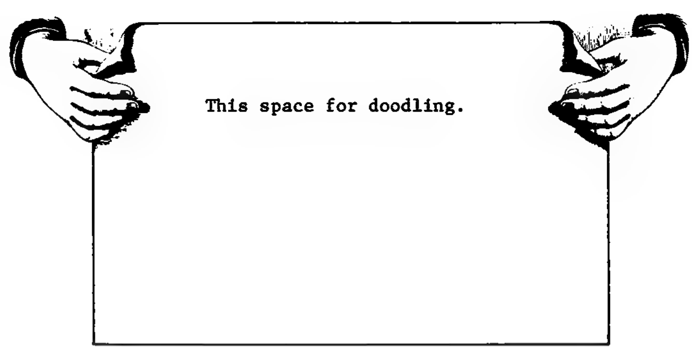
Not quite,
this won't work for EQAN or EQN, but will work for all others. In fact, disregarding the trivial examples of EQAN and EQN, that is exactly what we shall assume.

---

†(NULL S) can actually be written as

```
(NULL (LAMBDA (S)
    (COND
    ((ATOM S)(EQ S NIL))
    (T F)
      ) ))
```

If you got through that last function,
you need a rest.



| | | |
|---|---|---|
| 7.1) | Is this a <u>set</u>?<br>(APPLE PEACHES APPLE PLUM) | F,<br>since APPLE appears more than once. |
| 7.2) | (ISSET LAT), where<br>LAT is (APPLES PEACHES PEARS PLUMS) | T,<br>because no atom appears more than once. |
| 7.3) | (ISSET LAT), where<br>LAT is ( ) | T,<br>because no atom appears more than once. |

7.4) Try to write (ISSET LAT)

```
(ISSET (LAMBDA (LAT)
    (COND
    ((NULL LAT) T)
    ((MEMBER (CAR LAT)(CDR LAT)) F)
    (T (ISSET (CDR LAT)))
          ) ))
```

| | | |
|---|---|---|
| 7.5) | Does this work for the example<br>(APPLE 3 PEAR 4 9 APPLE 3 4) | Yes,<br>since MEMBER is now written using EQUAL instead of EQ. See 6.65. |
| 7.6) | Were you surprised to see the function MEMBER appear in the function ISSET? | You should not be, because we have written (MEMBER A LAT) already, and now we can use it whenever we like. |
| 7.7) | What is (MAKESET LAT), where<br>LAT is (APPLE PEACH PEAR PEACH PLUM APPLE LEMON PEACH) | (APPLE PEACH PEAR PLUM LEMON) |

7.8) Try to write (MAKESET LAT), using MEMBER.

```
(MAKESET (LAMBDA (LAT)
    (COND
    ((NULL LAT)( ))
    ((MEMBER (CAR LAT)(CDR LAT))(MAKESET (CDR LAT)))
    (T (CONS (CAR LAT)(MAKESET (CDR LAT))))
           ) ))
```

| | | |
|---|---|---|
| 7.9) | Using the function definition that you just wrote, what is the result of (MAKESET LAT), where<br>LAT is (APPLE PEACH PEAR PEACH PLUM APPLE LEMON PEACH) | (PEAR PLUM APPLE LEMMON PEACH) |

7.10) Try to write (MAKESET LAT), using MULTIREMBER.

```
(MAKESET (LAMBDA (LAT)
    (COND
    ((NULL LAT)( ))
    (T (CONS (CAR LAT)(MAKESET (MULTIREMBER
        (CAR LAT)(CDR LAT)))))
           ) ))
```

| | | |
|---|---|---|
| 7.11) | What is the result of (MAKESET LAT) using this second definition, where<br>LAT is (APPLE PEACH PEAR PEACH PLUM APPLE LEMON PEACH) | (APPLE PEACH PEAR PLUM LEMON) |
| 7.12) | Can you describe in your own words how the second definition of (MAKESET LAT) does what it does? | Here are our words:<br>"MAKESET saves the first atom in the LAT, and then recurses, after removing all occurrences of the first atom from the rest of the LAT". |
| 7.13) | Does the second MAKESET work for the example<br>(APPLE 3 PEAR 4 9 APPLE 3 4) | Yes,<br>since MULTIREMBER is now written using EQUAL instead of EQ. |

7.14) What is (SUBSET SET1 SET2), where
SET1 is (5 CHICKEN WINGS), and
SET2 is (5 HAMBURGERS 2 PIECES FRIED CHICKEN AND LIGHT DUCKLING WINGS)

T,
because each atom in SET1 is also in SET2.

---

7.15) What is (SUBSET SET1 SET2), where
SET1 is (4 POUNDS OF HORSERADISH), and
SET2 is (FOUR POUNDS CHICKEN AND 5 OUNCES HORSERADISH)

F.

---

7.16) Try to write (SUBSET SET1 SET2).

```
(SUBSET (LAMBDA (SET1 SET2)
    (COND
    ((NULL SET1) T)
    ((MEMBER (CAR SET1) SET2)(SUBSET (CDR SET1) SET2))
    (T F)
              ) ))
```

or, you could have written:

```
(SUBSET (LAMBDA (SET1 SET2)
    (COND
    ((NULL SET1) T)
    (T (AND (MEMBER (CAR SET1) SET2)(SUBSET
        (CDR SET1) SET2)))
          ) ))
```

---

7.17) What is (EQSET SET1 SET2), where
SET1 is (6 LARGE CHICKENS WITH WINGS)
SET2 is (6 CHICKENS WITH LARGE WINGS)

T.

---

7.18) Try to write (EQSET SET1 SET2).

```
(EQSET (LAMBDA (SET1 SET2)
    (COND
    ((SUBSET SET1 SET2)(SUBSET SET2 SET1))
    (T F)
              ) ))
```

or, you could have written:

```
(EQSET (LAMBDA (SET1 SET2)
    (COND
    (T (AND (SUBSET SET1 SET2)(SUBSET SET2 SET1)))
          ) ))
```

or, you could have written the "one-liner"

```
(EQSET (LAMBDA (SET1 SET2)
    (AND (SUBSET SET1 SET2)(SUBSET SET2 SET1))
          ))
```

---

7.19) (DOESINTERSECT SET1 SET2), where
SET1 is (STEWED TOMATOES AND MACARONI), and
SET2 is (MACARONI AND CHEESE)

T,
because at least one atom in SET1 is in SET2.

7.20) Try to write (DOESINTERSECT SET1 SET2).

```
(DOESINTERSECT (LAMBDA (SET1 SET2)
    (COND
    ((NULL SET1) F)
    ((MEMBER (CAR SET1) SET2) T)
    (T (DOESINTERSECT (CDR SET1) SET2))
          ) ))
```

or, you could have written:

```
(DOESINTERSECT (LAMBDA (SET1 SET2)
    (COND
    ((NULL SET1) F)
    (T (OR (MEMBER (CAR SET1) SET2)(DOESINTERSECT
        (CDR SET1) SET2)))
          ) ))
```

Note: Look back at (SUBSET SET1 SET2), and compare for similarities.

---

7.21) What is (INTERSECT SET1 SET2), where SET1 is (STEWED TOMATOES AND MACARONI), and SET2 is (MACARONI AND CHEESE)

(AND MACARONI)

---

7.22) Try to write (INTERSECT SET1 SET2)

```
(INTERSECT (LAMBDA (SET1 SET2)
    (COND
    ((NULL SET1)( ))
    ((MEMBER (CAR SET1) SET2)(CONS (CAR SET1)
        (INTERSECT (CDR SET1) SET2)))
    (T (INTERSECT (CDR SET1) SET2))
        )  ))
```

or, you could have written

```
(INTERSECT (LAMBDA (SET1 SET2)
    (COND
    ((NULL SET1)( ))
    ((NOT (MEMBER (CAR SET1) SET2))(INTERSECT
        (CDR SET1) SET2))
    (T (CONS (CAR SET1)(INTERSECT (CDR SET1) SET2)))
        )  ))
```

---

7.23) What is (UNION SET1 SET2), where SET1 is (STEWED TOMATOES AND MACARONI CASSEROLE), and SET2 is (MACARONI AND CHEESE)

(STEWED TOMATOES CASSEROLE MACARONI AND CHEESE)

---

7.24) Try to write (UNION SET1 SET2).

```
(UNION (LAMBDA (SET1 SET2)
    (COND
    ((NULL SET1) SET2)
    ((MEMBER (CAR SET1) SET2)(UNION (CDR SET1) SET2))
    (T (CONS (CAR SET1)(UNION (CDR SET1) SET2)))
        )  ))
```

---

7.25) What does this function do?

```
(XXX (LAMBDA (SET1 SET2)
    (COND
    ((NULL SET1)( ))
    ((MEMBER (CAR SET1) SET2)(XXX (CDR SET1) SET2))
    (T (CONS (CAR SET1)(XXX (CDR SET1) SET2)))
        )  ))
```

In our words:
" The value of (XXX SET1 SET2) is all the atoms in SET1 that are not in SET2."
(STEWED TOMATOES CASSEROLE)
Note: The function (XXX SET1 SET2) can be called (COMPLEMENT SET1 SET2)

---

7.26) Is this a <u>pair</u>?
(PEAR PEAR)

T,
because it is a list with only two atoms.

---

7.27) Is this a pair?
(3 7)

T.

---

7.28) Is this a pair?
(3 PAIR)

T.

---

7.29) How can you get a hold of the first atom of a pair?

By taking the CAR of the pair.

---

7.30) How can you get a hold of the second atom of a pair?

By taking the CAR of the CDR of the pair.

---

7.31) If you are given two atoms, A1 and A2, how do you make them a pair?

You CONS the first atom onto the CONS of the second atom onto ( ).
(CONS A1 (CONS A2( )))

---

7.32)

```
(FIRST (LAMBDA (P)
    (COND
    (T (CAR P))
        )  ))
```

```
(SECOND (LAMBDA (P)
    (COND
    (T (CAR (CDR P)))
        )  ))
```

```
(BUILD (LAMBDA (A1 A2)
    (COND
    (T (CONS A1 (CONS A2 ( ))))
        )  ))
```

What possible uses do these three functions have?

They will be used for <u>convenience</u> and <u>readability</u>, as you will soon see.

---

7.33) Is L a REL, where
L is (APPLES PEACHES PUMPKIN PIE)

F,
since L is not a list of pairs.
Note: REL stands for RELation.

---

7.34) Is L a REL, where
L is ((APPLES PEACHES)(PUMPKIN PIE)(APPLES PEACHES))

F,
since L is not a set of pairs.

---

7.35) Is L a REL, where
L is ((APPLES PEACHES)(PUMPKIN PIE))

T.

---

7.36) Is L a REL, where
L is ((4 3)(4 2)(7 6)(6 2)(3 4))

T.

---

7.37) Is REL a FUN, where
REL is ((4 3)(4 2)(7 6)(6 2)(3 4))

F,
Note: FUN stands for FUNction.

---

7.38) What is (ISFUN REL), where
REL is ((12 3)(4 2)(7 6)(6 2)(3 4))

T,
because (FIRSTS REL) is a set - see Chapter 3.

---

7.39) Try to write (ISFUN REL).

How about this?

```
(ISFUN (LAMBDA (REL)
    (COND
    ((NULL REL) T)
    ((MEMBER* (FIRST (CAR REL))(CDR REL)) F)
    (T (ISFUN (CDR REL)))
       ) ))
```

---

7.40) When will that function description for ISFUN work?

When (DOESINTERSECT (FIRSTS REL)(SECONDS REL)) is F.

---

7.41) Try again to write (ISFUN REL) so it will work for the case where
REL is ((12 3)(4 2)(7 6)(6 2)(3 4))

```
(ISFUN (LAMBDA (REL)
    (COND
    ((NULL REL) T)
    ((MEMBER (FIRST (CAR REL))(FIRSTS (CDR REL))) F)
    (T (ISFUN (CDR REL)))
       ) ))
```

or, much better

```
(ISFUN (LAMBDA (REL)
    (COND
    (T (ISSET (FIRSTS REL)))
       ) ))
```

---

7.42) What is (REVREL REL), where
REL is ((8 A)(PUMPKIN PIE)(GOT SICK))

((A 8)(PIE PUMPKIN)(SICK GOT))

---

7.43) Try to write (REVREL REL).

```
(REVREL (LAMBDA (REL)
    (COND
    ((NULL REL)( ))
    (T (CONS (BUILD (SECOND (CAR REL))(FIRST (CAR REL)))
        (REVREL (CDR REL))))
        ) ))
```

or, the following would also be correct:

```
(REVREL (LAMBDA (REL)
    (COND
    ((NULL REL)( ))
    (T (CONS (CONS (CAR (CDR (CAR REL)))
        (CONS (CAR (CAR REL))( )))(REVREL (CDR REL
         ) ))
```

Note: Now you know what we mean by convenience and readability.

---

7.44) Can you guess why FUN is not a FULLFUN, where
FUN is ((12 3)(4 2)(7 6)(6 2)(3 4))

FUN is not a FULLFUN, since the 2 appears more than once as a second atom of a pair.

---

7.45) Why does (ISFULLFUN FUN) have the value T, where
FUN is ((12 3)(4 8)(7 6)(6 2)(3 4))

Because the list (3 8 6 2 4) is a set.

---

7.46) What is (ISFULLFUN FUN), where
FUN is ((GRAPE RAISIN)(PLUM PRUNE)(STEWED PRUNE))

F.

---

7.47) What is (ISFULLFUN FUN), where
FUN is ((GRAPE RAISIN)(PLUM PRUNE)(STEWED GRAPE))

T,
because the list (RAISIN PRUNE GRAPE) is a set.

---

7.48) Try to write (ISFULLFUN FUN) three different ways.

1.
```
(ISFULLFUN (LAMBDA (FUN)
    (COND
    ((NULL FUN) T)
    ((MEMBER (SECOND (CAR FUN))(SECONDS (CDR FUN))) F)
    (T (ISFULLFUN (CDR FUN)))
              ) ))
```

2.
```
(ISFULLFUN (LAMBDA (FUN)
    (COND
    (T (ISSET (SECONDS FUN)))
              ) ))
```

3.
```
(ISFULLFUN (LAMBDA (FUN)
    (COND
    (T (ISFUN (REVREL FUN)))
              ) ))
```

---

7.49) What is another function name for
(ISFULLFUN FUN)

(ONE-TO-ONE FUN).

---

If the three ways you just wrote that last function were:

1. Sitting down

2. Standing up

3. Standing on your head

you were wrong!

## CHAPTER 8: "HELP" IS ON THE WAY, OR WELCOME TO THE HAMMOCK

8.1) Is this a palindrome?
(APPLE BANANA BANANA APPLE)

Yes.

---

8.2) (ISPALIN L), where
L is (A 5 4 M K M 4 5 A)

T.

---

8.3) (ISPALIN L), where
L is (HARRY)

T.

---

8.4) (ISPALIN L), where
L is (H A R R Y)

F.

---

8.5)

```
(ISPALIN (LAMBDA (L)
    (COND
    ((NULL L) T)
    ((NULL (CDR L)) T)
    ((EQUAL (CAR L)(RAC L))(ISPALIN (CDR (RDC L))))
    (T F)
         ) ))
```

What does this function do?

The function finds if L is a palindrome.

---

8.6) What is (RAC L) ?

(RAC L) is a help function. We don't know yet what it looks like, but we know it will be a function that will give us the last atom in a list.

---

8.7) Have we used any help functions before in this book?

Yes,
every time we used a function other than CAR, CDR, CONS, EQ, ATOM, NULL, AND, OR, NOT, NUMBERP, ZEROP, ADD1, and SUB1 to define another function, we were using help functions!

---

8.8) Can you name some help functions we used in the last chapter?

FIRST, SECOND, BUILD, FIRSTS, SECONDS, MEMBER, EQUAL, SUBSET, ISSET, and MULTIREMBER. Can you name some other functions we failed to mention?

---

8.9) What is (RDC L) ?
Note: "RDC" is pronounced "Rudercouder".

RDC is a help function. We don't know yet what it looks like, but we know it will be a function that will give us all the atoms in the list, without the RAC.

---

8.10) Shall we write RAC and RDC now?

No.
ISPALIN is written. We know what RAC and RDC do, so we can write them later.

---

8.11) What is (REVLAT LAT), where
LAT is (A BUNCH OF COCONUTS)

(COCONUTS OF BUNCH A)

---

8.12) Here is one way to write (REVLAT LAT):

```
(REVLAT (LAMBDA (LAT)
    (COND
    ((NULL LAT)( ))
    (T (SNOC (REVLAT (CDR LAT))(CAR LAT)))
         ) ))
```

Is this correct?

Yes.

---

8.13) What does the function (SNOC L S) do?

SNOC is a help function. We do not know what it looks like yet, but we know we need a function that sticks S-expression onto the end of a list, similar to the way CONS sticks S-expressions onto the front of a list. So for now, we will assume SNOC exists, and we will write it later.

---

COMMANDMENT No. 7

<u>Thou shalt always assume that any help function you need does exist. If you can describe what the help function does, then use it, and you can write it later.</u>

| | | |
|---|---|---|
| 8.14) | Is this a MAT?<br>((4 8 6 3)(12 4 86 2)(36 1 7 6)) | Yes,<br>because it is a list of VECs.<br>Note: MAT stands for MATrix. |
| 8.15) | Is this a MAT?<br>((16 3 2)(4 7 6)(14 8)) | No,<br>since not all the VECs are of the same length. |
| 8.16) | Is this a ZEROMAT?<br>((0 0 0)(0 1 0)(0 0 0)(0 0 0)) | No,<br>since not all the atoms of every VEC are zero. |
| 8.17) | (ISZEROMAT MAT), where<br>MAT is ((0 0 0 0 0 0)(0 0 0 0 0 0)(0 0 0 0 0 0)) | Yes,<br>because all the VECs are ZEROVECs. |

8.18) Try to write (ISZEROMAT MAT).

```
(ISZEROMAT (LAMBDA (MAT)
    (COND
    ((NULL MAT) T)
    (T (AND (ISZEROVEC (CAR MAT))(ISZEROMAT (CDR MAT))))
            ) ))
```

or, you could have written

```
(ISZEROMAT (LAMBDA (MAT)
    (COND
    ((NULL MAT) T)
    ((ISZEROVEC (CAR MAT))(ISZEROMAT (CDR MAT)))
    (T F)
            ) ))
```

| | | |
|---|---|---|
| 8.19) | What is (ISZEROVEC VEC) ? | It is a help function. |
| 8.20) | What does (ISZEROVEC VEC) do? | It asks if a VEC is the ZEROVEC - if all the atoms in the VEC are zero. |
| 8.21) | (ISZEROVEC VEC), where<br>VEC is (0 0 0) | T. |
| 8.22) | (ISZEROVEC VEC), where<br>VEC is (0 0 0 0) | T. |
| 8.23) | (ISZEROVEC VEC), where<br>VEC is (0 0) | T. |
| 8.24) | Is this a MAT?<br>((0 0 0)(0 0 0 0)(0 0)) | No,<br>since all the VECs are not of the same length. |
| 8.25) | What would be the value of (ISZEROMAT L), where<br>L is ((0 0 0)(0 0 0 0)(0 0)) | T,<br>because, even though L is not a MAT, ISZEROMAT only asks if each VEC is a ZEROVEC by considering each VEC in total isolation. |
| 8.26) | What is (SCALMAT N MAT), where<br>N is 3, and<br>MAT is ((6 3 2)(5 12 7)(3 2 6)) | ((18 9 6)(15 36 21)(9 6 18)) |
| 8.27) | What is (SCALMAT N MAT), where<br>N is 56, and<br>MAT is ((1)(2)(5)(7)(4)) | ((56)(112)(280)(392)(224)) |

8.28) Can you write (SCALMAT N MAT) ?

```
(SCALMAT (LAMBDA (N MAT)
    (COND
    ((NULL MAT)( ))
    (T (CONS (SCALVEC N (CAR MAT))(SCALMAT N (CDR MAT))))
            ) ))
```

8.29) What is (SCALVEC N VEC) ?

It is a help function.

---

8.30) What will (SCALVEC N VEC) do?

It will multiply each atom in the argument VEC by the argument N.

---

8.31) Is L a MAT, where
L is ((6 3 4)(8 2)(12 13 7))

No.

---

8.32) What is the value of (SCALMAT N L), where
N is 5, and
L is ((6 3 4)(8 2)(12 13 7))

((30 15 20)(40 10)(60 65 35))
L is not a MAT, but the help function (SCALVEC N VEC) considers each VEC one at a time - in total isolation.

---

8.33) What is (MATPLUS MAT1 MAT2), where
MAT1 is ((4 4 6)(3 12 15)(6 8 3)(4 4 6)), and
MAT2 is ((8 8 6)(9 0 3)(6 4 9)(8 8 6))

((12 12 12)(12 12 18)(12 12 12)(12 12 12))

---

8.34) What is (MATPLUS MAT1 MAT2), where
MAT1 is ((0 5 6 4)(3 2 1 6))
MAT2 is ((4 6 3 2)(8 12 3 6))

((4 11 9 6)(11 14 4 12))

---

8.35) Can you write (MATPLUS MAT1 MAT2) ?

```
(MATPLUS (LAMBDA (MAT1 MAT2)
    (COND
    ((NULL MAT1)( ))
    (T (CONS (VECPLUS (CAR MAT1)(CAR MAT2))
        (MATPLUS (CDR MAT1)(CDR MAT2))))
            ) ))
```

---

8.36) What is (DIMENSIONS MAT), where
MAT is ((4 6)(8 3))

(2 2)

---

8.37) What is (DIMENSIONS MAT), where
MAT is ((6 9 3 4 12 2 37)(4 12 6 7 84 3 172))

(2 7)

---

8.38) What is (DIMENSIONS MAT), where
MAT is ((4)(6)(5)(4)(2)(9)(12)(14)(26)(3)(1))

(11 1)

---

8.39) What is (DIMENSIONS MAT), where
MAT is ((4 6 5 4 29 12 14 26 31))

(1 9)

---

8.40) What is (DIMENSIONS MAT), where
MAT is (( )( )( )( )( )( )( ))

(7 0)

---

8.41) Try to write (DIMENSIONS MAT).
Use any help functions you need.

```
(DIMENSIONS (LAMBDA (MAT)
    (COND
    (T (BUILD (LENGTH MAT)(LENGTH (CAR MAT))))
          ) ))
```

or, you could have written

```
(DIMENSIONS (LAMBDA (MAT)
    (COND
    (T (CONS (LENGTH MAT)(CONS (LENGTH (CAR MAT))( ))))
          ) ))
```

---

8.42) What is (SORTUP VEC), where
VEC is (3 9 4 8 6)

(3 4 6 8 9)

---

8.43 What is (SORTUP VEC), where
VEC is (2 15 9 2 11 2)

(2 2 2 9 11 15)

---

8.44) Try to write (SORTUP VEC), using any help functions you need.

```
(SORTUP (LAMBDA (VEC)
    (COND
    ((NULL VEC)( ))
    (T (CONS (MINIVEC VEC)(SORTUP (REMBER (MINIVEC VEC)
         VEC))))
          ) ))
```

---

8.45) What does our help function (MINIVEC VEC) do?

It finds the smallest number in the VEC. Although we have not seen (MINIVEC VEC) yet, writing it later should be trivial - remember MAXVEC in Chapter 4?

---

8.46) What is (MINSMAT MAT), where
MAT is ((3 6 9)(4 2 0))

(3 0)

---

8.47) What is (MINSMAT MAT), where
MAT is ((2 6 7 9)(11 1 3 4)(1 9 12 15))

(2 1 1)

---

8.48) Now try to write (MINSMAT MAT).

```
(MINSMAT (LAMBDA (MAT)
    (COND
    ((NULL MAT)( ))
    (T (CONS (MINIVEC (CAR MAT))(MINSMAT (CDR MAT))))
            ) ))
```

---

8.49) What does (MINSMAT MAT) do?

(MINSMAT MAT) builds a VEC composed of the smallest number in each VEC of the MAT.

---

8.50) Here is the function (NONAME MAT); what does it do?

```
(NONAME (LAMBDA (MAT)
    (COND
    ((NULL (CAR MAT))( ))
    (T (CONS (MINIVEC (FIRSTS MAT))(NONAME(CDRVEX
       MAT))))
        ) ))
```

We are not sure, but let's find out.

---

8.51) What does (MINIVEC (FIRSTS MAT)) do?

It finds the smallest atom in the list composed of the CAR of each VEC of the MAT.

---

8.52) What is (MINIVEC (FIRSTS MAT)), where
MAT is ((4 8 2)(6 12 2)(3 6 8))

3

---

8.53) What is the meaning of (NONAME (CDRVEX MAT)) ?

The argument (CDRVEX MAT) should give us a list composed of the CDR of each VEC of the MAT. With this, we recurse.

---

8.54) What is (CDRVEX MAT), where
MAT is ((4 8 3)(6 12 2)(3 6 8))

((8 3)(12 2)(6 8))

---

8.55) What is (CARTES SET1 SET2), where
SET1 is (HOME MADE BREAD), and
SET2 is (BROWN RICE)

((HOME BROWN)(HOME RICE)(MADE BROWN)(MADE RICE)
(BREAD BROWN)(BREAD RICE))

---

8.56) What is (CARTES SET1 SET2), where
SET1 is (A 5 9 CORN), and
SET2 is (HELP 3 TIMES)

((A HELP)(A 3)(A TIMES)(5 HELP)(5 3)(5 TIMES)
(9 HELP)(9 3)(9 TIMES)(CORN HELP)(CORN 3)
(CORN TIMES))

---

8.57) Can you write in your own words what
(CARTES SET1 SET2)
does?

Our words:
"CARTES builds a REL composed of all possible pairs choosing first elements from SET1 and second elements from SET2."

8.58) Now write (CARTES SET1 SET2).

```
(CARTES (LAMBDA (SET1 SET2)
    (COND
    ((NULL SET1)( ))
    (T (APPEND (DISTRIB (CAR SET1) SET2)(CARTES
        (CDR SET1) SET2)))
            ) ))
```

---

8.59) What is (INTERSECTALL LSET), where
LSET is ((A B C)(C A D E)(E F G H A B)))

(A)

---

| | | |
|---|---|---|
| 8.60) | What is (INTERSECTALL LSET), where<br>LSET is ((6 PEARS AND)(3 PEACHES AND 6 PEPPERS)<br>(8 PEARS AND 5 PLUMS)(AND 6 PRUNES WITH LOTS<br>OF APPLES)) | (6 AND) |
| 8.61) | Now, using whatever help functions you need, write<br>(INTERSECTALL LSET). | (INTERSECTALL (LAMBDA (LSET)<br>(COND<br>((NULL LSET)( ))<br>(T (INTERSECT (CAR LSET)(INTERSECTALL (CDR LSET))))<br>) )) |
| 8.62) | What is (TRANSPOSE MAT), where<br>MAT is ((4 3 2 1)(2 7 6 8)(9 1 4 3)(4 2 6 7)) | ((4 2 9 4)(3 7 1 2)(2 6 4 6)(1 8 3 7)) |
| 8.63) | What is (TRANSPOSE MAT), where<br>MAT is ((1 2 3)(1 2 3)(1 2 3)(1 2 3)) | ((1 1 1 1)(2 2 2 2)(3 3 3 3)) |
| 8.64) | In your own words, what does (TRANSPOSE MAT) do? | Our words, again:<br>"TRANSPOSE builds a new MAT. The first VEC in this new MAT is (FIRSTS MAT); the second VEC is (SECONDS MAT), and so on." |
| 8.65) | Try to write (TRANSPOSE MAT). | How about:<br>(TRANSPOSE (LAMBDA (MAT)<br>(COND<br>((NULL MAT)( ))<br>(T (CONS (FIRSTS MAT)(TRANSPOSE (CDRVEX MAT))))<br>) )) |
| 8.66) | Is that correct? | Well . . . let's see. |
| 8.67) | (NULL MAT), where<br>MAT is ((3 2 4)(6 9 7)) | No. |
| 8.68) | (T (CONS (FIRSTS MAT)(TRANSPOSE (CDRVEX MAT)))) | CONS (3 6) onto (TRANSPOSE (CDRVEX MAT)) |
| 8.69) | What is (CDRVEX MAT) ? | ((2 4)(9 7)) |
| 8.70) | (NULL MAT) | No. |
| 8.71) | (T (CONS (FIRSTS MAT)(TRANSPOSE (CDRVEX MAT)))) | CONS (2 9) onto (TRANSPOSE (CDRVEX MAT)) |
| 8.72) | What is (CDRVEX MAT) ? | ((4)(7)) |
| 8.73) | (NULL MAT) | No. |
| 8.74) | (T (CONS (FIRSTS MAT)(TRANSPOSE (CDRVEX MAT)))) | CONS (4 7) onto (TRANSPOSE (CDRVEX MAT)) |
| 8.75) | What is (CDRVEX MAT) | (( )( )) |
| 8.76) | (NULL MAT) | No. |
| 8.77) | What? | Since each VEC of the MAT is the same length, we have now considered all the atoms of each VEC, but the MAT is not null; it is composed of null VECs. We obviously need a terminal condition to test when the VECs of the MAT are null. |
| 8.78) | Can you come up with something that would be a correct terminal condition line? | ((NULL (CAR MAT))( )) |

8.79) Now what is the correct function definition of (TRANSPOSE MAT)

```
(TRANSPOSE (LAMBDA (MAT)
    (COND
    ((NULL (CAR MAT))( ))
    (T (CONS (FIRSTS MAT)(TRANSPOSE (CDRVEX MAT))))
        ) ))
```

8.80) Now can you rewrite (NONAME MAT) using (TRANSPOSE MAT) and (MINSMAT MAT) as help functions?

```
(NONAME (LAMBDA (MAT)
    (COND
    (T (MINSMAT (TRANSPOSE MAT)))
        ) ))
```

8.81) Now was that more straightforward?

Correct.

8.82) This could be The End of the book, but we decided to give you all the help functions we have not yet defined. They are RAC, RDC, SNOC, MINIVEC, CDRVEX, APPEND, and DISTRIB. Do we need to refer back to the functions that used these help functions in order to write them?

No,
you must write the help functions in total isolation, that is, you need to know only two things:
1. The arguments of the help function;
2. What the help function should do.

COMMANDMENT No. 8

Thou shalt, when writing a function which requires one or more help functions, complete the main function first. Make a note of the arguments and a description of what the help function should do. LATER, write the help function in total isolation from the main function.

8.83) Write the description of the function (RAC L), which
1. Takes a non-null list as an argument, and
2. Has a final value which is the last S-expression in the list.

```
(RAC (LAMBDA (L)
    (COND
    ((NULL (CDR L))(CAR L))
    (T (RAC (CDR L)))
        ) ))
```

8.84) Write the description of the function (RDC L), which
1. Takes a non-null list as an argument, and
2. Has a final value which is the same L, but without (RAC L).

```
(RDC (LAMBDA (L)
    (COND
    ((NULL (CDR L))( ))
    (T (CONS (CAR L)(RDC (CDR L))))
        ) ))
```

8.85) Write the description of the function (SNOC L S), which
1. Takes a list and an S-expression as its arguments, and
2. Builds a list by sticking its S-expression argument onto the end of its list argument.

```
(SNOC (LAMBDA (L S)
    (COND
    ((NULL L)(CONS S ( )))
    (T (CONS (CAR L)(SNOC (CDR L))))
        ) ))
```

8.86) Write the description of the function (MINIVEC VEC) which
1. Takes a non-null VEC as its argument, and
2. Has as its final value the smallest atom in the VEC.

```
(MINIVEC (LAMBDA (VEC)
    (COND
    ((NULL (CDR VEC))(CAR VEC))
    (T (SMALLER (CAR VEC)(MINIVEC (CDR VEC))))
        ) ))
```

Sometime we need to write the function
(SMALLER N1 N2)
which
1. Takes two numbers as its arguments, and
2. Has the value of the smaller of the two numbers:

```
(SMALLER (LAMBDA (N1 N2)
    (COND
    ((LESSP N1 N2) N1)
    (T N2)
        ) ))
```

8.87) Write a description of the function (CDRVEX MAT), which
1. Takes a MAT as its argument, and
2. Has as its final value a list of the CDRs of each VEC of the MAT.

```
(CDRVEX (LAMBDA (MAT)
    (COND
    ((NULL MAT)( ))
    (T (CONS (CDR (CAR MAT))(CDRVEX (CDR MAT))))
        ) ))
```

---

8.88) Now write the description of the function (APPEND L1 L2), which
1. Takes two lists as its arguments, and
2. Has as its final value one list containing all the S-expressions of L1 and L2, in their original orders.

Example: where
L1 is (APPLES BANANAS CHICKENS), and
L2 is (DOGS EGGS FRUIT GRAPE)
then
(APPEND L1 L2) is
(APPLES BANANAS CHICKENS DOGS EGGS FRUIT GRAPE)

```
(APPEND (LAMBDA (L1 L2)
    (COND
    ((NULL L1) L2)
    (T (CONS (CAR L1)(APPEND (CDR L1) L2)))
        ) ))
```

---

8.89) Now write the description of the function (DISTRIB A SET), which
1. Takes an atom and a set as arguments, and
2. Builds a REL where each pair has A as its first atom, and a member of the set as its second atom.

```
(DISTRIB (LAMBDA (A SET)
    (COND
    ((NULL SET)( ))
    (T (CONS (BUILD A (CAR SET))(DISTRIB A (CDR SET))))
        ) ))
```

---

8.90) Now write the description of the function (ISZEROVEC VEC), which
1. Takes a VEC as its argument, and
2. Has as its value F if the vector contains some number other than 0.

```
(ISZEROVEC (LAMBDA (VEC)
    (COND
    ( (NULL VEC) T)
    ((ZEROP (CAR VEC))(ISZEROVEC (CDR VEC)))
    (T F)
        ) ))
```

---

8.91) Now write the description of the function (SCALVEC N VEC), which
1. Takes a number and a VEC as its argument, and
2. Builds a VEC with each value multiplied by N.

Example: where
N is 12, and
VEC is (2 8 15)
then
(SCALVEC N VEC) is
(24 96 180)

```
(SCALVEC (LAMBDA (N VEC)
    (COND
    ((NULL VEC)( ))
    (T (CONS (TIMES N (CAR VEC))
        (SCALVEC N (CDR VEC))))
        ) ))
```

---

8.92 Now that you have come this far, here is a real brainteaser. Write the function (DEPTH L).
(DEPTH L) is 3, where L is ((A) (((B) C)) D)
is 5, where L is (((((A))))) A ((B)))
is 1, where L is ((ATE) TOO (MUCH))
is 0, where L is (A B C)
This is a hammock problem, so give yourself some time.

```
(DEPTH (LAMBDA (L)
    (COND
    ((NULL L) 0)
    ((ATOM (CAR L))(DEPTH (CDR L)))
    ((GREATERP (ADD1 (DEPTH (CAR L)))(DEPTH (CDR L)))
        (ADD1 (DEPTH (CAR L))))
    (T (DEPTH (CDR L)))
        ) ))
```

---

You have reached the end of your beginning with LISP. What should you have learned? Are you now ready to tackle a major programming problem in LISP†? You are actually better prepared than you realize, but it would be worth your time to develop a fuller understanding of all the capabilities in LISP. For those of you who are intrigued with computer programming and would like to learn more about LISP, the books in the references, with the exception of Suppes [6] offer a complete coverage of the subject. Weissman [7], and Siklóssy [5] offer the next level of sophistication to the understanding of LISP. Maurer [3] is a good introduction to LISP for a competent computer programmer. Berkeley [1] and Minsky [4] have some interesting illustrations of LISP and McCarthy [2] is the definitive work on LISP from which these texts have evolved. For those others of you who do not foresee any further study of the subject, we certainly hope you have enjoyed the book and learned some interesting new things about symbol manipulation.

†The only function you must also be familiar with before you can interact with the computer is the function DEFINE. See page 15 of McCarthy [2] or page 67 of Weissman [6] for an example with DEFINE.

The Little LISPer is a programmer's guide aimed at the non-programmer. As such it offers an easily digested introduction to computers and symbolic and numeric processing. The goal of this book is largely to teach you a new way to think about and solve problems. LISP as a programming language is one way to implement your solution - but it is most important because the LISP experience teaches you this new and powerful technique for approaching and analyzing problems. LISP is a simple, elegant and powerful language; as such it is conceivable that (as one of the reviewers puts it) The Little LISPer (and therefore LISP) can be understood by "any logical human being from 8 to 80."

REFERENCES

[1] Berkeley, E. C., and Bobrow, D. G., (eds), The Programming Language LISP: Its Operation and Applications, Information International, Inc., Cambridge, Massachusetts, 1964.

[2] McCarthy, J., et al., LISP 1.5 Programmer's Manual, The M.I.T. Press, Cambridge, Massachusetts, 1963.

[3] Maurer, W. D., A Programmer's Introduction to LISP, American Elsevier, New York, New York, 1973.

[4] Minsky, M., (ed), Semantic Information Processing, The M.I.T. Press, Cambridge, Massachusetts, 1968.

[5] Siklóssy, L. S., Let's Talk LISP, Prentice-Hall, Englewood Cliffs, New Jersey, (in preparation).

[6] Suppes, P., Introduction to Logic, Van Nostrand Co., Princeton, New Jersey, 1957.

[7] Weissman, C., LISP 1.5 Primer, Dickenson Publishing Co., Belmont, California, 1968.

Reorder No. 13-2165